ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DOBLE INGENIERÍA DE TELECOMUNICACIÓN E INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PROYECTO FIN DE CARRERA

Moodle REST API Exploration

(Exploración de la API REST de Moodle)

Autor: Javier Benito García-Mochales

Tutor: Jesús M. González-Barahona

Curso académico 2012/2013

Abstract

This document contains the description and development process of the Moodle Web Services Python Library. In first place, the project was focused on exploring all Moodle interaction ways for discovering every one of its possible uses. The initial perspective is to investigate the chance of controlling the Moodle system from external applications. For it, the different Moodle versions were analyzed to see their offering chances. This clarified the possibilities of using Moodle in three main ways: with web browsers, with scrapping applications and with web services.

The Moodle web services will be studied for exploiting all its potential. In this study, diverse information will be picked up for activating and to configure those services. This study also revealed a powerful utility for doing Moodle management tasks through scripts. The external programs interacting way was discovered with information extraction of example codes from several sources.

This project also provides a full functional library of Moodle web services 2.5 for supporting applications development. This library was programmed in the Python programming language and provides a communications interface with all available web services functions.

This library has been fully tested with short scripts that can be taken as use examples. It also has been fully documented using Python data structures and taking as reference the official documentation.

One important met goal is to document all the process to use Moodle since its installation to its final use tests. Some steps taken along the project are documented in several different and not related sources. Sometimes this documentation is poor or confusing. This project compiles all this steps details and includes some missing documentation. This will help to Moodle users that want use this capability without wasting time looking for use guides.

Resumen

Este documento contiene la descripción y el proceso de desarrollo de la libraría Python de los servicios web de Moodle. En primer lugar, el proyecto se centra en explorar todas las formas de interacción con Moodle para descubrir cada uno de sus posibles usos. La perspectiva inicial es investigar la opción de controlar al sistema Moodle desde aplicaciones externas. Para ello se analizaron las diferentes versiones de Moodle para ver las oportunidades que ofrecían. Esto aclaró que Moodle se puede usar de tres formas principalmente: con navegadores web, con aplicaciones que usan scrapping y mediante servicios web.

Se estudiaran los servicios web de Moodle para explotar todo su potencial. En este estudio se recopilara diversa información para activar y configurar esos servicios. Este estudio también ha revelado una potente utilidad para hacer tareas de administración de Moodle mediante scripts. La forma de cómo interactúan las aplicaciones externas se descubrió extrayendo información de de diversos códigos de ejemplo obtenidos de diversas fuentes.

Este proyecto también proporciona una libraría completamente funcional de los servicios web de Moodle 2.5 para ayudar al desarrollo de aplicaciones. Esta librería ha sido programada en el lenguaje de programación Python y proporciona una interfaz de comunicaciones con todas las funcionalidades de los servicios web disponibles.

Esta librería ha sido completamente probada con scripts cortos que pueden ser tomados como ejemplos de uso. También ha sido completamente documentada utilizando las estructuras de datos de Python y tomando como referencia la documentación oficial.

Otro objetivo importante cumplido es la documentación de todo el proceso para usar Moodle, desde su instalación hasta las pruebas finales de uso. Algunos pasos dados a lo largo del proyecto están documentados en fuentes diferentes y no relacionadas. Este proyecto recopila todos los detalles de estos pasos e incluye alguna documentación no encontrada. Esto ayudará a los usuarios de Moodle que quieran utilizar esta capacidad sin perder el tiempo en buscar manuales de uso.

Index

Figures index

Tables index

1. Introduction

This chapter summarizes preliminary information that the reader should acquire for a complete understanding of this document.

Throughout this document the Moodle requirements, its main characteristics, its web services plug-in, the REST protocol sentences for interacting with web services, the python code to made that communication and diverse functionality provided by the developed python library will be explained.

1.1. Project motivation

Moodle has become a popular teaching support system all around the world. This and the increasing of mobile phone capacities to interact with many systems over the web have done to appear new applications for Moodle.

Because of this, Moodle developers decided to include some additional functionality to allow external applications to work with Moodle since version 2.0. These functions are implemented as a Moodle plug-in and its initial functionality was limited. Afterward, the available functions list was extended on version 2.5, so many things that Moodle can do with its web interface can be done through this external service (for example, creating courses, creating user groups or getting courses files).

However, using this Moodle functionality requires some advanced knowledge about Moodle functionality, REST systems or data exchange protocols for applications.

Connecting Moodle with other systems would open a new world of use possibilities, expanding the Moodle functionalities and a multi-platform interaction. This could extend the Moodle use fields, add functions already implemented in other systems, make Moodle usable from any device or simply improve its features. For example, synchronizing Moodle calendar entries with a personal calendar stored in a remote machine.

1.2. Project Objectives

This project tries to explore the Moodle web services utility and its potential uses. The necessary steps for being used will be explained and their utility will be shown with a generic example of use.

With this acquired knowledge, the project efforts will be focused on giving some tools to configure Moodle and getting this functionality easily with a library for programming development.
This library has been made to facilitate the development of applications that interact with Moodle and for exploiting this functionality.

However, Moodle system must be configured in order to do that the library works correctly. So a part of this memory is reserved to explain how to configure Moodle for working with web services and how to set up the library too.

Besides, to have user accounts in the Moodle site is necessary for using this library and the user permissions change the library behavior. These permission repercussions on the library functionality will be shown.

Even though some factors in the Moodle configuration should be taken into account to use this library, this is not a Moodle use tutorial, so only necessary Moodle sections to understand this project will be told.

Furthermore, the data structures returned by this library functions will be analyzed so the users will know how to obtain and manipulate the information they want.

The objectives of this project are summarized on these points:

- Study the development options for Moodle external programs that interact with it through different interfaces.
- Analyze Moodle web services plugin development point and investigate its scope and its possible advantages against other interacting ways.
- Facilitate web services use with a library for program development.

- Define the library API, implement its functions and do its documentation.
- Teach the library use with short application examples.

1.3. Memory sections

As stated above, one goal of this project is to use the Moodle system with external applications.

To do this, the first step is to study how to use Moodle. Logically, the Moodle system has to be installed in first place. But this is not a project objective, so this part will be summarized and referenced to external official documentation, where this installation process is explained in detail.

Then, Moodle must be configured for receiving and answering requests from other applications. On this step, different protocols that can be used to do this will be shown, but after that, the REST protocol will be assumed to be used. On that time, Moodle permissions will be analyzed, which are determined by system roles assigned to users. How these permissions affects to Moodle answers will be shown too.

Once Moodle has been well configured, the study of communication process will be started. First the options to get information from Moodle will be shown, and some application examples that use these options. One of these options is the web service plug-in in which the project will be focused.

There are two parts in the web services communication process. The first one is the authentication, where a user has a password associated to a web service with some specific functions. The second one is the request itself, and the resulting response will depend on the user capabilities.

After that, the request and responses involved in this communication will be studied, what data needs the web service and what data returns. In this step, how to get JSON formatted responses will be shown in order to facilitate the data extraction in python code.

Once this is done, web services API will be analyzed, and how to create some functions will be shown in order to make a library in the Python programming language. Library functions will have the same functionality than the functions provided by the Moodle web services.

Then, the library API will be shown, the functions implemented, their functionality, and a use example to show the necessary parameters and the data returned.

This is a project structure summary:

- In first place, there will be introduced some specific vocabulary and technical terminology used in the project.
- Then, the Moodle system will be introduced exposing its purpose, the parts that compose it and its possible ways to work.
- Afterward, the project requirements will be set, with its design decisions and the development methodology followed.
- After that, the programs types that externally use Moodle will be analyzed, and how they interact with it to fulfill with their function. This will introduce the Moodle web services which are the project core.
- Later, web services will be shown and used in examples. The project evolution will carry to developing a web services library which simplifies its use interface. This library development process will be explained too.
- Finally the creation process of external applications interacting with Moodle using the library will be explained.

2. State of the art

This section explains several ideas about technical concepts used along the memory. These concepts will help to understand the project domain and the analyzed systems use.

2.1. Project Field

Before getting in details, some concepts about web systems will be explained, programming terms and other informatics notions to understand this project domain. In this section these general basis will be explained without much details, focusing on those that will serve to explain the development of this project later.

2.1.1. Web servers

In first place, a web server is a system that runs programs to generate a response to the client requests through the Internet. The main purpose of a web server is delivering web pages providing any kind of information.

Usually, the users interact with the web servers with a web browser, which one communicates with the web server with the HTTP protocol. The web server responds to the web browser with a HTML file. This document is interpreted by the browser, which shows the information provided for reading by a human being.

However, servers can have other functionalities besides the web pages services. There are database systems, storage systems, distributed revision control systems and many others. Also, there are many applications besides web browsers that do requests on the Internet, and they use other protocols instead of HTTP to interact with the server. The reason of that web servers are the most known is that every server usually has a web interface where users can register and get details of how to use the system they are using.

Some of the most important web servers are Apache, Internet Information Services (IIS), Cherokee and Tomcat.

2.1.2. Database systems

The database systems are important in this context too, because their mission is to storage big quantities of data. These systems are composed by two parts: the database itself, whose purpose is an efficient storage of the information, and the database management system in charge of administration and providing the database information to users and other applications. The database management system defines the database structure, which can storage data in many different ways.

Some database system examples are MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Microsoft Access and Oracle. Each one of those systems has different database definitions and one created with a database system cannot be used by other database system. Nevertheless these systems can interoperate between them and with other applications using some standards like SQL, ODBC or JDBC.

2.1.3. Communication protocols

There are many protocols that can be used to communicate two processes over the Internet. We have already mentioned the HTTP protocol, used by the web browsers and other protocols can be used to exchange information. This protocol is based on the REST (Representational State Transfer) [1] software architecture technique and systems that follow this architecture are known as RESTful systems. The REST architecture describes the structure style to make distributed systems, as those that compose the Web. HTTP [8] is the transport protocol used in the REST specification, but the architecture defines many other system components. REST determines some system design parameters that differ from other system architectures. The architecture main characteristics are:

- Clients and servers are separated in development terms so clients do not need to know what servers do with their requests and vice versa.

- The communication is stateless, so client and server do not have a protocol status because each request and response contains all the information necessary to understand it. The session status is controlled on the client-side.

- The client experience can be improved making a cacheable system. This means that all responses provided by the server include information about whether they can be saved on the client-side and for how long. Doing this, if users request the same information, a new petition to the server will not be necessary because the data will be achieved from the client cache.

- The REST main characteristic, that distinguishes it from other system architectures, is its uniform interface. This simplifies the interaction with the server and separates the provided services from their implementation. According to REST, an uniform interface must follow four principles:

    - Resource identification (using URIs in web-based systems) independent from its representation, which changes with the request information.

    - The clients can manipulate the resources through the representations obtained as answer to their requests.

    - The messages are self-descriptive, so each message has the information to be understandable.

    - Hypermedia as the engine of application state, which means that clients do only requests to the server through the hypermedia placed in the returned representations provided by the own server.

- REST has only four different well defined operations: POST, GET, PUT and DELETE.

Moreover, many applications run some code in a remote machine and this computer returns the execution result. For this, the RPC (Remote Procedure Call) protocols were developed, which provide a wide variety of functions to do that. An example of these protocols types are the XML-RPC or its successor SOAP. On these protocols are defined an extensive IDL (Interface Description Language) with all the operations that servers can do with that protocol in contrast to the RESTful systems, which have only four possible operations.

2.1.4. Data structures

Some information exchange protocols have been said but the information has to be formatted too, depending of what formats can servers and clients to support. Those formats define the structure of the contained data, so the receiver knows the data and its meaning. Some example of this data structures are XML, JSON or HTML. XML and HTML are markup languages that define how documents must be structured to assign at each kind of data their corresponding mark. These marks are named labels or tags. These both languages generate documents that are simultaneously human-readable and machine-readable.

HTML [11] is used in the web navigation through Internet. Web browsers do requests to an URL and they receive a HTML document which is interpreted with those labels for showing the data in a more comfortable way for human beings.

