PROYECTO DE GRADO

Presentado ante la ilustre Universidad de Los Andes como requisito final para obtener el Título de Ingeniero de Sistemas

Desarrollo de un Esquema de Rendimiento para MOODLE

Br.Gustavo Alejandro Mejía Briceño
Tutores:
Yaneth Moreno, M.Sc
Francisco Hidrobo, Ph.D

Mérida, Febrero de 2021

Resumen

En el presente trabajo se expone el proceso de diseño e implementación de un esquema transaccional que busca optimizar el acceso a recursos digitales (documentos, archivos multimedia, entre otros) por parte de los usuarios de plataformas e-learning basadas en el proyecto Moodle, aplicando una metodología basada en funcionalidades (FDD por sus siglas en inglés). El esquema transaccional consiste específicamente de una extensión creada con el fin de intermediar entre la interfaz de usuario de Moodle y las librerías que manejan el almacenamiento de datos y recursos. Además, solicita recomendaciones sobre los recursos más concurridos de la plataforma a un recomendador externo e implementa una base de datos en memoria utilizando Redis en la que se almacenan los archivos más descargados por los usuarios.

Palabras clave: Almacenamiento, Esquema transaccional, EVA, Moodle.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Con el pasar de los tiempos, la evolución de las tecnologías de información y comunicación han traido al mundo cambios en todo aspecto de la vida humana, facilitando entre algunos las comunicaciones, los negocios y por supuesto, la educación. En los ultimos años, las plataformas dedicadas a la formación de profesionales a distancia han ganado tal popularidad que es incalculable la gran cantidad de usuarios que estas poseen, ofreciendo una amplia gama de carreras y areas de formación con el fin de brindar oportunidades a quienes no cuentan con los medios para disfrutar de una educación prescencial universitaria.

Pero a pesar del apoyo y soporte que hay detras del mundo del aprendizaje electrónico (*e-learning*), estas plataformas no escapan de la realidad de los sistemas web en general, y es que debido a la gran cantidad de usuarios que acceden día a día a distintos cursos en línea y de manera simultanea, los tiempos de respuesta pueden tornarse un problema a gran escala en especial si no se cuenta con los medios necesarios para adaptar los equipos a los nuevos requerimientos de los sistemas.

Entre tantas plataformas de aprendizaje electrónico podemos resaltar el proyecto de software libre Moodle. Esta se define como una plataforma de aprendizaje, diseñada para proporcionar a educadores, administradores y estudiantes un sistema integrado único, robusto y seguro para crear ambientes de aprendizaje personalizados (Gabriel Barrios, 2019).

En este trabajo se presenta el desarrollo de un componente que optimizará la gestión de almacenamiento en las plataformas e-learning haciendo uso de memorias caché para almacenar los recursos que se determinen deban estar a primera mano.

Este será implementado en Moodle y se encargará de efectuar las transacciones entre la memoria secundaria y el servidor web de Moodle siguiendo las reglas especificadas para su comunicación con la finalidad de mantener una copia de los recursos requeridos en la memoria caché, reduciendo el tiempo necesario para brindar respuesta a futuras solicitudes.

1.1. Antecedentes

Hoy en día existe una gran cantidad de componentes y extensiones (*plugins*) creados para Moodle con el fin de cubrir una necesidad de la plataforma o brindar una funcionalidad adicional a esta. Estos pueden ser instalados de forma manual o desde la misma página oficial de moodle (`moodle.org`). Una categoría importante es la de los *plugins* que integran memoria caché, entre los cuales podemos resaltar **Alternative PHP Cache (APC)** (Hemelryk, 2014) el cual proporciona un caché de tamaño limitado pero de excelente desempeño destinado al almacenamiento de datos de aplicaciones persistentes de PHP; **Memcache Cluster** (Merrill, 2014) que implementa una versión modificada de la memoria caché estándar de Moodle que permite mantener varios almacenamientos memcache sincronizados entre sí.

Con el fin de lograr los objetivos planteados en este trabajo, se hará uso de la librería AdoDB, la cual no es más que una capa de abstracción de base de datos popular, rápida y fácil de usar para PHP. Una aplicación de esta librería la podemos conseguir en (Mgheder y Ridley, 2008) en el cual se propone estudiar la posibilidad de usar metadatos almacenados en bases de datos para desarrollar elementos de interfaz de usuario. En este trabajo se hace uso del lenguaje PHP en conjunto con la librería AdoDB por ser un buen candidato para generar interfaces de usuario web dinámicamente.

Por otra parte, en (Sanchez et al., 2017) se presenta un framework para PHP basado en el modelo de 3 capas con el fin de identificar y separar la aplicacion final en diferentes capas que faciliten su construcción y mantenimiento. Este enfoque integra diferentes tecnologías y patrones de diseño con el fin de proporcionar una herramienta eficaz que respalde a la comunidad en la creación de aplicaciones web con PHP, entre estas, se integra la librería AdoDB en la capa de acceso a datos para gestionar la comunicación entre la aplicación y la base de datos relacional.

(García, 2017) realiza el diseño e implementacion de una aplicación móvil para la intranet de la Universidad Politécnica de Madrid basada en el framework cordova y tecnologías web. La idea de este trabajo, fué proponer la programación híbrida usando el patrón MVC (Modelo-Vista-Controlador) como una forma de programación que sustituiría el desarrollo ordinario para plataformas nativas, evitando el desarrollo en paralelo para cada sistema operativo móvil, como Android o iOS, y brindando una mejor implementación. El propósito de usar la librería AdoDB en este trabajo es, principalmente, evitar el mantenimiento extra en el caso de migraciones de la base de datos o cambios en los controladores de esta. Tambien, la implementacion de AdoDB brinda a los desarrolladores la facilidad de usar el mismo código para acceder a una amplia gama de bases de datos.

1.2. Planteamiento del Problema

El rendimiento de una aplicación web es importante, pero en una plataforma de formación es crucial. La percepción del alumnado sobre la calidad de los cursos puede verse deteriorada si durante su formación la plataforma responde de forma lenta o inesperada. Los sistemas de enseñanza aprendizaje apoyados por *e-learning* constituyen actualmente la nueva tendencia en la web para aquellas instituciones que apuestan por la educación a distancia o semipresencial, bien sea impartiendo cursos, diplomados, estudios de pregrado, estudios de postgrado, entre otros programas de formación académica, ejemplos de ello en Venezuela, podemos mencionar, la Universidad Centro Occidental Lisandro Alvarado (UCLA) a través del dictado de algunos diplomados a distancia, la Universidad de Carabobo (UC) impartiendo la Especialización en Desarrollo de Software con modalidad semipresencial y la Universidad de Los Andes (ULA) con el programa de formación de la carrera de Derecho a distancia, el cual es impartido por medio de la Plataforma de Estudios Interactivos a Distancia de la ULA (EIDIS), esto reafirma el interés de las instituciones educativas por favorecer a aquellas personas que se les dificulta tener acceso a la formación presencial. Es aquí donde la elección de una plataforma de aprendizaje ideal entra en juego, y en este sentido Moodle es una de las opciones más empleadas para tal fin por distintas instituciones educativas. La gestión del almacenamiento en Moodle es un aspecto significativo, ya que allí se manejan los recursos más valiosos

con los cuales se transmite la formación al participante, como lo son, los archivos, y dependiendo del desempeño que provea esta parte de la plataforma, para que dichos recursos se gestionen eficientemente cada vez que el usuario los requiera, el proceso de enseñanza se percibirá bastante oportuno y rápido. Así, surgue la necesidad de aumentar la velocidad y la eficiencia de las aplicaciones web, y de manera específica de Moodle, apuntando a enriquecer la calidad en cuanto a la experiencia de uso, es decir que el servicio satisfaga las expectativas y necesidades del usuario, y por otro lado, a mejorar la calidad de servicio, referida a la medida del rendimiento del sistema. El problema planteado en este trabajo, es el requerimiento de Entornos Virtuales de Aprendizaje (EVA) más eficientes para gestionar los requerimientos de usuarios cada vez mas exigentes, y que ellos perciban u observen que su productividad no esta siendo afectada. En lo particular, se requiere mejorar el rendimiento en cuanto a la gestión de almacenamiento, para ello se debe detectar necesidades y comportamientos a través de los procesos que se llevan a cabo entre la librería AdoDB y la base de datos de la plataforma Moodle, que nos permitirán desarrollar e implementar al menos un esquema de rendimiento apropiado que se integre a la plataforma, que permita al servidor acceder más rápido a los archivos que soliciten los usuarios.

1.3. Justificación del Proyecto

Si bien es cierto que la eficiencia de un sitio web influye enormemente en la experiencia del usuario, cuando se trata de una plataforma de aprendizaje es crucial que los tiempos de respuesta sean rápidos, pues esto ayudará a que el interes de los alumnos no decaiga. Tomando en cuenta esto, en este trabajo se busca optimizar el sistema gestor de almacenamiento de las plataformas e-learning, tomando como caso de estudio Moodle, con el fin de minimizar las deficiencias que poseen estos sistemas a la hora de gestionar los datos y recursos de los distintos cursos y usuarios inscritos.

Adicionalmente, es importante mencionar que este proyecto se encuentra enmarcado en una linea de investigación doctoral que trata sobre un Sistema de Gestión de Almacenamiento para entornos e-learning basado en Big Data, el cual requiere el diseño, desarrollo e implementación de un esquema de transacción de datos, un componente que actuará de intermediario entre la interfaz de usuario de Moodle y el servidor de almacenamiento de datos y recursos.

1.4. Objetivos del Proyecto

El objetivo general de este trabajo es desarrollar a nivel de backend un esquema transaccional de datos en la plataforma *e-learning* Moodle.

Para ello se debe cumplir con los siguientes objetivos específicos:

1. Analizar el modelo de datos y el rendimiento de los procesos de almacenamiento que se llevan a cabo dentro de la plataforma Moodle.

2. Estudiar los procesos que se llevan a cabo entre la librería AdoDB y la base de datos de Moodle.

3. Determinar el alcance y requerimientos del esquema de transacciones de datos a desarrollar.

4. Diseñar el esquema transaccional.

5. Implantar el esquema diseñado en la plataforma Moodle.

1.5. Metodología

En la actualidad, la agilidad al cambio es un factor de suma importancia en los proyectos de software. Los requisitos y diseños tienden a cambiar rapidamente con el tiempo para adaptarse a las necesidades del proyecto. Por esta razón, existen las *metodologías ágiles*, que nacen con el fin de brindar un mayor enfoque al proceso de desarrollo, enfatizar la comunicación cara a cara en lugar de la documentación y en especial facilitar la refactorización y la adaptación a los cambios. En este proyecto se pleantea aplicar un Desarrollo basado en funcionalidades o *Feature Driven Development* (FDD), puesto que se trata de una metodología ágil basada en la calidad y el monitoreo constante del proyecto. Se enfoca en iteraciones cortas, que permiten entregas tangibles del producto en un período corto de tiempo.

La metodología FDD define 5 procesos: Proceso 1 - Desarrollar un modelo global, Proceso 2 - Desarrollar una lista de funcionalidades, Proceso 3 - Planificar por funcionalidad, Proceso 4 - Diseñar por funcionalidad y Proceso 5 - Implementar por funcionalidad.

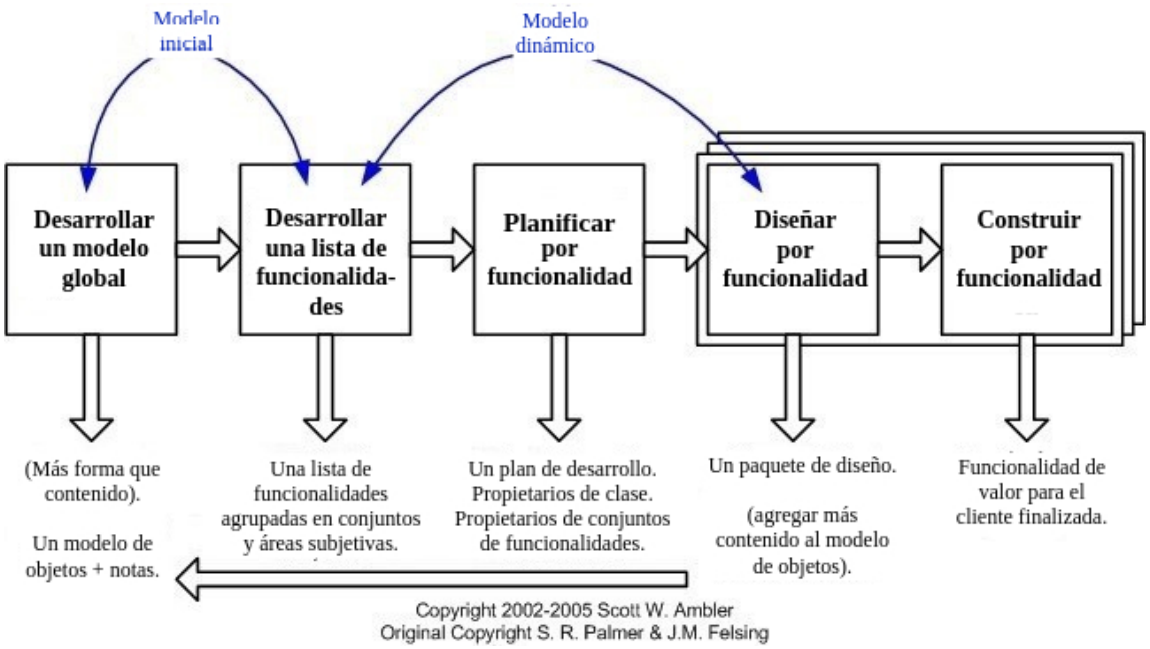


Figura 1.1: Procesos de la metodología FDD (modificado de (Ambler, 2002))

Con el fin de adaptar estos 5 procesos a nuestro trabajo, se especificaron, para cada uno ellos las siguientes tareas:

Proceso I: Desarrollar un modelo global.