XML [3] is used as a data structures defining language to exchange information in many contexts because of is an extensible language and its simplicity. The extensible language property means that with this language you can combine data of other different languages, allowing a general use in many platforms.

Another difference between them is that a non well-formed HTML document can be interpreted by a web browser, but the XML specification is focused on creating only well-formed documents.

JSON (JavaScript Object Notation) [4] is known as a light data exchange format from the JavaScript programming language. It transports programs data with the same structure used in the JavaScript programs. This provides a simplicity that improves the data transfer process and facilitates data extraction process because it does not use tags like XML or HTML. This and the JavaScript expansion to all modern web browsers have made that JSON is being used in many contexts, especially as substitute of XML.

The Figure 1 shows how the above explained concepts interact and do possible the web communication between clients and servers in different ways. The machine placed on the top left corner represents an ordinary user that does requests with a web browser through the HTTP protocol and gets HTML files which will be treated by the browser. The bottom left machine

symbolizes the applications that interact with Internet during their running and the diverse options that they have to do it.

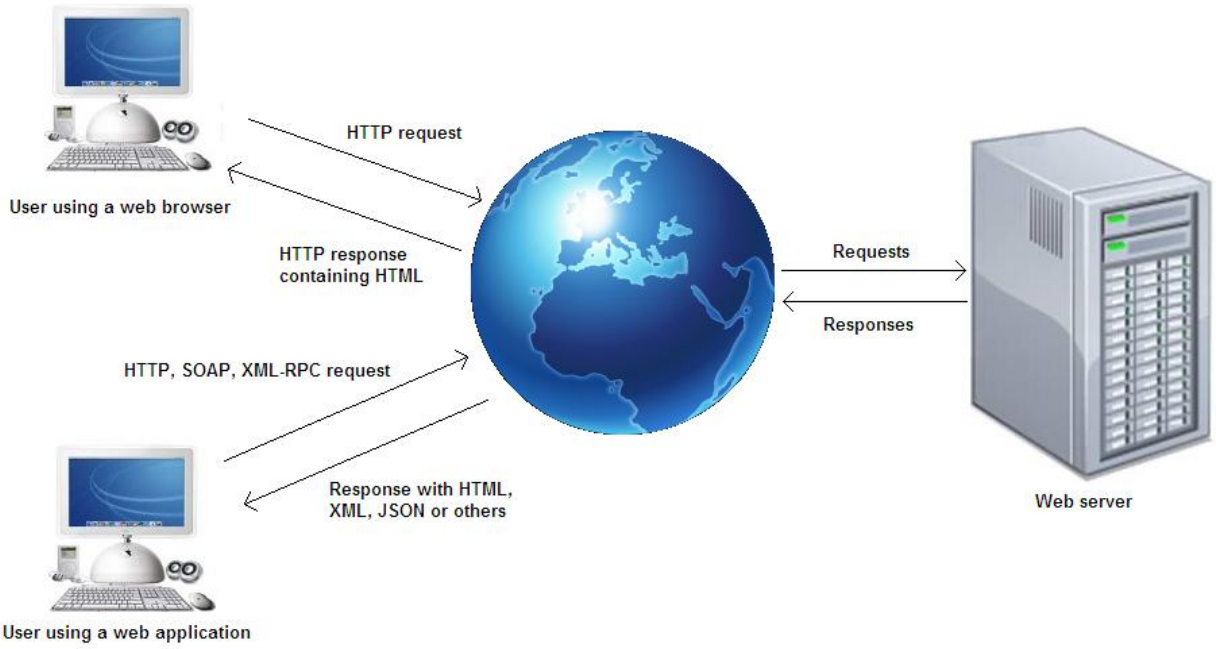

Figure 1: Web communication schema

2.1.5. Parsing

Parse a sentence consists in separate its components and analyze them syntactically. In computing, parsing is applied to text strings obtained from different sources, for example from a file. This analysis separates the information contained and metadata, which represents the data structure. Once its data and its structure are separated, the sentences can be analyzed to detect syntactical problems. Interpreters and compilers do this parsing process to check that the program codes have a correct syntax according to its programming language.

2.1.6. Scrapping

The "scrapping" is the technique of extracting original data from processed data, using reverse engineering for retrieving original data used to generate the document which is being analyzed for this data extraction.

The web scraping does this data extraction from websites and is used by many programs. These programs interact with a website like a normal user which uses a web browser, and analyze the HTMLs returned by the web sites to extract the necessary data which they need to work. These data are stored, to improve efficiency, in a data type known by the programming language used.

These programs are known as *bots* and they can do requests to the servers much faster than normal users. Some web servers try to restrict their access because they can be harmful (for example with a Deny Of Service (DOS) attack, which consists on saturating the server with unnecessary requests, so the server cannot attend at legitimate users).

2.1.7. Programming Languages

All these applications, that exchange data, work to meet its usefulness depending on their programming defined in their source code. This source code can be wrote in many programming languages, each one of them with different characteristics. Those languages are used to give instructions to a machine, usually a computer, and determine their behavior to resolve problems, obtain information, do calculus or anything else that can be done with a computer. Those languages can be interpreted or compiled, can have different programming paradigms (object-oriented, imperative, functional, declarative, etc), have their own libraries and many other properties that differentiate them. Those properties do that programmers have to think on one way or another to do programs depending of the programming language used. There are many programming languages and many variations of each one, but actually some of the most important and that will be used in this project are PHP, JavaScript and Python.

PHP [9] is a free scripting language designed for web developing on the server-side but it can be used to develop any other type of applications. It is a programming language interpreted, imperative, functional, and reflexive and it became object-oriented since its third version. It usually works on the server-side for web navigation because of its capacity of generating HTML files with embedded code without need of processing the code with external functions. This means that a HTML file can have PHP code inside it and the PHP processor executes the code for including its results in the HTML returned to the user. However, since its beginning,

PHP has been extended to include other programming paradigms and it has become a general-purpose programming language. [1]

JavaScript [10], like PHP, is a programming language used for creating dynamic web pages but, in this case, on the client-side. Despite its name, JavaScript has no relation with the famous programming language Java. It was developed by other people and for different purposes. The interpreted code allows web servers to return web pages with JavaScript code embedded, which can be interpreted by any modern web browser. JavaScript code allows creating dynamic effects such as showing or hiding buttons or text, creating animations, showing warning messages and everything without the server participation.[2]

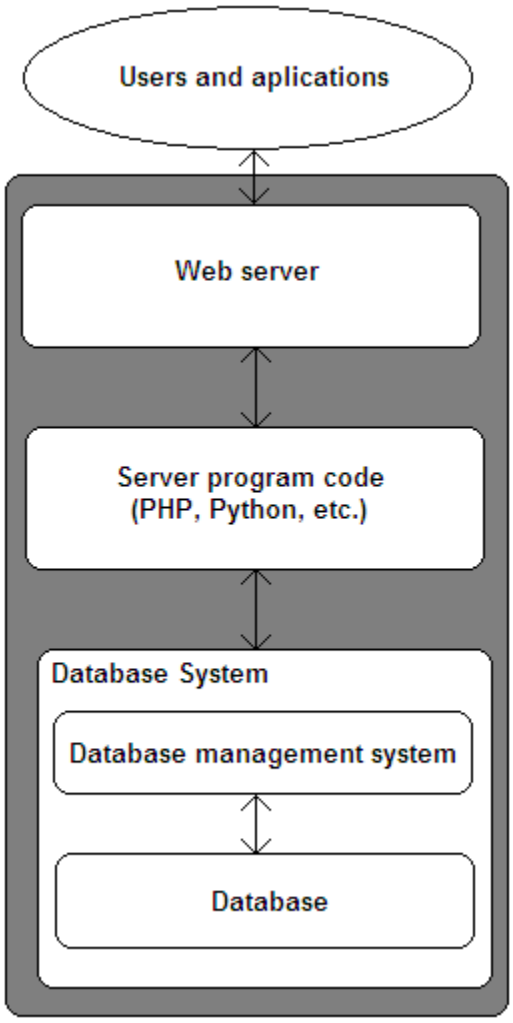

Figure 2: Server structure

Python [2] is an object-oriented, interpreted, and interactive programming language. It is a high-level language capable to design programs with much fewer code lines than many other programming languages. Another of its properties is that generates easily readable source codes because of the language specification. The document indentation is used to define code blocks and affects at the code behavior. Normally, the programming languages do not use spaces and tabulations to define block sentences, they use other chars or reserved words to define the block start and end. Also, Python has a complete and easily usable standard library. All this characteristics and its multi-paradigm programming have done that it becomes into a general-purpose programming language, used in both scripting and non-scripting contexts. By these reasons, Python language was selected as the programming language to develop this project.

[1] http://www.php.net/manual/en/intro-whatis.php
[2] http://en.wikipedia.org/wiki/JavaScript

2.1.8. Integrated Development Environment

An IDE (Integrated Development Environment) is a program that provides several tools to help programmers in applications developing.

Some of those tools are a source code editor, automatic build tools and a debugger. The editor usually suggests the possibilities to auto complete the code that is being written or shows some possible fixes for code problems found .The debugger finds execution time problems showing system status after every code line executed.

Some IDEs are prepared to applications development with a specific programming language, but they can become into multi-language platforms adding external packages. Some popular multi-language IDEs are Eclipse, Oracle JDeveloper, NetBeans and Microsoft Visual Studio.

2.1.9. Learning Management Systems

A LMS (Learning Management System) [12] is a software application used for teaching and educational purposes. Those systems provide a platform where students and teachers can exchange information about their courses, resolve problems in forums, share files to complete the teachings or for evaluating works made by the students, set online grades and much more. These systems usually have a web interface to facilitate access and administration tasks. They allow registered users to enter and these users are registered in different courses, where they can access. The courses can be created and modified by administrators, managers and teachers, where they put contents for the students. Another characteristic of these systems is the capacity of tracking the student progress and report their activities or analyze their skills.

The LMS term is frequently misused for referring a similar type of system such as CMS (Course Management Systems) or LCMS (Learning Content Management System). These ones are more related with development and creation of contents, unlike the LMS that is focused on administrating contents in structured courses and gives other teaching support tools taking external data contents provided by the users.

LMS systems are both pay and free. The most used LMS in 2011 is the non-free Blackboard Learning System, followed by the Moodle system, a free software platform. [3] Both characteristics, being free software and being popular, leaded to Moodle being chosen as the LMS for being explored in this project.

2.2. Moodle

As noted above, Moodle [6] is an open source code system with teaching and learning purposes. It allows making dynamic web pages online, to create and manage courses with the web interface. This made it popular between teachers around the world as a helpful tool that provides resources at students and supports teaching. It needs a data base system, a web server and the PHP interpreter to work properly.

2.2.1. Content Management

Moodle organizes his resources in a hierarchy structure. Its main course structures are the categories, which can contain courses and other sub-categories. These categories could represent a whole university career, one year courses or a set of courses to reach some goal.

A course always has to be in a category or a sub-category, and contains all the course resources divided in blocks. These blocks are used to set a course format. This format can be weekly, split in topics, to show social forums and for using SCORM (Sharable Content Object Reference Model) [13], a set of standards and specifications to share contents between web-based educational systems such as the LMSs.

The resources are stored as course modules and they can be items such as books, files, folders, URLs, etc. or activities such as assignments, chats, forums, tests and many other things.

2.2.2. Authentication and roles

Usually, users need to log in Moodle to access its contents, and each user access can be restricted only to some contents. A non logged user can use the Moodle guest account to see

[3] *A Profile of the LMS Market (page 18)*, CampusComputing, 2011

public contents, which are in courses that allow the guest account access. The user restrictions are established by their user roles. A user is able to have different roles at the same time and their roles can be different depending on the Moodle part where he is. For example, a user can have the "teacher" role in a course and the "non editing teacher" role in other course. These roles give users different capabilities on the system. Depending of these capabilities, users will be able to do some changes and consults to the system or not. By default, a user without role does not have any capability on the system, so he can do anything. There are some Moodle predefined roles:

- Manager: Moodle site managers can see, create, modify and delete every Moodle course, category, user and/or other elements.
- Course creator: manage courses structure but its purpose is not to change the courses contents.
- Teacher: can manage the courses where he is enrolled, adding, deleting and hiding resources and can enroll students in their courses.
- Non-editing teacher: can see all courses resources but cannot add or remove them.
- Student: normal students have permissions to see non-hidden courses and resources but cannot do administrative actions. For example, they can see and download contents from the courses they are enrolled, can write in some forums and can upload files if a teacher enables a resource for it.
- Guest: users logged with a generic guest account, when it is enabled. This role has the minimum permissions to prevent external intrusions.
- Authenticated user: is a user already logged in.
- Authenticated user on front page: user already logged in and viewing the main page.

More roles with the specified capabilities can be created and assigned by the site administrators. The roles are assigned in different contexts. The context determines the user roles in a Moodle site page or another.

These role contexts are “System”, “User”, “Category”, “Course”, “Block” and “Activity module”. Each one's name describes its Moodle area extension.

Once a user is logged in, he has access to those courses where he is enrolled. Normally, users can self-enroll themselves or are enrolled by an administrator but there are many other enrollment methods. Users have to be enrolled in those courses that they participate, so they receive information about those courses and not from every course created in the Moodle site.

2.2.3. Moodle modules

The Moodle core is programmed in PHP and is split in different modules and blocks, which represent Moodle parts. Each module is responsible of its own functionality so all Moodle modules can interoperate between them. For example, every action done in a forum, such as creating a new forum or write a message on it, is programmed in the "forum" module. External modules can be developed and implemented to expand Moodle functionalities. Some Moodle modules are:

- Blog: allow users to create personal blogs.
- Calendar: shows a calendar panel in the browser and different kind of events can be created on its dates.
- Course: manages the control of categories and courses structure.
- Group: allows creating users groups and creating groupings of these groups.
- Message: provide a messenger service, with a searcher of users and a contact list to exchange messages between users.
- Notes: creates personal notes about users.
- User: manage Moodle users and their information.
- Web service: provides some Moodle functionalities for external applications.

This project will be focused on this last mentioned module, which is able to interact without using the web interface.

2.2.4. The Moodle web interface

The Figure 3 shows an example of a Moodle main web page shown in a web browser.

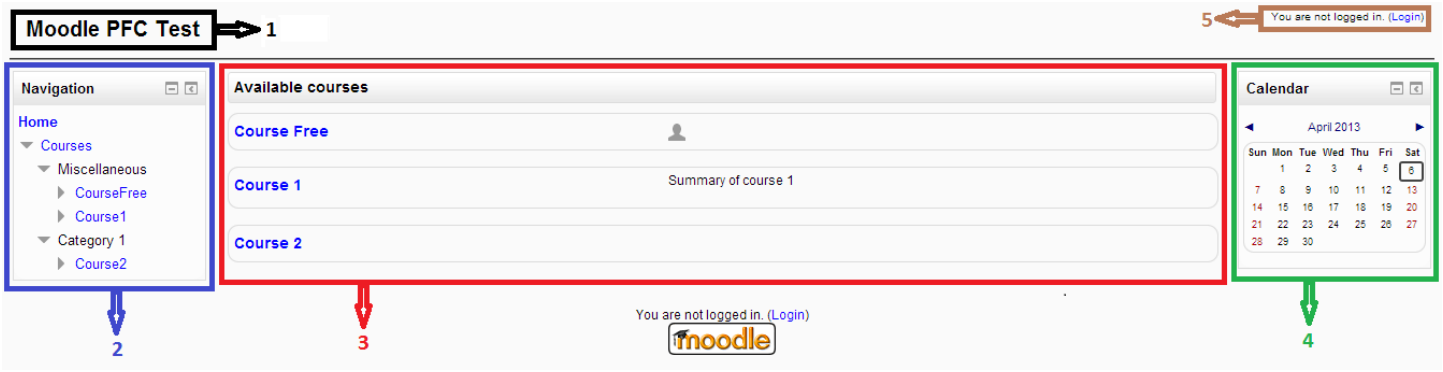

Figure 3: Moodle front web page

Marked fields are:

1) The Moodle web site name

2) Navigation panel. With this, users can go through categories and courses. Logged users can edit resources and panels shown in their main front page.

3) Available courses on this site. This section can be configured to show courses organized by categories.

4) Calendar panel, one of the Moodle modules.

5) Login user information. Once you have logged in you can see your name on this section and access to your user information.

This main page can be modified by an administrator user to show other contents or panels. In the same way, administrators can change many other Moodle options with the administration panel.

2.2.5. Administration panel

When you login with a Moodle administrator account you can configure and modify every part of the web site. You can do this with the administration panel which is divided in 3 sections:

- With the first section, the current web page shown can be configured. In this case, the front page can be edited but it could be a course, a user or other Moodle element.
- The second is the user profile settings. This section is available for every registered user.
- With the last one, every part of the Moodle website can be configured.

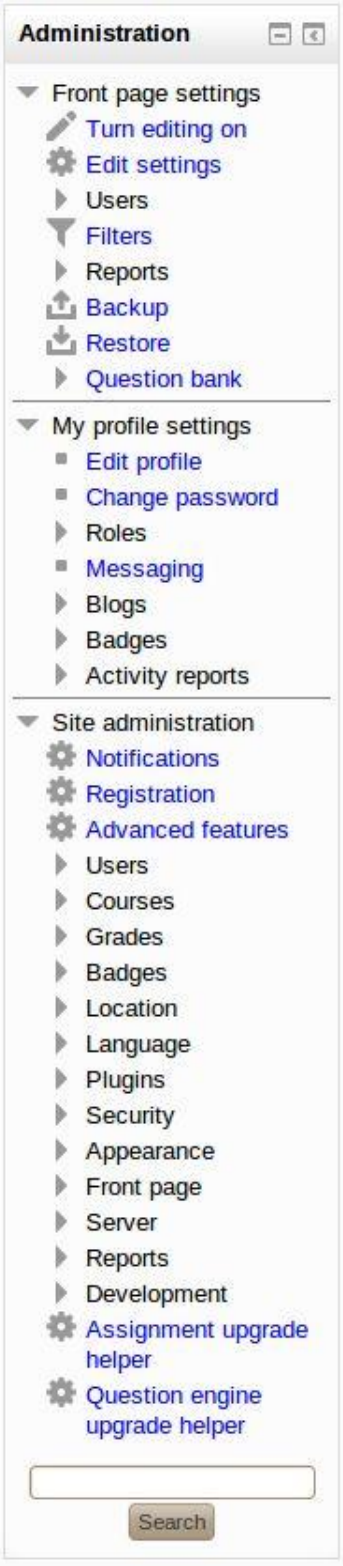

Figure 4: Administration panel

2.2.6. *Moodle Web Services*

The Moodle web services are an additional Moodle module included in Moodle version 2.0. They can be activated and configured in the "Plugins" section in the site administration panel.

Web services have been thought for Moodle interaction with other applications instead of web browsers and they allow using several Moodle functionalities through different protocols. Supported protocols are AMF, REST, SOAP and XML-RPC.

Web services module is in charge of giving the functions defined in the rest of Moodle modules as functions for being used externally with all enabled communication protocols. When a new function is added to a module, it has to be added in the Moodle core file which contains all services provided by the site (in moodle/lib/db/services.php). The function names go with a short

description used for documentation and the paths of their modules so they can be sought in the corresponding file.

Moreover, it handles the creation of functions documentation, which is generated online for every user that want to see it if he has permissions to do it.

The external functions have to be in the corresponding module in the externallib.php file and must be written with a specific structure. This structure defines three necessary methods for each defined function. One defines the data structure for its parameters, setting the necessary data types and their descriptions. Another does the same but for the function data returned. The last method implements its functionality, executing PHP sentences in the Moodle site providing the requested response or the desired result in the Moodle site if no issues happen during the execution.

For example, if a functionality of the Moodle calendar is going to be used by external applications, it has to be defined in the external functions file, inside the calendar Moodle module. Then, the web services module collects all the functions defined in the Moodle core file as services for being provided by a the server. When a user accesses to the web services documentation, the documentation of the enabled functions in a specific web service are generated. For this, the service short description, the data parameters and the returned values defined in every module are used.

2.2.7. Moodle external applications

As explained above, the web services plugin is for interacting with other systems or applications. However, the web services are not the only way to develop applications for Moodle. On this sub-section different kind of applications for Moodle will be explained and how they work to get the information they need to run.

Normally, users get the information they want from Moodle by its web interface with a web browser. Web applications can be classified in two big groups depending on how they obtain these data. They can use the scrapping technique, which was explained above, doing requests as a normal user and extract the information analyzing the HTML file returned by

web server. They can also exchange information with other available systems in the server through other protocols.

a) Scrapping HTML applications

Moodle web services are an additional functionality added to Moodle on its version 2.0 and maintained on later versions. However, Moodle applications development started before this plugin exist, so they cannot use this system to interact with the Moodle system. Instead of this, they usually parse the HTML achieved from a normal HTTP request and use the scraping technique to get the information they want.

Those applications usually search through the HTML file some specific strings or HTML *tags* to get specific information, like the tag used to get the user's name logged in or the courses shown in the main page. They take the links necessary to navigate in the Moodle site from this HTML, so they get the information like a normal web explorer would do.

With this, the applications can work without any additional system and are compatible in different web pages without configuring anything.

However, those applications have some important problems. One of them is that they could not work in some Moodle places because of changing their appearance or version will affect to the HTML received. This could cause that those strings used to search the information necessary to the application will not be found, and make a malfunction of the application.
Also, the HTML processing can be very complex in some cases, with all the problems that this processing could result in.

Another disadvantage is that each request gets the entire HTML file so much information is dropped because an external application do not need it (for example, the information used to show the elements in the web browser or that related with the page visual design). This carries several problems like using more bandwidth than necessary and therefore more delay in the obtaining data process.

Besides, the Moodle site cannot differentiate between users navigating with normal web browsers and applications that work on this way. These applications can be detected with other systems which analyze net traffic and recognize these applications' behavior.

An example of this kind of programs is the MDroid application[4]. It was developed for android mobile phones and can get course files and forum discussions. It shows a simple interface with the login page, once has been logged shows the list of available courses and can show each one of their containing files and forums. It also has the option of working offline with those contents that have been already downloaded.

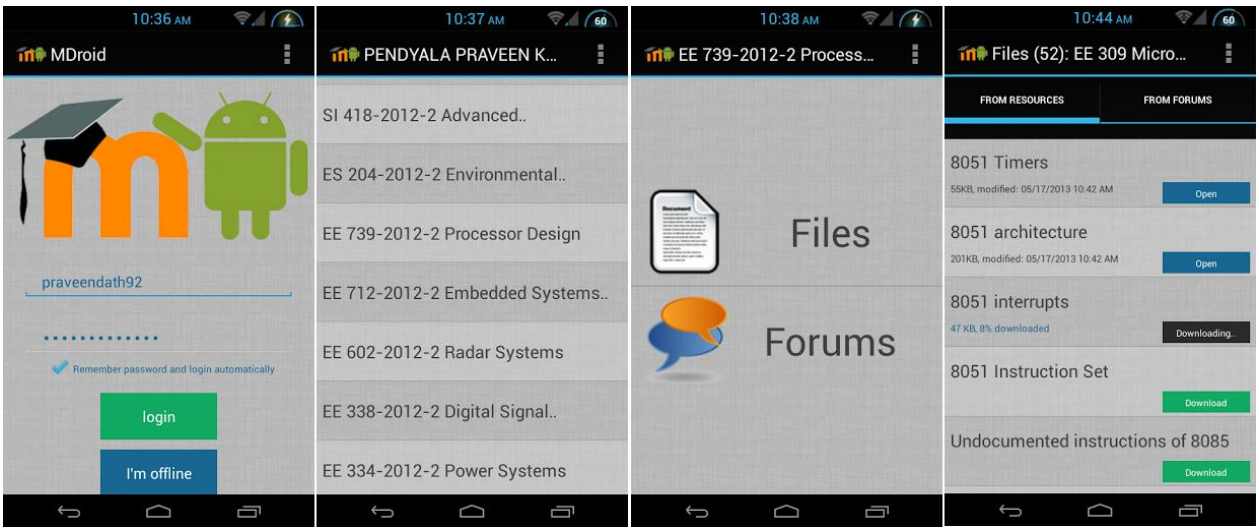

Figure 5: MDroid application screenshots

(Obtained from:

https://play.google.com/store/apps/details?id=in.co.praveenkumar&hl=es)

b) Using Moodle Mobile app

Another option to do this is to use the Moodle Mobile app web service. This solution is similar to the web services solution. As its name indicates, this option uses a special Moodle web service developed for mobile applications. This service has some predefined web services functions and is available in the Moodle site by default since the Moodle

[4] Source code available at https://github.com/praveendath92/MDroid

web services were implemented. However, as every web service, the web services option has to be enabled in the Moodle site to use it.