- Analizar el modelo de datos y el rendimiento de los procesos de almacenamiento de Moodle.
- Estudiar los procesos que se llevan a cabo entre la librería AdoDB y la BD Moodle.
- Determinar el alcance y requerimientos del esquema a desarrollar.
- Determinar el modelo de objetos.

Proceso II: Construir una lista de funcionalidades.

- Construir una lista de funcionalidades en base a los requerimientos previamente determinados.

Proceso III: Planificar.

- Construir el plan de desarrollo:
  - Determinar el orden en que se desarrollarán las funcionalidades.
  - Determinar el tiempo estimado que tomará desarrollar e implementar cada funcionalidad.

Proceso IV: Diseñar.

- Especificar las funcionalidades del esquema de rendimiento adecuado que defina la solución de los problemas de gestión en la plataforma Moodle.
- Elaborar la documentación detallada de cada funcionalidad:
  - Diseño general de funcionalidades.
  - Diagramas de secuencia.
  - Diagramas de clases.

Proceso V: Implementar.

- Desarrollar e implementar las funcionalidades en el orden especificado en el plan de desarrollo.
- Implantar el esquema de optimización desarrollado en la plataforma Moodle.

1.6. Alcances

En el presente trabajo se propone el diseño e implementación de un esquema transaccional de datos y recursos procurando mejorar el rendimiento de las plataformas e-learning y en general de cualquier sistema web. Este será implementado y probado en Moodle. Entre las tareas que se deben llevar a cabo es importante mencionar el estudio y análisis de los procesos internos de Moodle así como su comunicación con la librería AdoDB y distintos *plugins* y componentes que se encuentran integrados a esta o que pueden ser instalados por parte del administrador del sistema con el fin de conseguir la forma óptima de implementar el esquema previamente mencionado.

Capítulo 2

Marco Referencial

En éste capítulo, se pretende exponer las bases teóricas necesarias para la comprensión del diseño e implementación de un esquema transaccional que optimice la gestión de archivos en Moodle.

2.1. Entorno Virtual de Aprendizaje(EVA)

Un Entorno Virtual de Aprendizaje, mayormente conocido como plataforma *e-learning*, hace referencia a una plataforma web destinada a la enseñanza en línea mediante un método que puede ser completamente no presencial, en el cual el aprendizaje se hace completamente mediante Internet, o mixto, combinando la enseñanza en línea con experiencias en el salón de clases.

"El objetivo primordial de una plataforma *e-learning* es permitir la creación y gestión de los espacios de enseñanza y aprendizaje en Internet, donde los profesores y los alumnos puedan interactuar durante su proceso de formación" (Gabriel Barrios, 2019), facilitando la comunicación pedagógica entre los participantes de un proceso educativo en lo que (Bello Díaz, 2005) denomina un "aula sin paredes", distal y multicrónica.

(Gabriel Barrios, 2019) en su trabajo, clasifica a los entornos virtuales de aprendizaje con respecto a su funcionalidad, pudiendo ser plataformas de carácter general o específicas. Las plataformas de carácter general se caracterizan principalmente por no enforcarse en una sola catedra, al contrario, estan orientadas al aprendizaje de distintos tópicos. Entre estas destacan como las más utilizadas los Sistemas de Gestión

del Aprendizaje (SGA) como por ejemplo Moodle, .LRN, e-College, Desire2Learn, entre otras.

Los SGA resaltan por ser los más completos en cuestión de características y funcionalidades, entre las principales y más comunes que poseen la mayoria de estos sistemas se pueden mencionar: la administración del EVA, comunicación de los participantes, gestión de contenido, gestión del trabajo en grupos y, la evaluación.

En el caso de los sistemas dedicados al desarrollo de una destreza o aprendizaje de una materia en específico, podemos destacar las plataformas orientadas al aprendizaje de las lenguas, que integran herramientas que se adaptan a las metodologías específicas de enseñanza de esta competencia (Gabriel Barrios, 2019).

Existen algunas plataformas aun más orientadas a un modelo o método de aprendizaje espécifico, entre estas existen los Entornos Personales de Aprendizaje (PLE, por sus siglas en inglés). Inspiradas en el fenómeno de la **Web Social** o **Web 2.0**, un enfoque que enfatiza en la colaboración online, conectividad y compartir contenidos entre usuarios (`https://disenowebakus.net/la-web-2.php`). "Están basadas en el modelo de aprendizaje socio-constructivista en el que el aprendiz es protagonista de su propio aprendizaje, cooperando y colaborando con el grupo para construir nuevos conocimientos." (Gabriel Barrios, 2019).

2.1.1. MOODLE

Moodle es una plataforma web libre de aprendizaje colectivo y es, en este trabajo, el caso de estudio para la implementación de un esquema transaccional de datos. Esta se encuentra actualmente en la versión 3.7 pero la versión que será implementada en este proyecto será la 3.3

Moodle está diseñada para soportar tanto la enseñanza como el aprendizaje guiado por la pedagogía de constructivismo social, puesto que proporciona un conjunto de poderosas herramientas centradas en el estudiante y ambientes de aprendizaje colaborativo (Gabriel Barrios, 2019).

El proyecto Moodle impulsa decenas de miles de ambientes de aprendizaje globalmente con más de 79 millones de usuarios, entre usuarios académicos y empresariales, que la convierten en la plataforma de aprendizaje más ampliamente utilizada del mundo (Moodle, 2019).

Modulos en Moodle

Los modulos en Moodle son grupos de características en un curso. Significa propiamente algo a lo que los estudiantes pueden contribuir directamente, y a menudo es contrastada con un recurso, como por ejemplo un archivo o una página, el cual es presentado por el profesor a los alumnos.

Los modulos principales en Moodle son: tarea, consulta, foro, diario, cuestionario, recurso, encuesta, wiki, taller. Éstos se encuentran detallados en (Gabriel Barrios, 2019).

Extensiones en Moodle

En informática, un complemento o «*plug-in*» es una aplicación (o programa informático) que se relaciona con otra para agregarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la interfaz de programación de aplicaciones.