The official Moodle developers have created an application based on this web service. The Mobile app provide some Moodle features as downloading resources, sending private messages, showing course contents and also can exchange contacts with the mobile phone contacts list. [5]

This web service functions are unchangeable and limited so not every web service function can be used with this service. Because of this, in this project will be used generic Moodle web services, with which ones every available function can be added for exploiting all this Moodle plugin functionality.

c) Using other web services

Using Moodle web services instead of web scraping has its advantages and its disadvantages. If web scraping is used a deep analysis of the HTML files is necessary, but if web services are used, how to use those services should be learned. To do that, the web services API will be under study, just like the way for making requests with the enabled protocol to communicate with the server. In this API, all the web services functions are defined with their names, the parameters they need to execute, these parameters types and the data structures returned by the functions. The wrong use of these functions will do that the Moodle service returns an error.

Once this API is studied, a specific web service has to be created and configured with desired functions for being used. The available functions that this service could provide depend on the Moodle version installed and other external plugins which can add non-official functions.

These functions can be used in an external program and will work in the same way in future versions. On the other hand, if web scraping is used the potential applications functions will not depend on the web services plugin. They will only depend on what can be done with the scrapping process. However, the use mode of the web service functions

[5] Official application web page: http://docs.moodle.org/25/en/Mobile_app

changes with the protocols supported. Each protocol has different forms of representing and transporting information, so both parts of the communication have to be programmed according with the established protocol. A protocol and in general, any system that can be used by applications must have a detailed documentation of how that system should be used. This documentation is collected in the API, the Application Programming Interface.

The Moodle supports AMF, XML-RPC, SOAP and REST APIs on its version 2.5.

Others developers have made libraries such as the one created in this project with other programming languages. However, these ones use older Moodle versions or they use currently deprecated functions or do not provide all the available web services functionalities.

3. Technological description

On this section the searched goals in this project will be explained. These goals will be the project scope limitation and will define some development restrictions.

Along the project there will be different options that can be chosen to achieve the project goals, so the taken options will be told and the reasons because they have been chosen will be explained. Also the development methodology that guides the project into defined phases will be exposed.

The requirements, the design decisions and the methodology will be the bases of the next section where the project objectives will be reached.

3.1. Requirements

This project main objective is to study the possible Moodle use ways. Usually, the way of interacting with Moodle is using a web browser. It provides a graphical interface thought an intuitive use based in a generic representation of the information. Web servers and browsers are based on the REST distributed system architecture but other network applications do not have to follow these rules.

The purpose behind this study is the applications development for external Moodle manipulation. Those programs can offer other kind of uses, automate routine tasks or add new functionalities to Moodle.

For example, a script can be created to show the events defined at Moodle in the next ten days. Another use case is to get forum notices and post them in a social network or, on the contrary, to include external notices into Moodle forums. Also, a backup program can be created to save contents in other web storage systems.

However, there are many program paradigms with which applications can be created and each one has different valid forms to connect and communicate with other programs. The communication between the system parts is given by the system architecture. They can be split in two big commonly-known groups: client/server and peer-to-peer.

The peer-to-peer model defines every system part as a node, and the nodes are programmed as equals. The same program is distributed over a network and its programming allows communicating different machines.

Internet is based on the client/server model, composed by two entities. One is the client, and is in charge of starting the communication process. It uses a service provided by the other architecture entity, the servers. These contain information that users want or execute some implemented operations that users need. This structure is adequately for this project programs, participating on the communication as the client that request information to a server, the Moodle system. Also the applications do not need to have any relation with the Moodle code, so a separation in development terms is sought, like in this architecture.

Besides, the specified services can be proportioned in many different ways. How those services are provided is defined by the distributed application, but its interface depends mainly of the communication protocol used. For example, if a protocol with status is used, the communication status should be taken into account at the moment of creating the program sentences.

The functionalities of the programs developed in this project are not defined, they can do everything that the programming language used let them to do. Therefore, the interaction with the Moodle system should be as simple as possible, for adding flexibility to create programs. Also, this pliability can be improved using a protocol which does not need previous requests to do the next request.

The Moodle external application types have already been mentioned in section 2.2.7 and one possible choice is using the web services Moodle plug-in. The purpose of this module is to get requests from systems which are independent from Moodle, using different type of communications protocols for it. One of these protocols is the one defined by the REST system architecture. The HTTP protocol matches with the REST specifications and is the used in the communication on these distributed systems.

The REST protocol meets all the above requirements to make an easily usable interface of communications with Moodle. It is stateless, follows the client/server model and reduces its use complexity defining only four type of possible methods. These operations are the most

general operations that can be performed. GET is for getting server resources. PUT is for updating or creating new resources to the server through an URL or save any kind of information. POST is for creating some resource but it can also be used for giving information to the server as, for example, parameters for executing an operation or changing some value on the server. DELETE is used for removing resources from the server.

Also, REST systems have been used since the Internet beginning, so it is a mature architecture, very extended and implemented in many systems around the world.

By all the above reasons, the web services analysis will be focused on using the REST model, in both the Moodle and the developed applications.

The web services API sets how they must be used. The REST interface works with the POST method, one of the four possible methods of the protocol used in RESTful systems, so the communication design is limited at using this method. The POST request specified in the HTTP protocol will be used to say what function want to be executed and its parameters, if are necessary. Every request will be independent from other in network terms because even if the application needs several functions to fulfill its programming, the requests will be treated independently.

Once how to use Moodle with REST requests is known, the development of programs and scripts using this Moodle interface will be analyzed. This analysis will be more orientated to generate the necessary sentences to use the Moodle functions than to doing a complete functional application with a specific purpose. Because of this, short scripts programs will be used in the project phases instead of big designed and structured applications

These were the initial requirements, but once this part was completed, to provide a framework was decided for developing applications using these tools. Before starting any application development, to get tools and learn to use them is important for facilitate creating applications and improving the productivity. The Moodle web services are a recent characteristic of that system and are being developed yet. Excepting some examples found in the Moodle web site and other open codes provided by programmers, there are a few mechanisms for applications implementation.

This framework consists in programming a library that simplifies the communication process with Moodle, so applications can use this library instead of get worried about how to generate the requests necessary to interact correctly with Moodle.

In this library development process, all web services functions were analyzed and tested for its implementation. The library provided functions and their functionality will be explained later.

During this procedure several similarities between functions and data structures were found, facilitating its utilization. Some issues were found in other functions, in their use form or in their implementation.

All these steps and issues will be detailed in the Implementation section.

With this library already finished and its methods defined, an example of script development will be shown using the mentioned library.

3.2. Design

Here there will be exposed the design decisions taken in this project and why those decisions have been taken.

There is a wide range of options for using one programming language or other. The applications development process is not important on this project so the programs should be implemented easily for not increasing its complexity. Therefore, a high-level programming language should be chosen because they provide abstractions to simplify the programming. These abstractions are given through using libraries, a collection of methods and functions to perform complex operations without being implemented every time that they are needed.

The libraries of a programming language are frequently a limiting factor of its use area. The library functions are as important as its documentation. A good documentation makes easier the development process, reduces its duration and increases the language use possibilities. A programming language having tools for connecting systems with REST will be necessary, so the chosen should have official libraries with that functionality. Also, the open source programming languages has the advantage that everyone can add new libraries. If a required functionality is not implemented, it may be developed by third persons and incorporated to the language.

Having a source code easy to read is another desired property, but this usually depends on the programmer and not on the used language. However, the Python programming language needs source codes correctly tabulated and structured to work. Initially this could look as a language limitation, but seeing code well placed is pleasant, especially when it is going to be read by other programmers.

Python is used in many areas and platforms, and it is currently quite spread within the programming languages. Its documentation could be sometimes difficult to understand, but many non-official codes of applications and examples are available on the net complementing the documentation. In addition, it has been used in previous works, so a starting tutorial will not be necessary.

All above reasons indicate that Python is the appropriated programming language to be used in the project. It will be used for the library developing and to show programming examples.

The Python programs that will be made will have the main objective sending and receiving HTTP messages. This can be done with the urllib2 default library, which has most of the HTTP functionalities. However, in further project phases this communication will be difficult to perform with this library, because its mechanisms for doing complex requests are not intuitive. Instead of wasting many efforts to do those requests, other options were searched.

“Requests” is an external Python library which has been recently published. Its only purpose is to use HTTP protocol in Python programs with a simple and intuitive syntax. It

takes care of every HTTP connection details, automating the request parameters encoding and maintaining the existing sessions, with the help of the included library “urllib3”. Furthermore, an extensive documentation supports the library with many use cases and their corresponding examples. So the sentences of urllib2 were replaced by the corresponding instructions of the "Requests" library and this last one was used in the remaining project parts.

The programs creation process will be assisted by using an Integrated Development Tool. **Eclipse** is the IDE chosen because it is a free software development tool, commonly and easily usable with an intuitive user interface. Eclipse is a development tool for Java applications by default, but due to its fame, many external plugins exist so it can be used for applications development on other programming languages. There has been decided that Python will be the programming language used in this project, so one of these external Eclipse plugins will be necessary. The Eclipse plugin used is called **PyDev** and provides all IDE basic functionalities for python such as code completion with auto import, syntax highlighting, code analysis, go to definition function, refactoring, mark occurrences, debugger and many others.

Once the Moodle web services use was established there was discovered its aptitude of giving responses in the XML or JSON formats. JSON syntax is very similar to the basic Python data structures, and the standard library provides a module to pass data in JSON format to these structures. Furthermore, JSON model uses less data to transfer the same information than its equivalents, saving bandwidth in the transmission. For these reasons, the means to get responses with JSON format from Moodle will be studied.

3.3. Development methodology

This project has followed a spiral development methodology, which consists in a set of tasks that are repeated in an order in different cycles to incrementally match its goals. Each cycle consists in four steps:

1- Determine cycle objectives: in this phase, the tasks that would let to reach the final goal in future cycles have to be determined. To seek all possible ways to complete these tasks is important in order to take the best choice. Besides, to have well defined the cycle scope is necessary as well as its limitations depending on the dedicated time to that cycle.

Figure 6: A software development spiral (Image obtained from: wikipedia)

2- Risk analysis: predictable problems are analyzed to evaluate their cost and decide if resolve those problems and how resolve them, or search other alternatives to avoid those problems.

3- Developing and testing: this phase consist in making those tasks programmed in the first phase with the solutions provided in the second one. Before finishing this phase it has to be ensured that all products works correctly and as they were defined in the initial phases.

4- Planning of next phases: once this cycle objectives are reached, new targets have to be defined for future cycles taking into account the time expended and the knowledge acquired in this cycle.

4. Implementation

This section explains the whole developing process followed in the project, indicates the steps followed to: analyze the Moodle API and how to use its functions, creating a Python library to facilitate using these functions and the way to use the library. Each one of these steps represents a developing cycle of the followed spiral methodology.

Before starting any installation, the first thing chosen was the Moodle version to be used. Many Moodle versions match the project requirements with one or other configuration. Older versions than 2.0 can get some of these functionalities using the "OK Tech Web Services" which was thought for administrative purposes. It has to be installed after Moodle, for being added as an external program. These web services are still in development but their backward compatibility makes that many options only are functional in old versions. In the Moodle version 2.0 the web services were integrated as a Moodle plug-in. These initial web services had a few functions so they were not very attractive for applications development. However, in version 2.2 the web services were extended with new functionalities and new names with the old ones, doing that the already existing function names were deprecated. Since this version until the currently 2.5, the web services have been improved with more functionalities following the same schema. In first project phases, the Moodle version 2.5 was used which was in development process at that time. Once the stable version was available, the Moodle test site was updated and this version was kept until the work ends.

4.1. Work with Moodle using REST API

On this first phase, the target is to get a Moodle web interface for working with external programs different to web browsers. For that, the current web applications will be studied for looking the best choice to get it. Once all the options had been explored, the Moodle web services were chosen as the adequate interface. In this cycle, the web services will be put on running documenting all the steps followed. At the end of the cycle, some scripts will test those services.

4.1.1. Install Moodle and access to its web interface

The first step to develop Moodle applications is to have a functional Moodle to use and test these applications. So in first place the Moodle installation is necessary. Moodle is a system that needs other sub-systems to work and they should be prepared and configured before Moodle installation process is started.

The Moodle core is programmed in PHP so it needs its interpreter for being an interpreted language. This interpreter should be installed where the Moodle system will be working so the computer can understand and execute the PHP code. This project was performed using the Moodle 2.5 version and the minimum PHP version for this Moodle version is the 5.3.3.

It also needs a database system to store all information relative to the Moodle site. This information can be users data, courses contents, stored files, students grades and many others kinds of data records. Moodle core system is programmed to work with several data base systems. These are the supported data base systems:

- MySQL, minimum required version: 5.1.33. This one was installed to be used in this project.
- PostgreSQL, minimum required version: 8.3
- MSSQL, minimum required version: 2005
- Oracle, minimum required version: 10.2

A last additional system that Moodle needs is a web server. This is the system part that will take over the communication between Moodle and its users, with the web browser or with other web services. The web server is the interface that connects the Moodle site with the Internet, making it accessible from everywhere. Fully supported web servers are Apache and IIS, and they ought to be configured so they can serve PHP files. "The version is not critical but try to use the newest web server build available to you."[6]

[6] http://docs.moodle.org/25/en/Installing_Moodle

Once those systems are installed, some steps to prepare the Moodle installation should be followed.

The Moodle files can be downloaded and put in the web server location so they can be accessible with a web browser.

After that, a new empty database may be created in the database system and configured in order for Moodle can access and store its information on it. In this process, four necessary parameters have to be remembered so the Moodle site can manage the database. These parameters are the database server hostname where is the database system, the database name, the username to access at the database and the password for the above user.

Then, a directory has to be created where uploaded Moodle files will be stored. The directory permissions must be changed so Moodle can access and write on it. Finally, the Moodle installation process should be started using the command line installer or the web based installer. Both options guide through the installation and request information for basic Moodle configuration.

With this, an empty Moodle site has been created, with no courses, no resources, and only with the administrator user account created in the installation process. With the admin user can be created other users with permissions to manage the Moodle site, or create other admin accounts. With those accounts, categories can be created for containing courses with the structure agreed. Those courses will be individually managed by their associated teachers, who will have their accounts configured for it. Students' accounts can be created manually by managers or can be self-created by the students if an admin has activated the self registration option for the login page.

The URL to access at the Moodle web page has two parts. The first is the web server URL, and the second the relative path where the Moodle is placed in the web server. Following the standard Moodle installation guide, the URL to enter in Moodle main page should be something like:

http://webserverurl.com/moodle/

From this URL all Moodle system resources are accessible. A web browser will shown the main Moodle web page in this URL (see section 2.2.4).

4.1.2. Activate and configure web services.

Before using the Moodle web services there are several things are necessary to be configured. These actions can be performed only by an administrator of the Moodle site and a user with permissions in the data base created for being used by Moodle. Once the web service is activated, it can be used by every Moodle user.

Activating Moodle web services

The first step is activating the Moodle web services functionality in the site. For that, an admin will log in and use the administration panel. In this panel, go to the *Advanced features* section and enable the option *Enable web services*.

After that, one of the web services protocols will be enabled, at least, according with the external application programming. In this case, the REST protocol is being under study, so this one will be enabled in the *Site administration* → *Plugins* → *Web services* → *Manage protocols* section. In this section, enabling the "Web services documentation" is recommendable too, so users will be able to see what functions of the web services they are able to use depending of their capabilities. If a user does not have the required capabilities to use a function, it will not appear in his documentation page (this will be explained in more detail later, in the *Checking users' capabilities* section).

Create a new web service

Once web services are enabled, a new web service will be created to use its functions. For this, go to Site administration, Plugins, Web services, External services section and click in the "Add" button.

A new window lets to introduce the new service name and the "enabled" checkbox has to be activated in order to use it. If more options are shown, two more will appear:

"Can download files" option lets users to download files with web services. If this library functionality is wanted, this option has to be selected.

The "Authorised users only" option can be set to restrict the use of the web services to a group of users. Only the users with the capability selected in the "Required capability" list will be able to use this web service.

Adding functions to the web service

Once the web service is created, it can be seen in the External services section but it has no functions. So they are added with the functions link for the corresponding web service. At this moment, all desired functions for its using will be added. The library developed in this project works with all non-deprecated standard functions for this Moodle version. To see the functions list supported by this library see Appendix 1. When all functions that will be used have been added to the web service, each one of them can be seen in a list with a short description and the required capabilities that a user is required to have to use that function.

Set a service short name

With the Moodle web service created on the website, to access the database system is necessary for doing some modifications. How to enter in the database depends on the database system installed before the Moodle installation, but the necessary changes can be done with the same account that Moodle uses to access the system (and which was provided in Moodle installation process). Once inside the database system, the "external_services" table can be found in the database used by Moodle. In this table, information about the existing web services in the Moodle site can be found and one of them should be the service created before. One of the parameters that can be found in this table is the "shortname", and the new web service should have this field empty. For allowing users to obtain their token (string used for authentication in web services) using their username and password with the web services, a shortname have to be provided to the web service.

Checking users' capabilities

Once the web service is ready, the user capabilities should be modified to let them use the web services. A user capability is a specific permission to do something, and what a user

can do is limited by his user roles. A role is a list of allowed capabilities that can be assigned to a user, so the user can do everything that his roles allow him to do. There are several necessary capabilities for using the web services and they are not provided by default to users except at administrators.

The first required capability depends on the protocol used for the web service. In this case the REST protocol is used, so users are required to be able to use that protocol with the Moodle site. The name of this capability is “webservice/rest:use”.

The other required capability is necessary in case that is desired that users can see their web services tokens, manage them and see the documentation about the available web services functions (if the "Web services documentation" was activated in the *Manage Protocols* section in the administration panel). All these things can be done by the users if they have the "moodle/webservice:createtoken" capability and they use their *Security Keys* section, which is available in their profile administration panel.

Furthermore, using each function requires that users have specific capabilities for security reasons. Therefore the list of available functions for a user depends on his capabilities.

There are two ways to assign those capabilities to users. One is to modify the existing default roles in order to every user created with that role can use the web services. The other one is creating a specific role with those two capabilities and assigning it to users that will be allowed to use the web services. However, this second option is recommended because those capabilities have to be assigned in the system context (see next paragraph). A role with a system context means that a user which has that role will have the capabilities assigned to that role independently of the user’s location at the Moodle site. The other role contexts limit the capabilities assigned to the Moodle parts that they specify.

In order to create a role, an administrator has to go to the *Site administration* → *Users* → *Permissions* → *Define roles* section and press the *Add a new role* button. Then a short name and a full name should be set to the role, set the system context for the role, select the “moodle/webservice:createtoken" and “webservice/rest:use" capabilities so they will be marked as allowed and press the *Create this role* button.

Once the role is created, it can be assigned to the chosen users in the *Site administration* → *Users* → *Permissions* → *Assign system roles* section, choosing the role created in the list and moving the users from the potential users list to the existing users list with the *Add* button.

4.1.3. Get token, login with a web service

A token is a personal key generated for user authentication using web services. So the token identifies both the user and the web service. A token is composed by a random sequence of numbers and letters like this: 77246fc16e1198dbd171e7f467dfa906

This key is generated by the system and can be obtained in different ways. If a user has the "createtoken" capability, that user can see his tokens for the existing web services with a web browser in the Moodle page in his *Security Keys* section in the profile administration panel. He also could generate a new token in this page if he suspects that his token has been obtained by other persons. If the web services documentation has been activated, in this section will be a *Documentation* link that let see the available functions in that service for that user. However, if user do not has that capability an administrator has to generate the token for every user that will use the web service.

The token also can be obtained without using the Moodle web interface if a service short name was provided. The token can be got doing a request to the URL

http://www.yourmoodle.com/login/token.php

This request needs the following parameters to return the user's token:

- Username: the user name which is normally used to log in at Moodle.
- Password: the password for that user
- Service: the service short name for the wanted web service. This name is provided by the Moodle database administrator.

If those parameters are added to the URL, it will look like this: http://webserverurl.com/moodle/login/token.php?username=USERNAME&password=PASSWORD&service=SERVICESHORTNAME

Once the user knows his token for the desired web service, he can start to do requests to the system.

4.1.4. Moodle web services API

The API, in general, provides all the necessary information about software to know how to use it. This info is usually made with the possible functions that can be performed by that software, the requirements to use them, their result and all the necessary information detailed for a complete understanding of that program utilization.

The web services API shows every available function for the user that is currently accessing to it. This API shows a list of functions that can execute its corresponding codes to make changes or get information from Moodle. Clicking on a function will deploy the details of that concrete method. Here, a short function description, the structure and meaning of their input parameters as well as their return values can be seen. The parameters info is available in a generic data structure, in PHP data structures and specifically for all activated protocols because each protocol uses a specific syntax for their requests.

All information about Moodle web services is available in *Settings* → *Site administration* → *Plugins* → *Web services* → *API Documentation*. This section can be seen only by administrators so they can see the functions they want to activate with the web services in the Moodle site. Normal users can only see the functions they are able to use according with their capabilities with de Documentation link in the *Administration* → *My profile settings* → *Security keys* page. The Figure 7 shows the "core_group_delete_groups" function documentation, which determines "groupids" as a required parameter as a list of integers. It also defines that the function do not return anything if everything went well or otherwise the error message.

This API has been instrumental in all the project phases and has determined many results. It is the main programming guide for web services developers and all their functionalities should be collected here. It also has been helpful for analyzing the Moodle organization and to set several decisions about the library developed later. The library functions names follow the same convention and keeps its modularized structure to keep the functionalities classified.

core_group_delete_groups

Deletes all specified groups.

Arguments

groupids (Required)

```
General structure

list of (
int   //Group ID
)
```

```
XML-RPC (PHP structure)

[groupids] =>
    Array
        (
        [0] => int
        )
```

```
REST (POST parameters)

groupids[0]= int
```

Response

Error message

```
REST

<?xml version="1.0" encoding="UTF-8"?>
<EXCEPTION class="invalid_parameter_exception">
    <MESSAGE>Invalid parameter value detected</MESSAGE>
    <DEBUGINFO></DEBUGINFO>
</EXCEPTION>
```

Figure 7: Moodle web services API function example.

Get JSON formatted responses

During the web services documentation process, the chance of getting functions responses formatted with JSON was found. Moreover one Python standard library can transform JSON formatted data in Python's basic data structures and vice versa. As a result of this, JSON message format was decided to be used in Moodle responses.

Saying at Moodle to answer in JSON format is quite simple. Only one more parameter has to be added to the request and the server will return JSON responses. The parameter name is “moodlewsrestformat” and its value should be “json”. With this, the request can be easily processed using the “json” default Python library.

Returned data structures

Data stored in a Python variable contains different information depending on the executed function. A function to get a course contents has been executed in this case, but the contents information can be structured in many different ways. The information returned by the function and its structure is defined in the Moodle web services API.