Moodle cuenta con un amplio directorio de extensiones que permiten añadir características y funcionalidades adicionales, como por ejemplo, nuevas actividades, nuevos tipos de preguntas para exámenes, nuevos reportes, integraciones con otros sistemas y muchas más[1].

2.1.2. Estructura de un *plugin*

En (Ivorra Oltra y Luján-Mora, 2009) se describe el esquema estandar de un *plugin* de Moodle de la siguiente manera:

Nombre	Tipo	Descripción
version.php	Archivo	Contiene la meta información sobre el *plugin*, por ejemplo, la versión de este.
settings.php	Archivo	Archivo opcional que contiene el formulario con las opciones generales del *plugin*.

[1]Directorio de *plugins*: `https://moodle.org/plugins/`

index.php	Archivo	Sirve para mostrar todas las instancias de una actividad en un curso, es decir, una lista con todas las instancias del mismo *plugin*.
view.php	Archivo	Ésta es la página que muestra una instancia de la actividad.
lib.php	Librería	Librería de funciones del *plugin*. En este archivo se implementarán todas sus funciones y procedimientos.
mod_form.php	Formulario	Formulario para crear o modificar una instancia de la actividad.
lang/	Directorio	Almacenar los archivos de idioma del *plugin*. Este debe contener los archivos de idioma con las cadenas de texto necesarias por el *plugin* en ingles y sus traducciones a los idiomas de los usuarios finales.
db/	Directorio	Directorio donde se almacenarán los archivos con las tablas de las bases de datos necesarias.

access.php	Archivo	Archivo opcional que contiene los permisos del *plugin*. Los permisos no son obligatorios, pero si muy recomendables para garantizar qué usuarios pueden acceder a las distintas partes de este.
install.xml	Archivo	Archivo que describe la estructura de las tablas del *plugin*.
upgrade.php	Archivo	Código de actualización, aquí es donde se deben de hacer las alteraciones de las tablas, si las hay, entre versiones.

Tabla 2.1: Tabla sobre la estructura de un *plugin*.

2.1.3. Instalación de un *plugin*

Existen tres formas de instalar un *plugin* en un servidor Moodle. La primera de estas y la más sencilla es la instalación directa desde el directorio de *plugins* de Moodle, haciendo uso de la herramienta que este tiene que permite la instalación rapida y sencilla del *plugin* deseado desde la misma página del proyecto.

En segundo lugar la instalación mediante un archivo ZIP cargado al servidor, esta consiste en descargar un archivo comprimido que contiene al *plugin* y posteriormente cargarlo al sitio web desde su herramienta local.

Por ultimo, la instalación manual en el servidor. Esto significa que el administrador se encarga de copiar manualmente el código a la carpeta correcta del sistema de archivos del servidor web.

2.2. Memorias caché

Una caché es un componente de hardware o software que almacena datos para que las solicitudes futuras de esos datos se puedan atender con mayor rapidez.

En Moodle existen varias integraciones de memorias cache ya incluidas en el código fuente y otras que pueden ser instaladas mediante *plugins*, algunas de estas son Memcached[2], MongoDB, APC user cache (APCu)[3], XCache[4] y Redis[5].

Para el desarrollo de las funcionalidadeds de este proyecto, se tomará como base la extensión para la gestión de caché de Redis ya integrada en la plataforma. Redis permite el almacenamiento clave-valor lo que facilita el mapeo de archivos en memoria principal.

2.3. Herramientas tecnológicas

- **PHP:** Lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera el HTML resultante.

- **AdoDB:** ADOdb es una capa de abstracción de base de datos popular, rápida y fácil de usar para PHP. Permite utilizar el mismo código para acceder a una amplia gama de bases de datos. Ha sido mantenido activamente desde el año 2000 por el fundador del proyecto y numerosos colaboradores de la comunidad. ADOdb contiene componentes para consultar y actualizar bases de datos, así como una biblioteca de registros activos orientada a objetos, administración de esquemas y monitoreo del rendimiento (Damien Regad y Community, 2014). Moodle hace uso de AdoDB como su capa de abstracción de base de datos por defecto, esta se encuentra incluida en su código fuente y es usada de forma automática al momento de realizarse cualquier consulta a la base de datos.

[2]Memcached: https://moodle.org/plugins/cachestore_memcachedcluster
[3]APCu: https://docs.moodle.org/37/en/APC_user_cache_(APCu)
[4]XCache: https://moodle.org/plugins/cachestore_xcache
[5]Redis: https://docs.moodle.org/37/en/Redis_cache_store

- **Atom:** Atom es un editor de texto de código abierto para macOS, Linux, y Windows con soporte para múltiples *plug-in*, desarrollado por GitHub.
- **Servidor HTTP Apache:** Apache es un software de servidor web gratuito y de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, con el cual se ejecutan el 46 % de los sitios web de todo el mundo. Permite a los propietarios de sitios web servir contenido en la web.
- **MySQL:** MySQL es un sistema de gestión de bases de datos relacional. Está considerada como la base datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

Capítulo 3

Análisis, Planificación y Diseño de funcionalidades

En este capítulo se explicarán los tres primeros procesos de la construcción del *plugin*: Desarrollo de un modelo global, construcción de una lista de funcionalidades y planificación por funcionalidades. Estas tres fases corresponden a la iteración cero, en la que se estableció un modelo global del proyecto y se planificó el desarrollo de las funcionalidades. Seguidamente se hablará sobre los esfuerzos del proceso de diseño por funcionalidades en cada una de las iteraciones.

3.1. Desarrollo de un modelo global

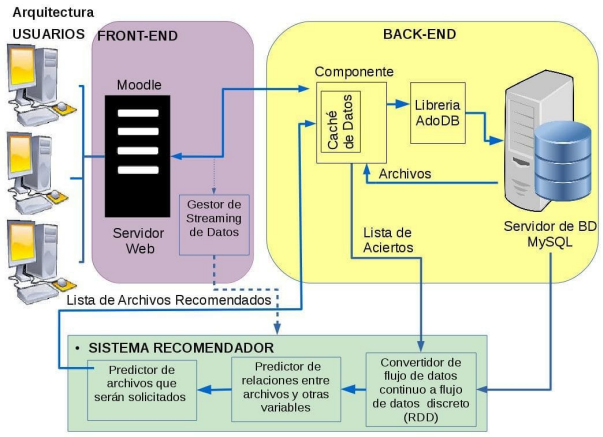


Figura 3.1: Arquitectura global de la solución

La solución al problema planteado consta de la integración conjunta entre un sistema recomendador externo a Moodle y una extensión que maneje las transacciones entre el sistema recomendador, la memoria caché implementada con Redis, las librerías de la plataforma y la interfaz de usuario (figura 3.1).