The response structure for the “core_course_get_contents”, which returns information about the specified course contents, is shown in Figure 8. Speaking in Python's basic variable types terms, course contents are in a list where each element is a dictionary. A dictionary represents a course section and contains its information with his keys and values. Some of this fields are optional (could not appear in the answer) and others are required, such as the section identifier, its name, etc. In the same way, every Moodle entity (users, modules, events, messages, courses, etc.) is defined by the Moodle web services as dictionaries with their different keys and values.

The section dictionary has a modules field which also contains a list of dictionaries that have the modules information. The modules have a contents list as well, all of them with their particular values. So seeing the response structure, the contents which can be files, URLs to

other sites, or folders are in different modules such as forums, assignments, quizzes and other Moodle activities. These modules are grouped in the courses sections that set the course format (weekly, by topics, etc).

At this point the data obtained and its type is known, so now they can be used by the developed application to do what it needs. In this example, it can be used to show the course sections names, or to download every available resource in the course (if the web service allows it).

Other functions were tested to getting used at web services management. After a few test, the similitude between many functions was detected, excepting the specific function parameters. Also, the need of knowing the parameter structure required to make the petition complicates the creation of programs based in web services. That is why the creation of a library to simplify the communication process was established as a new project goal.

Response

```
General structure


list of (
object {
id int    //Section ID
name string    //Section name
visible int  Optional //is the section visible
summary string    //Section description
summaryformat int    //summary format (1 = HTML, 0 = MOODLE, 2 = PLAIN or
4 = MARKDOWN)
modules    //list of module
list of (
object {
id int    //activity id
url string  Optional //activity url
name string    //activity module name
description string  Optional //activity description
visible int  Optional //is the module visible
modicon string    //activity icon url
modname string    //activity module type
modplural string    //activity module plural name
availablefrom int  Optional //module availability start date
availableuntil int  Optional //module availability en date
indent int    //number of identation in the site
contents list of (
object {
type string    //a file or a folder or external link
filename string    //filename
filepath string    //filepath
filesize int    //filesize
fileurl string  Optional //downloadable file url
content string  Optional //Raw content, will be used when type is
content
timecreated int    //Time created
timemodified int    //Time modified
sortorder int    //Content sort order
userid int    //User who added this content to moodle
author string    //Content owner
license string    //Content license
}
)}
)}
)
```

Figure 8: function response structure

4.1.5. Use some functions: test with courses functions

At this point, some tests will be performed to check the information collected until now. This subsection corresponds with the tests of this project phase. Only an example to put in practice the above steps will be shown as reference instead of all the tests really done. The only difference between these tests would be the executed function name and the parameters provided, so they are not necessary for understanding this step.

With the web service token, a user can be authenticated to use its functions, but is not the only requirement. He also needs to provide the function parameters and to have the specific required capabilities to use that function. Some functions do not have parameters or additional permissions, but it depends on each function implementation. The required capabilities for each function can be seen entering in the Documentation link of the corresponding web service in *Security Keys* user's profile configuration area.

As a first test, a function from the courses module was chosen for getting the content of one course. This function name in the Moodle web services documentation is "core_course_get_contents". The function requires the course's unique identifier set by Moodle at the course's creation. This identifier can be found looking at the URL when using the normally web browsing in the course's page. For example, the URL http://webserverurl.com/moodle/course/view.php?id=4 shows the Moodle course with the unique identifier value 4. The ID can be achieved with other web service functions too.

Now that the parameters have been obtained, the REST request will be generated using them. The Moodle resource that serves at the web services requests is the server.php file, which location depends on the protocol used. In this case, the URL where the program should do its request will be: http://webserverurl.com/moodle/webservice/rest/server.php Moodle expects web services requests on this URL, and they have to be done with the POST method because this REST method is thought for sending parameters in the request. The POST parameters can be sent both in the URL and in the body request.

To adding the parameters to the request, the Moodle official documentation [7] about web services is consulted and shows that web services always need two parameters, the web service token, identified with the "wstoken" parameter name, and the web service function

that are going to be called, with the “wsfunction” name. These two parameters are generic for all web services calls, but specific function parameters have to be added too. To know how to send those parameters, the API of the Moodle web service functions is checked again and shows that the course identifier parameter name is “coursed”. With the URL, the parameters names and their values, the resulting request if the parameters are included in the URL will be something like:

http://webserverurl.com/moodle/webservice/rest/server.php?wsfunction=core_course_get_contents&wstoken=77246fc16e1198dbd171e7f467dfa906&courseid=2

All the parameters have to be encoded appropriately for being sent through the net and to be decoded by the receptor. In the transferring procedure, the characters have to be adequate for being interpretable by the intermediary machines. The communication protocols use special chars to delimitate the message parts but these chars could be send as part of the transferred information. For being able to transport these data, they are codified with a char set, a set of rules that changes these characters by a special string which identifies the character during the communication. The other communication extreme has to decode the message with the same char set to recover the original data values. The utf-8 char set is used by the HTTP protocol and will be utilized with the “urllib” Python library for sending POST requests.

This request was made with the following python test program using the default libraries for REST requests “urllib” and “urllib2”.

```
import urllib2
import urllib

if __name__ == '__main__':
    url = 'http:// webserverurl. com /moodle/webservice/rest/server.php'
    function = 'core_course_get_contents'
    token = '77246fc16e1198dbd171e7f467dfa906'
    param = urllib.urlencode({'wstoken':token,'wsfunction':function,'courseid':'2'})
    req = urllib2.Request(url, param)
    response = urllib2.urlopen(req)
    print response.read()
```

Moodle processes the petition and if no errors occurred returns a response with XML format. Possible errors are that web services are not enabled, web service function is not

available on that service, the token is invalid, the token is valid but the user have not the required capabilities or the function parameters are incorrect (in this case, for example give a non-existing “courseid”). If Moodle have any of these problems in the petition processing, it will return a generic error message with a short description of the trouble.

Once the response is obtained it can be processed for data extraction so they can be used by the program. The XML message treatment would consist in searching the tags in the message body. XML has these labels because is defined as a generic markup language which is able to contain and transport any kind of information. The tags can contain the data specified with tag attributes or can contain another labels. Grouping those labels, a data hierarchy can be reconstructed to have a right information representation.

If the response is in JSON format the extraction process would be different because instead of looking for tags, there should be the special characters that this formats uses to structure the data. However, Python and many current programming languages have specific tools to extract JSON data into their basic variable types.

4.2. Develop a python library for Moodle web services

This library tries to provide an interface for using web services and help the development of Moodle applications. The used tools to create this library will be mentioned, as well as the guidelines taken to do it. The library's functionalities catalogue will be exposed with a description of their utility and a use example will be shown after.

4.2.1. Used tools

As the initial study was made using Python programs, the library was developed on this programming language to take advantage the already written code.

Also, developing process was supported with the utilization of the IDE **Eclipse**. It would help to solve programming syntax errors and organize project progression. However, Eclipse has development tools for Java applications by default and not for Python. A plugin has to be

added to Eclipse to program with Python language. The plugin installed is named **Pydev** and provides the basic tools of a Python IDE.

On every created program its documentation is important to help other programmers to understand the code and to use those programs without the need of knowing how they work. This last concept is one of the abstraction method bases, a necessary engineering tool to extend and encourage more complex applications development. Therefore a tool is necessary for helping to create good documentation for the library. The **Epydoc** program automatically generates a structured document with the help of commentaries in the source code and the project structure. Also, **Graphviz** is a complement of this tool. It similarly makes graphs and figures that help visualizing the document structure.

At the moment of manipulating files with Moodle web services, the urllib2 Python default library was replaced by an external Python library for the communication process. The requests done until then were made easily with urllib2 but the files transfer requires a complex utilization of the library methods.

The Requests library [5] main purpose is to provide all the HTTP capabilities with a simpler use than the urllib2 methods. Urllib2 has available many HTTP uses, but performing complex operations requires a full understanding of the library methods and big initial work. The Requests library was developed to provide these functionalities but following the Python philosophy of making simple source codes.

The following codes have the same code using one or other library. The simplification using Requests is evident.

```
import urllib2
gh_url = 'https://api.github.com'

req = urllib2.Request(gh_url)
password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')
auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)
urllib2.install_opener(opener)
handler = urllib2.urlopen(req)

print handler.getcode()
print handler.headers.getheader('content-type')
```

```
import requests

r = requests.get('https://api.github.com', auth=('user', 'pass'))

print r.status_code
print r.headers['content-type']
```

[7]

4.2.2. Library structure

Looking the Moodle web services API, its functions classification can be seen. They are separated in modules depending on the Moodle parts affected. Functions which change or take courses information are defined in the Moodle core module for courses. This can be detected looking the functions names. So in order to develop this library, to follow the same schema has been decided, keeping this modules structure. Each module has a Python class with the functions that affect to the corresponding Moodle components, so, for example, functions which affect to a course are in the course module.

This modulation scheme promotes the independent development of functions and reduces the source code complexity. Instead of having a file with thousands of sentences, the code can be separated in different source files depending of their functionality. The modules contain only one Python class with the same name to keep this scheme.

Also, two more modules were designed, one superior to all defined modules with web services functions and other inferior.

All functions have some common code, like the code to connect with the Moodle site. What changes at using one function or other are the provided parameters, so common code is defined in the superior module which can be used by all other modules.

The class where this code is placed is called MoodClass and all other classes of the project inherit from it.

The inferior module was included for accessing to the rest of modules, so it can use all their defined functions. With this module called MoodLib is not necessary to call a particular module to use a specific function because all of them are accessible from the MoodLib class.

[7] Example codes obtained from the official Requests library page: https://gist.github.com/kennethreitz/973705

This class also defines a function that cannot be included in other modules because is a function that gives information about the own Moodle web service that provides the enabled functionalities.

All this modules are in the library package, which name is MoodPyth and it should be imported in the Python programs that want to use this library.

Following all these constraints, the next figure represents the resulting structure:

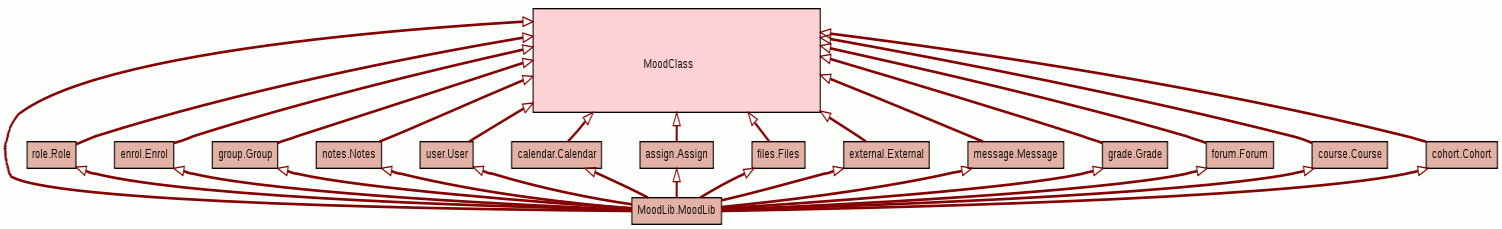

Figure 9: Python library structure

In Figure 9, the produced library flat design can be seen.

4.2.3. Library functions

Here the designed functionalities will be explained, showing the correspondence between the function names in the Moodle web services documentation and the function names defined in the library. The explanation is split in blocks that match with the defined library scheme to show together the available functionalities for each Moodle module.

One thing that should be taken into account is the utilized data structures. The information about a Moodle entity will be composed by dictionaries, a type of structured data with key/value pairs. The keys determine the entity properties, and values give information about those properties. For example, if the entity is a course, the information about that course will be in a dictionary and one of its entrances contains the "name" key and the value contains the course name.

There are four different data types in the Moodle responses: integers, strings, dictionaries and lists. Combining them, every Moodle entity can be represented and its information can be extracted. Some examples of Moodle entities are courses, users,

assignments, events, groups, forums, messages or notes. All these entities are represented by different dictionaries with their specific key/value pairs.