Para comprender el alcance y la ubicación de nuestra extensión dentro de Moodle y su interacción con los diversos módulos y librerías de este, se partió tomando en cuenta la arquitectura de Moodle que se puede observar en la figura 3.2.

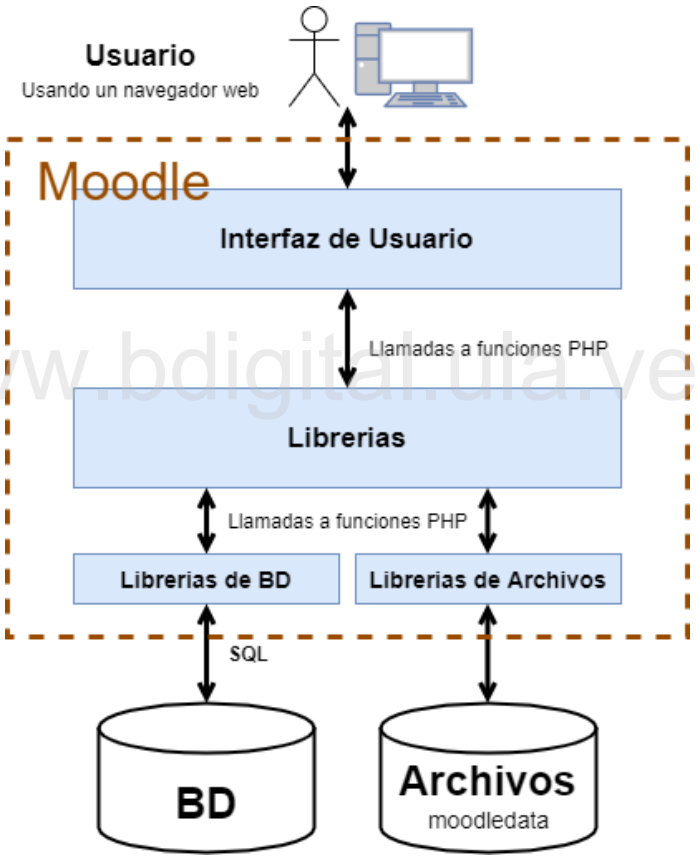


Figura 3.2: Arquitectura de Moodle (modificado de (Tim, 2010))

Nuestro *plugin* se insertó entre la interfaz de usuario de Moodle y el almacén de datos y archivos como se observa en la figura 3.3, funcionando como una capa intermedia o middleware que recibe tanto las consultas de cada usuario como las respuestas desde el servidor.

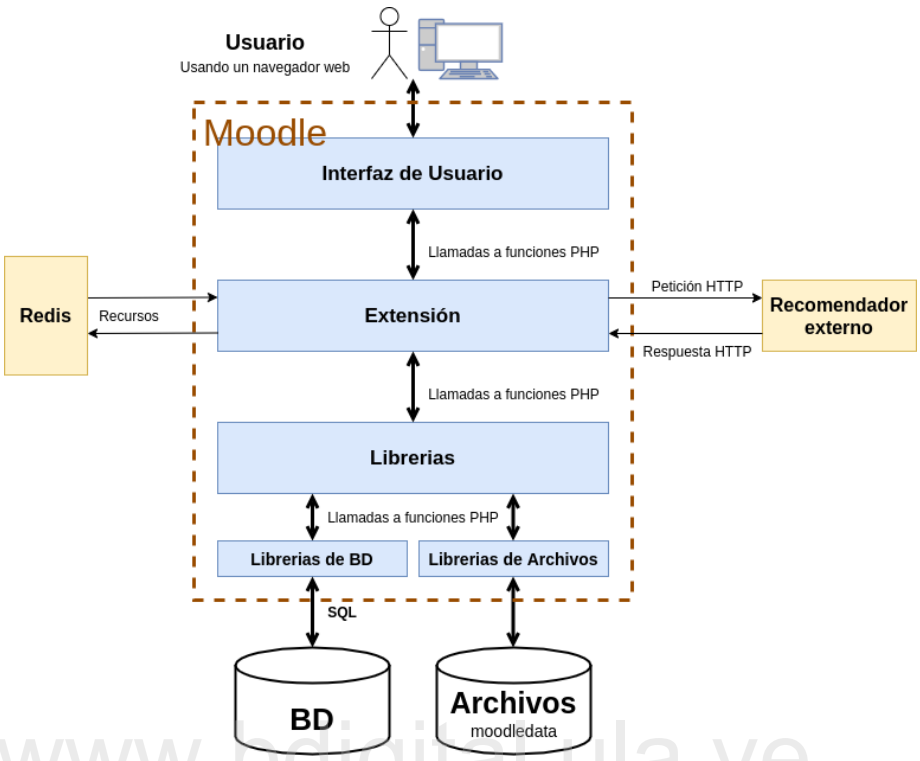


Figura 3.3: Ubicación de la extensión en la arquitectura de Moodle (modificado de (Tim, 2010))

Nuestra extensión consta de una memoria caché implementada usando Redis, ya que el lenguaje de programación PHP posee librerías que facilitan su integracón conjunta. Redis es un motor de base de datos que hace uso de la memoria principal del computador para alojar datos en un esquema clave-valor, esta permite de la misma forma almacenar datos o archivos binarios, permitiendo alojar documentos, imágenes, archivos ejecutables, etc, que no excedan un tamaño de 512 MB.

El diseño de la memoria caché utilizada en este componente consiste en una memoria que se ubicará en el servidor web en el cual se encuentra alojado nuestro cliente de Moodle. Su función es la de mantener copias de los archivos a los cuales los usuarios accedan con mayor frecuencia en la plataforma con el fin de proveer una respuesta más rápida a sus solicitudes.

Adicionalmente, tiene la tarea de solicitar al recomendador externo, cada vez que sea necesario un listado actualizado de los archivos que este determine deban permanecer en la memoria caché y crear una copia de estos en la misma, así como

proveer al recomendador un listado de aciertos y errores que pueda usar para futuras recomendaciones.

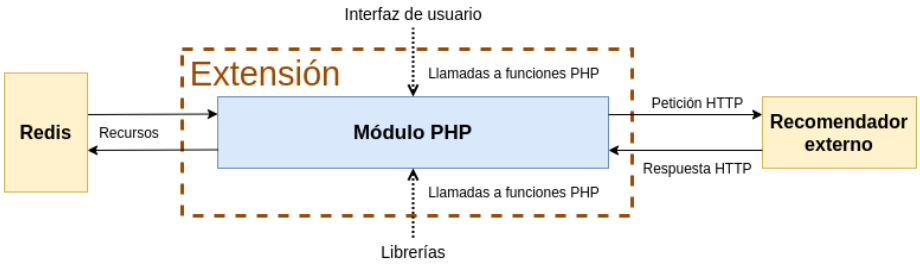


Figura 3.4: Arquitectura de la extensión

El fin de esta memoria caché es almacenar los archivos más concurridos y debe ser actualizada cada vez que se produzca una nueva recomendación. El *plugin* se encarga de remover los archivos que ya no sean necesarios y cubrir este espacio con nuevos recursos.

El componente no solo debe encargarse de dar respuesta a las solicitudes cuando el recurso solicitado se encuentre en la memoria caché, sino que también debe responder cuando este no se encuentre a primera mano, buscandolo en el servidor de archivos y enviandolo al usuario que lo ha solicitado.

3.2. Construcción de una lista de funcionalidades

Según (Ambler, 2002), Una funcionalidad es una pequeña función valorada por el cliente expresada en la forma <acción><resultado><objeto>. Por ejemplo, “Calcular el total de una venta”, “Validar la contraseña de un usuario” y “Autorizar la transacción de venta de un cliente”.

Acorde a lo descrito en la sección anterior en la cual se desarrolló un modelo global de nuestra extensión, la lista de funcionalidades que se estableció es la siguiente:

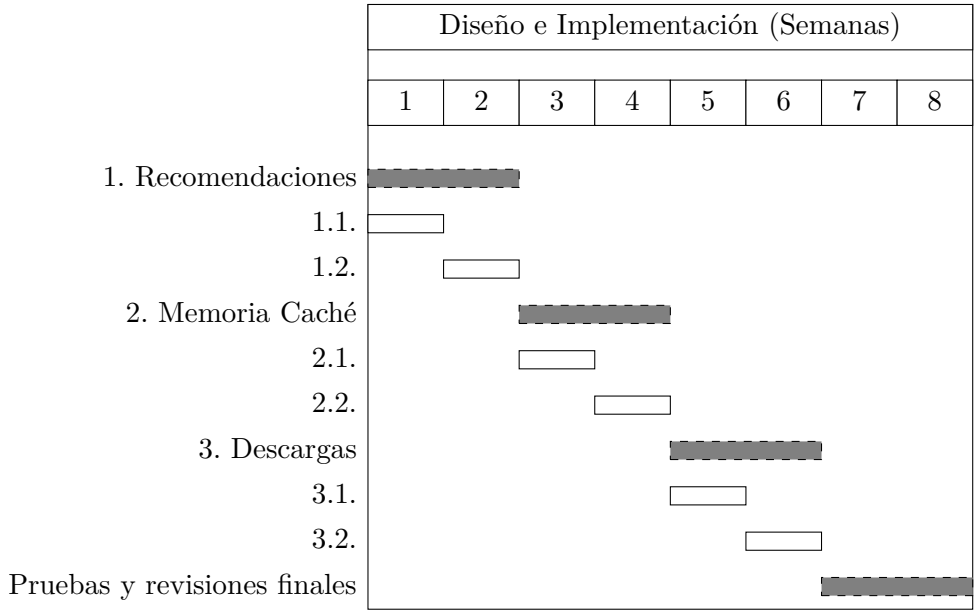
1. Recomendaciones
1.1. Solicitar recomendaciones al recomendador externo (HTTP Request).
1.2. Enviar métricas de aciertos y errores al recomendador externo (HTTP Request).

2. Memoria caché
2.1. Limpiar archivos innecesarios de la memoria caché cada vez que se produzca una nueva recomendación.
2.2. Copiar archivos nuevos a la memoria caché de acuerdo a la última recomendación producida.
3. Descargas
3.1. Descargar archivos solicitados por los usuarios desde la memoria caché en caso de encontrarse en la misma.
3.2. Descargar archivos solicitados por los usuarios desde el almacenamiento en caso de no encontrarse en la memoria caché.

Tabla 3.1: Lista de funcionalidades

3.3. Planificación por funcionalidades

Los esfuerzos de construcción fueron divididos en tres iteraciones de dos semanas, cada una comprendiendo un set de funcionalidades en el orden descrito a continuación:



3.4. Diseño de funcionalidades

El diseño de los sets de funcionalidades se realizó al comienzo de cada iteración en la cual correspondía su desarrollo como se estableció en la sección anterior.

3.4.1. Recomendaciones

La solicitud de recomendaciones y el envío de métricas se deben realizar de forma automática cada vez que sea necesario. Por ende, la extensión hace uso de las tareas sincronizadas de Moodle. Una tarea es una unidad de trabajo que debe ser realizada en un tiempo determinado, son especialmente útil para ejecutar una tarea de mantenimiento en un horario regular.

Figura 3.5: Comunicación entre la extensión y el recomendador

En la figura 3.5 se puede observar la interacción entre nuestra extensión y el recomendador externo. Las recomendaciones y métricas se recibirán y enviarán como solicitudes HTTP en formato JSON.

El diagrama de secuencia mostrado en la figura 3.6 corresponde a la secuencia que sigue el proceso de solicitud de recomendaciones y envío de métricas al momento de ejecutarse la tarea programada que se encarga de comunicarse con el recomendador externo.

3.4.2. Memoria caché

La memoria caché implementada en este proyecto tiene la funcion de almacenar los archivos más concurridos por los usuarios de la plataforma Moodle.

Luego de recibir un listado de recomendaciones desde el recomendador externo mencionado en la sección anterior, nuestro componente debe limpiar de la memoria caché los archivos que no se contemplen en este listado antes de proceder con el copiado.

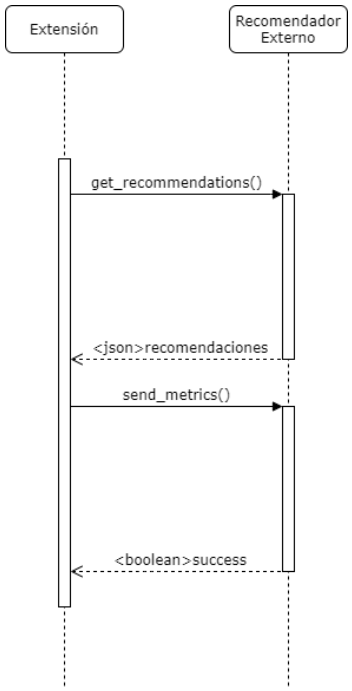


Figura 3.6: Diagrama de secuencia de la interacción extensión-recomendador

Posteriormente el componente busca uno por uno los archivos del listado en el almacenamiento de archivos. Para cada caso se verifica que el recurso no se encuentre almacenado ya en la memoria cache y de no ser así se verifica si el espacio disponible es suficiente para agregarlo a la memoria considerando el límite establecido. Para este proyecto se fijó un límite de aproximadamente 800MB ya que se cuenta con un almacenamiento físico de 2GB. Si el almacenamiento disponible es mayor al tamaño del archivo se procede a copiarlo.