Many implemented functions are for creating, updating, deleting or getting information about Moodle parts. The creating functions add a new entity with the provided parameters to Moodle and return information about the new created object, at least, its unique identifier. The updating functions work in a similar way, changing the value of the existing objects fields with its identifier. To delete the objects is only necessary to give their identifiers in a list. Functions that only return information and which do not change anything in the Moodle site require some field to search the entity desired and return a list of dictionaries with data of those entities.

The library function names correspond with Moodle function names except the name part that indicates at the function module. The Moodle function name has a common string at the beginning of the functions name that belongs to the same module. For example, all the functions defined in the "course" module start with the "core_course_" string, the assignment module function names has the "mod_assign_" as initial string, "group" module has the "core_group_" string, etcetera. The Moodle functions defined in the "external" module has only the "core_" string because this module could contain functions with diverse functionality. The "core_webservice" functions are defined in the MoodLib module because is the library basic unit. All Moodle web services functions can be seen in Appendix 1.

The library is already divided in modules, so this part of the function name is not kept. The library functions use only the second part of the function name string which determines the purpose of the function. So, if the Moodle function name is "core_message_search_contacts", the corresponding library function name is "search_contacts". The library function names will be shown with their corresponding web services function in the following modules definitions.

- **Assign**: this module contains three functions to obtain information about them. The "get_assignments" function retrieves information about the assignments requested through their unique ID field. In the same way, info about the grades established to the students by the teacher and the status of the submitted files by the students in

indicated assignments can be obtained with the “get_grades” and “get_submissions” functions respectively.

get_assignments	mod_assign_get_assignments
get_grades	mod_assign_get_grades
get_submissions	mod_assign_get_submissions

Table 1: Assign module functions

- **Calendar:** can create, delete and get information about calendar events with the “create_calendar_events”, “delete_calendar_events” and “get_calendar_events” functions. The events will appear in the Moodle calendar panel and they notify planned reunions, seminaries, and other happenings. Those events can be at user level, so only the user that created that event can see it, at course level, so it can be seen by all the users that can enter in that course, or at system level, so the event warns at everyone. One bug found in the “get_calendar_events” function is that it only returns events created with web services, not those created with the web interface. The function behavior is determined in Moodle core files, which should be changed by Moodle developers for future versions.

create_calendar_events	core_calendar_create_calendar_events
delete_calendar_events	core_calendar_delete_calendar_events
get_calendar_events	core_calendar_get_calendar_events

Table 2: Calendar module functions

- **Cohorts:** It has the “create_cohorts”, “delete_cohorts”, “update_cohorts” and “get_ cohorts” functions and they do what their names say. The cohorts are used to classify users into groups inside the existing Moodle categories or in the entire system. Therefore, functions to add and remove users from cohorts are necessary. This can be done with the “add_cohort_members” and “core_cohort_delete_cohort_members”

functions. Also, the cohort members can be known through the function "core_cohort_get_cohort_members".

add_cohort_members	core_cohort_add_cohort_members
create_cohorts	core_cohort_create_cohorts
delete_cohort_members	core_cohort_delete_cohort_members
delete_cohorts	core_cohort_delete_cohorts
get_cohort_members	core_cohort_get_cohort_members
get_cohorts	core_cohort_get_cohorts
update_cohorts	core_cohort_update_cohorts

Table 3: Cohorts module functions

- **Course:** module is in charge of managing courses and categories. It has the functions for creating, updating, deleting and getting data of categories and courses (the functions are "create_courses", "create_categories", "delete_courses", etc). The course information is its short name, the data of the category which belongs, its summary, the number of sections and many other data. These data are different from the course contents, which are the added resources inside the course. To get those contents the "course_contents" functions should be used. Courses also can be duplicated with the "duplicate_course" function to create a new course with the same structure and resources than another existing course. Also, the contents of a course can be copied directly into other created course with the "import_course" function. As last implemented functionality is the option of deleting the stored modules in a course, which can be activities such as assignments and quizzes or resources as files, with the "delete_modules" function.

create_categories	core_course_create_categories
create_courses	core_course_create_courses

delete_categories	core_course_delete_categories
delete_courses	core_course_delete_courses
delete_modules	core_course_delete_modules
duplicate_course	core_course_duplicate_course
get_categories	core_course_get_categories
get_contents	core_course_get_contents
get_courses	core_course_get_courses
import_course	core_course_import_course
update_categories	core_course_update_categories
update_courses	core_course_update_courses

Table 4: Course module functions

- **Enroll:** the module returns the enrolled users in a course with the "get_enrolled_users". Those users list can be filtered using the "get_enrolled_users_with_capability" for users with specific permissions. The "get_users_courses" returns the courses where the specified users are enrolled. Also, a user can be enrolled with the indicated role with the "manual_enrol_users", which corresponds with the "enrol_manual_enrol_users" Moodle function.

get_enrolled_users	core_enrol_get_enrolled_users
get_enrolled_users_with_capability	core_enrol_get_enrolled_users_with_capability
get_users_courses	core_enrol_get_users_courses
enrol_manual_enrol_users	enrol_manual_enrol_users

Table 5: Enroll module functions

- **Files:** The files control with web services are managed by a different Moodle program and to access to its functionalities, another URL must be used. Uploading files to Moodle are done with the http://webserverurl.com/moodle/moodle/webservice/upload.php resource. The “upload_files” function will be in charge of this functionality using this URL. However, the files can only be uploaded into a private file area which can be seen only by the own user.

  With the normal web services resource, the “get_files” function was implemented, and its purpose is browse files in the Moodle site, not downloading contents. Using this function requires a deep knowledge of the Moodle file system, which is not intuitive.

  The file system delegates the responsibility of the resources access to the modules where those resources are stored. If a course has a file as resource, this file will be accessible from its URL, which has to be provided by a course module function. In the same way, files uploaded to assignments or to forums will only be accessible if those modules give their URLs with a function.

  Downloading files function is implemented but it only can download text plain files and cannot to work alone because there is not a general way to get contents from Moodle. The Moodle file system separates its contents in a folders structure with numbers and values that depend on the module where the file have been placed. So the downloading files capability is delegated to those modules which can contain files. All downloadable contents are available as resources, and those resources URLs are set by the module that contains them. So if a file is uploaded to an assignment, the assign module is responsible of giving or not its resources URLs. The course module is currently the only one that provides this functionality, giving all course resources files with their download URLs. These URLs can be given as parameter to the "download_files" function, and the returned value of the function will be the file content. Once the content is obtained, it can be saved in a file, printed by console or whatever that the application wants to do with it.

get_files	core_files_get_files
upload_files	/webservice/upload.php
download_files	/webservice/pluginfile.php

Table 6: Files module functions

- **Forum:** this module implements the "get_forum_discussions" function, which get information about the discussions that are being carried out in the requested forums using their ID. The information about forums placed in a course can be obtained with the "get_forums_by_courses" function and the course ID.

get_forum_discussions	mod_forum_get_forum_discussions
get_forums_by_courses	mod_forum_get_forums_by_courses

Table 7: Forum module functions

- **External:** the module provides functionalities that do not imply of the Moodle entities use. Currently, only "get_strings" and "get_component_strings" functions are implemented. The strings purpose is to get language text strings to use in the user interface, to show the best text in every language for each user.

get_component_strings	core_get_component_strings
get_string	core_get_string
get_strings	core_get_strings

Table 8: External module functions

- **Grade:** module has the “get_definitions” function, which provides information about an activity grading. The activities are identified by a cmid (course module identifier) because course modules can be activities and resources. The activity grading system can be configured to evaluate depending of the criterion of the teachers. The evaluation system information can be obtained using this function.

get_definitions	core_grade_get_definitions

Table 9: Grade module functions

- **Group:** manages the association of users in two structures, the groups which link at users and the groupings that contain groups. Their utility is similar to cohorts but in specific courses. The groups and groupings can be created, deleted and updated with habitual functions (“create_groups”, “delete_groups”, “update_groups”, “create_groupings”, “delete_groupings” and “update_groupings”). Other functions are to get the groupings from courses (“get_course_groupings”), get groups form courses (“get_course_groups”), get grouping details with their ID (“get_groupings”) and the same for groups (“get_groups”). Users can be added to a group (“add_group_members”) and removed from it (“delete_group_members”). Also a group can be assigned to a grouping (“assign_grouping”) and unassigned from it (“unassign_grouping”). Finally, there is a function to get the identifiers of users belonging to a group (“get_group_members”).

add_group_members	core_group_add_group_members
assign_grouping	core_group_assign_grouping
create_groupings	core_group_create_groupings
create_groups	core_group_create_groups
delete_group_members	core_group_delete_group_members
delete_groupings	core_group_delete_groupings
delete_groups	core_group_delete_groups

get_course_groupings	core_group_get_course_groupings
get_course_groups	core_group_get_course_groups
get_group_members	core_group_get_group_members
get_groupings	core_group_get_groupings
get_groups	core_group_get_groups
unassign_grouping	core_group_unassign_grouping
update_groupings	core_group_update_groupings

Table 10: Group module functions

- **Message:** The Moodle module gives an instant messaging system with a contact list, a searcher of users and simple messaging tools as blocking users and sending messages. The implemented functions are: searching contacts by their name (“search_contacts”), adding users to the contacts list(“create_contacts”), removing contacts from the list (“delete_contacts”), blocking and unblocking contacts (“block_contacts”, “unblock_contacts”), getting the contacts list (“get_contacts”) and sending instant messages to the contacts (“send_instant_messages”). During the implementation of the last function, an error on the Moodle core function was found and fixed, so the Moodle code was changed. The code fix details are in the library documentation.

block_contacts	core_message_block_contacts
create_contacts	core_message_create_contacts
delete_contacts	core_message_delete_contacts
get_contacts	core_message_get_contacts
search_contacts	core_message_search_contacts
send_instant_messages	core_message_send_instant_messages
unblock_contacts	core_message_unblock_contacts

Table 11: Message module functions

- **Notes**: manage short personal notes about users. The notes always are in a course, but they can be personal, for the course and for the entire site. Those notes can be created (“create_notes”), deleted (“delete_notes”) and retrieved (“get_notes”).

create_notes	core_notes_create_notes
delete_notes	core_notes_delete_notes
get_notes	core_notes_get_notes
update_notes	core_notes_update_notes

Table 12: Notes module functions

- **Roles:** module can only assign (“assign_roles”) and unassign (“unassign_roles”) roles to users using the identifiers of the roles and the users.

assign_roles	core_role_assign_roles
unassign_roles	core_role_unassign_roles

Table 13: Roles module functions

- **User**: module provides the usual functions of a module: “create_users”, “delete_users”, “update_users” and “get_users”. The “get_users” function is thought for searching users that match some of the user data fields provided. To get a specific user data the “get_users_by_field” function should be used. Also, the user profiles in different courses can be obtained with the course ID and the user ID using the “get_course_user_profiles” function.

create_users	core_user_create_users
delete_users	core_user_delete_users
get_course_user_profiles	core_user_get_course_user_profiles
get_users	core_user_get_users
get_users_by_field	core_user_get_users_by_field
update_users	core_user_update_users

Table 14: User module functions

- **MoodLib:** Finally, the "core_webservice_get_site_info" is implemented in the MoodLib module, and get general information about the web service used, the Moodle site used, and the user that do the request.

get_site_info	core_webservice_get_site_info

Table 15: MoodLib module functions

After implementing all these library modules, one hundred percent of the web services functions in the Moodle version 2.5 have been covered. So the development section of this project cycle has concluded.

4.2.4. Library use

Now, the testing phase can be started to ensure that all implemented functions work fine. For it, each module has its corresponding test file in the project files. These files prove every module function independently with different parameters and try to use the functions on every possible way. They check the appearing of the function data in correct sentences and the appropriated error messages in malfunction tests. After using the library functions, the tests print on console the returned data with an auxiliary function to show their structure but they

do not do anything useful as application. Therefore, a function use example will be used for explaining the required sentences and the data manipulation to develop programs.

The library use consists in two simple steps: initialization and execute functions. On the initialization the common parameters to every request will be set, and their values will be saved in class attributes for future requests.