Antes de continuar con el siguiente archivo se debe actualizar la memoria usada. De quedar entradas sin revisar en el listado el componente buscará el siguiente archivo y realizará el proceso. De no poseer espacio suficiente, se procede con el siguiente recurso del listado hasta que este sea cubierto en su totalidad o la memoria esté usada por completo.

En la figura 3.7 se puede ver el proceso explicado anteriormente representado con un diagrama de actividad.

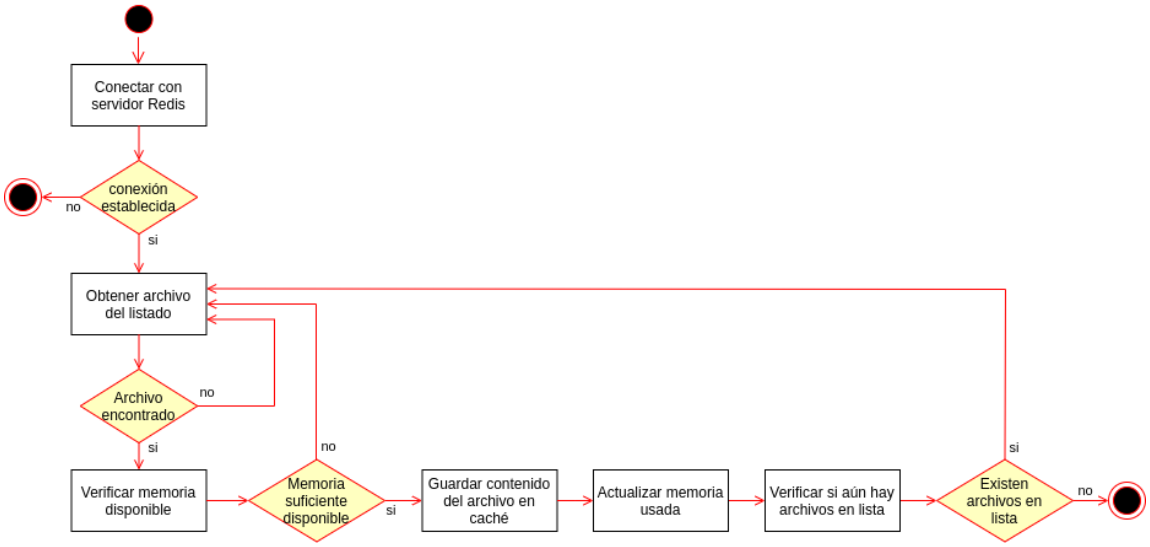


Figura 3.7: Diagrama de actividad: Copiado de un archivo a la memoria caché.

3.4.3. Descargas

Para integrar estas funcionalidades al flujo normal de los procesos de Moodle es necesario adentarnos en su código fuente, especificamente en la rutina encargada de gestionar todas las descargas que los usuarios solicitan.

La mayoría de actividades que involucran la descarga de archivos y reproducción de recursos multimedia hacen uso de la rutina ***pluginfile.php*** que Moodle posee en su código fuente. Esta rutina sirve de interfaz tomando directamente la solicitud de los usuarios para crear el enlace de descarga del recurso solicitado. Por ende, es allí en donde se realizó la inserción de nuestro *plugin*.

Sin embargo, ya que no siempre se servirán archivos desde la memoria caché administrada por nuestra extensión, solo se realizó una bifurcación en la que luego de obtener los argumentos del archivo solicitado se verifica si este existe o no en caché. Si el resultado de esta verificación es satisfactorio se procede a buscar el archivo en la memoria, si por el contrario se determina que el archivo no se encuentra en la misma, el proceso continua de la misma manera que lo haría si nuestro *plugin* no estuviera implementado.

Luego de determinar el método a utilizar se deben actualizar las métricas de aciertos y errores. En caso de que el archivo se encontrara en la memoria caché,

la cantidad de aciertos de las recomendaciones sumarían uno. Por el contrario si el archivo debiera obtenerse desde el almacenamiento, representaría un desacierto y sumaría uno la cuenta de fallos. Estas métricas son escritas en una tabla de la base de datos creada con este fin llamada **download_optimizer_metrics**.

En la figura 3.8 se puede ver el proceso explicado anteriormente representado con un diagrama de actividad.

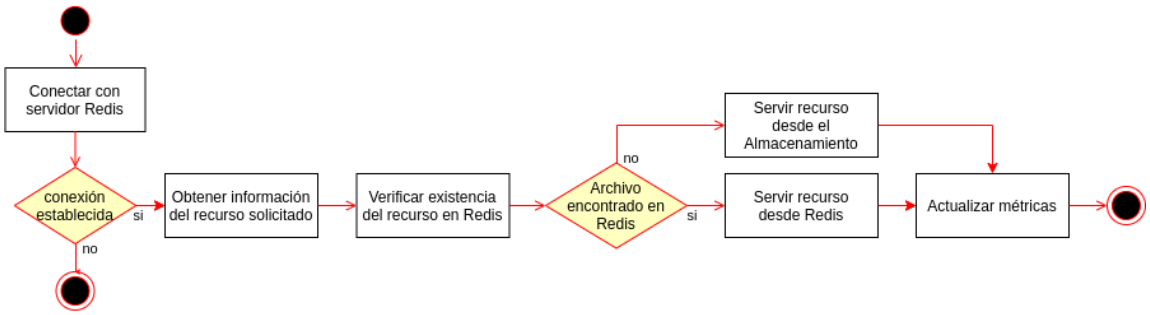


Figura 3.8: Diagrama de actividad: Descarga de archivos.

Capítulo 4

Implementación, Pruebas y Resultados

A continuación se presenta la explicación detallada del proceso de implementación seguido de las pruebas que se realizaron para garantizar que el funcionamiento y rendimiento sea el esperado.

4.1. Implementación de funcionalidades

El último proceso de la metodología trata de la implementación de las funcionalidades y se aplica de forma iterativa en conjunto con el proceso de diseño. Antes de cada iteración se seleccionaron los sets de funcionalidades que serían diseñados e implementados en la misma. Cada set de funcionalidades especificado fue implementado uno por uno como se aprecia a continuación.

4.1.1. Recomendaciones

Se requería que la solicitud de recomendaciones y el envío de de métricas al recomendador externo se hiciera de forma automática en intervalos de tiempo que el administrador de la plataforma decidiera sean adecuados. Para conseguir esto se implementaron estas funcionalidades en una tarea sincronizada de Moodle. Las tareas sincronizadas son ejecutadas a través del proceso Cron de Moodle[1] que es un script

[1]Cron de Moodle: https://docs.moodle.org/all/es/Cron

PHP contenido en el código fuente que debe ejecutarse regularmente en segundo plano y se encarga de ejecutar las tareas sincronizadas en sus intervalos agendados.