The Moodle URL (shown in section 4.1.1) is required to use the Moodle Python Library, given as a parameter to the library class constructor. The entire web server URL must be specified so the class can send requests to the Moodle site, including the "http://" string and the Moodle relative location without the string "moodle". So if the web server URL is "http://webserverurl.com/relativepath/moodle/", the URL which should be given to the class constructor is: "http://webserverurl.com/relativepath"

The user's token is the other necessary parameter to call at the class constructor of the library because is used to authenticate at the user on every executed function. It can be set directly with the constructor's token parameter or it can be requested to Moodle as explained in section 4.1.3. This request is made automatically by the library when the token is not provided to the class constructor. If the user's name, password or service short name are not provided or are wrong, a "ValueError" exception will be thrown specifying the problem. If the token parameter is provided, this request will not be made even if the user name, password and the service short name are indicated.

If the provided token is invalid, the constructor will not return any error because it is not checked with Moodle. The error will appear when using any Moodle function saying that the token is not valid.

With the Moodle URL and the user's token, every function can be already used. A function usually needs some specific parameters for execute and other optional fields. Those parameters change for every function and are detailed in the library documentation. When a library function is executed with valid parameters, the library generates the corresponding request with the parameters given, sends it to Moodle, obtains the response in JSON format and returns the data on a Python structure specified in the documentation.

4.3. Develop short programs using this library

With web services activated and the library implemented, a short program will be developed as use example. The example will use the same Moodle functionality than the one performed in section 4.1.5 to see the library use advantages.

In first place a variable will be used to make reference at the MoodLib class instance that makes possible the communication with Moodle. The Moodle web site URL and the authentication parameters will be given to the class constructor. The parameters can be the token or the user, the password and the web service's short name. If token is used the constructor call will look like:

```
url = 'http://adry3000.dyndns.org/'
token = '77246fc16e1198dbd171e7f467dfa906'
t = MoodLib(url, token)        # class constructor using token
```

Otherwise, if the user, password and service short name are used, it would be:

```
url = 'http://adry3000.dyndns.org/'
username = 'MyUser'
pasword = 'MyP@ssword'
service_short_name = 'test_service'
t = MoodLib(url, user= username, pasw= pasword, service= service_short_name)        # class constructor using user, pasword and service name
```

After, the variable defined executes its “get_contents” function, with the identifier of one existing Moodle course as necessary parameter. The value returned by the function will be stored in other variable that will keep the course contents. The data structure is defined both in the Moodle API and the library documentation. So the problem complexity now has become into understand and manipulates returned data structures. For example, the course’s files can be named in a list with those data. This can be done going through the course sections, through their modules and their contents and if the type field identifies the content as a file, show its file name.

The following example shows the resulting source code.

```
from MoodPyth.MoodLib import MoodLib

if __name__ == '__main__':
    url = 'http://adry3000.dyndns.org/'     # initializes constructor parameters
    token = '77246fc16e1198dbd171e7f467dfa906'
    t = MoodLib(url, token)         # class constructor
    contents = t.get_contents(2)    # get contents form the course which identifier is 2
    for section in contents:        # go through course sections
        if 'modules' in section:
            for module in section['modules']:   # go through section modules
                if 'contents' in module:
                    for content in module['contents']:  # go through module contents
                        if str(content['type'])=='file':    # if the content is a file
                                print content['filename']       # its name is printed
```

Another use more realistic example could be the next situation. The students data are stored in some external support such as in files or a in a database. A script that get those data and create new users in the Moodle system with the students' information can be written, automating the users' creation.

These examples show the wide range of possibilities that those functions perform. They can be implemented without needing too complex programs. However, the potential applications with web services are limited by the provided Moodle functionalities with these services.

Errors returned while using the library

On program developing time, the possible errors obtained at using the library should be taken into account. Those errors can be grouped in two big categories.

The first error type is produced as consequence of library functions parameters misusing. These errors stop the program execution before sending any message to Moodle. These problems are indicated with a Type Error exception with different messages depending of the issue found. They can appear because a function parameter type does not match with the required type, or because a required key in an input parameter structure is not found. For example, a string is given

when an integer is necessary, or a dictionary identifier field is required and it is not found. If a necessary dictionary key is not found, the message error will specify the missing key name.

The second type is a problem found in the server execution action. On this situation, the request is made to Moodle and it returns an error which is processed by the library as a Value Error python's exception. These mistakes can occur for many different reasons, as a value type error or getting missing information (for example, get details about a non-existing user). The errors messages will go with a short problem description. However, sometimes this description is not enough to know the problem's origin. If debugging option can be set, Moodle will return more detailed information about the problem and the library will automatically add it to the exception information. This mode can be activated with the administrator account, entering in *Site administration* → *Development* → *Debugging* panel and setting the *Debug messages* option to *Normal.*

5. Results and conclusions

The main project conclusion is that Moodle web services have many potential uses, both for administration tasks and for external applications development. However, the current available functions provides general Moodle site operations as creating, deleting, updating or getting information about different Moodle entities (assignments, courses, users, forums, events, etc.) but many available options in the web interface are unimplemented in the web services interface. As a result, the present functionalities can be exploited with short scripts and programs with a specific purpose, but they cannot give the necessary functionalities to develop a full web services-based application to replace the web interface.

At the beginning of the project, a study of the different existing Moodle versions was done to include all the use possibilities. The interfaces have been added along the Moodle development and this study reveals the options and their improvements. Contemplating all the options, the current version was used on the rest of project.

The Moodle interfaces have been presented and explored, showing its interaction options with applications, which can be normal web browsers, applications for using Moodle or other systems with different purposes. The most recent of these interfaces, the Moodle web services, was analyzed to find out its possibilities. This work has clarified the Moodle web services status and their utility. For it, a deep search of documentation was done in the Moodle official web page and in their forums. The initial steps for installing Moodle and for using the web interface were easy to find and to carry out.

However, despite that the web services activating and tokens management are documented, they are not prepared to be understood easily by new users of Moodle. Furthermore, this documentation is dispersed in several web pages and forums and it is not located in a specific area that collects all the necessary information.

Besides, the documentation of web services utilization is poor and not much detailed, using a few examples as base for its understanding. Moreover, most of those examples are written in PHP, the Moodle native code, what adds difficulties to rookies in that programming language. In

addition, the descriptions of the web services functions are schematic and not too much intuitive at first glance, so the initial tests with the web services were hard to be performed.

So once the interacting with web services was learned, a library for simplifying its use would be more useful than a concrete application. This would imply three important steps: implementing all the web services functions, testing each one of them and documenting all the methods details.

Once the structure of the functions parameters are understood, the implementation process went faster than what looked like because the methods are very similar between them. However, checking those implementations delayed the development for doing several tests which can be taken as use examples. Besides, the documentation was not easy for having to understand the variables meanings with the provided documentation and because finding some incoherencies between the functions implementation and them working way. The library documentation was extracted from the web services API, changing the text so it can be understood easily but it is limited by the official documentation provided.

These steps took some time but they were not especially complicated. However, at the moment of having a problem with any function, the solutions to those problems were not obvious and the documentation did not help too much. So, each one of these issues delayed the project end.

Summarizing, the project presents all the current Moodle interaction ways with their corresponding advantages and disadvantages, provides a full guide to activate Moodle web services, gives the communication process details through the REST API, shows basic Python program sentences to use this interface, simplifies the services utilization through a Python library fully tested and documented including every functionality, expose how the library could be use for developing scripts and collects all the information for a complete understanding of the Moodle web services interface. Furthermore, all this project results are available on the net in the following web page:

https://github.com/javibgm/MoodlePython

Future work

The implemented functions at the server-side are the main web services limitation, so the first step is to improve these services participating in the Moodle community, adding new and better functionalities. The study has shown that the course module is one of the most complete, showing the courses contents and offering their download links. These and other functionalities would be very useful in other modules that are not too much developed. These services can be supplemented with web scrapping to increase the library functionalities without needing to create the corresponding web services for that function.

Besides a better official documentation about the web services functions use would be very helpful. The web services activation process is well specified but is not linked with the rest of the necessary documentation to use them effectively. The web services API could be changed to be more intuitive and showing its contents in a better way, for example adding indentations to the general structure data, detailing the used parameters meaning and including some use examples to show what the functions do with different parameter values.

In the same way, if new functions are added, they should be implemented in this project's python library to keep it updated. The current functions can also be enhanced, for example, doing the download files function can download all kind of documents, and not only text plain files. The library also could be improved, for example, adding security on messages transporting through the HTTPS protocol instead of HTTP, supporting other transport protocols or making a program which imports the official documentation directly into the library documentation.

A program for external applications automatic integration could be another Moodle enhancement. The processes of activating web services, adding their functions, setting a external service short name and all the Moodle configuration steps could be automated or, at least, guided with a program. This program could do the timely changes in Moodle and requests for the necessary parameters on each step, specifying the data needed to complete the Moodle configuration to use a concrete application.

6. Bibliography

[1] Roy T. Fielding. Architectural styles and the design of network-based software architectures. *PhD Thesis*, University of California, Irvine, 2000

[2] Guido van Rossum. Python Programming Language.

Official Website: http://www.python.org/

[3] XML. Extensible Markup Language Specification: http://www.xml.com/axml/testaxml.htm

[4] JSON (JavaScript Object Notation): http://www.json.org/ http://en.wikipedia.org/wiki/JSON

[5] Requests Python Library. Requests: HTTP for Humans. http://docs.python-requests.org

[6] Moodle Official Web Site: https://moodle.org/

[7] Moodle 2.5 Official Documentation Page: http://docs.moodle.org/25/en/Main_page

[8] HTTP - Hypertext Transfer Protocol: http://www.w3.org/Protocols/

[9] Rasmus Lerdorf. PHP Hypertext Pre-processor, 1995. Official website: http://php.net/

[10] JavaScript programming language: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[11] HTML - HyperText Markup Language. World Wide Web Consortium & WHATWG. HTML Main Web Page: http://www.w3.org/html/

[12] LMS- Learning Management Systems: http://hal.archives-ouvertes.fr/docs/00/69/20/67/PDF/Watson-2007.pdf

[13] SCORM - Sharable Content Object Reference Model: http://www.adlnet.org/scorm

7. Appendices

7.1. Appendix 1

List of web services functions that the Python library supports:

core_calendar_create_calendar_events
core_calendar_delete_calendar_events
core_calendar_get_calendar_events
core_cohort_add_cohort_members
core_cohort_create_cohorts
core_cohort_delete_cohort_members
core_cohort_delete_cohorts
core_cohort_get_cohort_members
core_cohort_get_cohorts
core_cohort_update_cohorts
core_course_create_categories
core_course_create_courses
core_course_delete_categories
core_course_delete_courses
core_course_delete_modules
core_course_duplicate_course
core_course_get_categories
core_course_get_contents
core_course_get_courses
core_course_import_course

core_course_update_categories
core_course_update_courses
core_enrol_get_enrolled_users
core_enrol_get_enrolled_users_with_capability
core_enrol_get_users_courses
core_files_get_files
core_files_upload
core_get_component_strings
core_get_string
core_get_strings
core_grade_get_definitions
core_group_add_group_members
core_group_assign_grouping
core_group_create_groupings
core_group_create_groups
core_group_delete_group_members
core_group_delete_groupings
core_group_delete_groups
core_group_get_course_groupings
core_group_get_course_groups
core_group_get_group_members
core_group_get_groupings
core_group_get_groups
core_group_unassign_grouping

core_group_update_groupings
core_message_block_contacts
core_message_create_contacts
core_message_delete_contacts
core_message_get_contacts
core_message_search_contacts
core_message_send_instant_messages
core_message_unblock_contacts
core_notes_create_notes
core_notes_delete_notes
core_notes_get_notes
core_notes_update_notes
core_role_assign_roles
core_role_unassign_roles
core_user_create_users
core_user_delete_users
core_user_get_course_user_profiles
core_user_get_users
core_user_get_users_by_field
core_user_update_users
core_webservice_get_site_info
enrol_manual_enrol_users
mod_assign_get_assignments
mod_assign_get_grades

mod_assign_get_submissions
mod_forum_get_forum_discussions
mod_forum_get_forums_by_courses