El programa ***cron.php*** de Moodle debe ser invocado regularmente, esto con el fin de que las tareas sincronizadas se ejecuten correctamente en los intervalos establecidos. Ya que PHP permite ejecutar programas desde la consola de comandos de linux se configuró el Crontab de Linux para que ejecute cron.php cada minuto.

En el sistema operativo Unix, cron es un administrador regular de procesos en segundo plano que ejecuta procesos o rutinas a intervalos regulares como se especifican en el fichero crontab.

La configuración de la tarea sincronizada creada para ejecutar la solicitud de recomendaciones y el envío de métricas al recomendador externo se encuentra en **db/tasks.php** dentro de la carpeta de la extensión y contiene la especificación del intervalo en el que se debe correr.

El archivo **classes/task/http_requests.php** contiene las funciones que se ejecutaran en esta tarea.

http_requests.php:

```
<?php
//
// This file is part of Moodle - http://moodle.org/
//
// Moodle is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// Moodle is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with Moodle. If not, see <http://www.gnu.org/licenses/>.

/**
 * This file contains the scheduled tasks needed by the plugin.
 *
```

```
 * @package    cachestore_download_optimizer
 * @copyright  2020 Gustavo Mej'ia <bfmvtm@gmail.com>
 * @license    http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
 */

namespace cachestore_download_optimizer\task;
include_once($CFG->dirroot.'/cache/stores/download_optimizer/lib.php');

/**
 * Scheduled task to request new recommendations and send metrics.
 */
class http_requests extends \core\task\scheduled_task {

    /**
     * Return the task's name as shown in admin screens.
     *
     * @return string
     */
    public function get_name() {
        return get_string('httprequests', 'cachestore_download_optimizer');
    }

    /**
     * Execute the task.
     */
    public function execute() {
        check_metrics_availability();

        send_metrics();

        $recommendations = get_recommendations();

        clear_cache($recommendations);
        retrieve_files($recommendations);
    }
}
```


Las actividades a ejecutar se encuentran en el cuerpo de la función **execute**. En primer lugar, se invoca a la rutina **check_metrics_availability** que se encarga de verificar que se encuentren las entradas necesarias en la tabla **download_optimizer_metrics** como se muestran en la figura 4.1 y de no encontrarse

de este modo, se crean. Esto es útil como seguridad para evitar errores que se puedan presentar al momento de escribir los aciertos y errores en la tabla.

id	metric	value
1	fail	0
2	success	0
NULL	NULL	NULL

Figura 4.1: Estructura que debe tener la tabla download_optimizer_metrics.

Posteriormente se invoca la rutina **send_metrics** que envía las métricas capturadas al recomendador. Esta función se encarga de recuperar de la base de datos los valores capturados de aciertos y errores para el último listado de recomendaciones y enviarlos en una petición HTTP de tipo **POST** usando la **biblioteca cURL** de PHP. Una vez finalizado el envío de la petición se reestrablecen los valores de la tabla a cero.

Función **send_metrics:**

```
function send_metrics() {
  global $DB;

  $success = $DB->get_field('download_optimizer_metrics', 'value',
     ['metric' => 'success']);
  $fail = $DB->get_field('download_optimizer_metrics', 'value', ['metric'
     => 'fail']);

  $metrics = array(
    'success' => $success,
    'fail' => $fail
  );

  $payload = json_encode(array('metrics' => $metrics));

  $url = 'https://obscure-lake-39056.herokuapp.com/api/get-metrics';
  $ch = curl_init($url);

  curl_setopt( $ch, CURLOPT_POSTFIELDS, $payload );
```

```
  curl_setopt( $ch, CURLOPT_HTTPHEADER,
      array('Content-Type:application/json'));
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

  $response = json_decode(curl_exec($ch));

  curl_close($ch);

  clean_metrics_values();
}
```

La función **get_recommendations** se encarga de solicitar al recomendador una lista de recomendaciones actualizada. Para conseguir esto también se hace uso de la **biblioteca cURL** pero en este caso se envía una petición de tipo **GET**. Esta rutina retorna un arreglo que contiene los identificadores de los archivos más concurridos en cada una de sus posiciones. En caso de que ocurra un fallo, se retorna un mensaje de error.

Función **get_recommendations:**

```
function get_recommendations() {
  $url =
      'https://obscure-lake-39056.herokuapp.com/api/send-recommendations-aux';
  $ch = curl_init($url);

  curl_setopt($ch, CURLOPT_HTTPGET, 1);
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

  $response = json_decode(curl_exec($ch));
  $recommendations = $response->recommendations;

  curl_close($ch);

  if ($response->success) {
    return $recommendations;
  } else {
    return 'There was an error retrieving recommendations';
  }
}
```

4.1.2. Memoria caché

En este proyecto se implementó una memoria caché utilizando el motor de base de datos en memoria Redis el cual es posible manejar desde nuestro *plugin* a través de la librería de PHP **PhpRedis**[2] que proporciona una API para comunicarse con el almacenamiento clave-valor usando código PHP. Su instalación se detalla en el apéndice B.

La función **clear_cache** se encarga de remover de la memoria caché los archivos que ya no son necesarios, es decir, que no se encuentran en el listado recibido. Esta rutina toma un array conteniendo el nuevo listado de recomendaciones y se conecta con Redis usando la instrucción **connect** de la librería **PhpRedis** que recibe como argumentos la dirección y el puerto del servidor Redis y obtiene todas las claves contenidas en la memoria con la instrucción **keys**, estas claves corresponden a los identificadores de los archivos como se encuentran registrados en la base de datos de Moodle. Al comparar los identificadores contenidos en caché con los que se encuentran en el nuevo listado de recomendaciones se obtiene un arreglo con las claves innecesarias. La instrucion **del** toma como argumento el arreglo de las claves a eliminar y las remueve del servidor. De ser removidos todos los archivos innecesarios, la función retorna el valor booleano **TRUE**, en caso contrario, Muestra un mensaje de error en la consola y retorna el valor booleano **FALSE**.

Función **clear_cache:**

```
function clear_cache($recommendations) {
  $redis = new Redis();
  $redis->connect('127.0.0.1', '6379');

  $allkeys = $redis->keys('*');

  $unnecessarykeys = array_diff($allkeys, $recommendations);

  if ($redis->del($unnecessarykeys) == count($unnecessarykeys)) {
    echo "All keys cleaned.\xA";
    return true;
  }
```

[2]Repositorio de PhpRedis: https://github.com/phpredis/phpredis

```
  echo "There was a problem removing unnecessary files from cache.\xA";

  return false;
}
```

La función **retrieve_files** recibe como argumento el array de recomendaciones recibido desde el recomendador y se encarga de copiar los archivos desde el almacenamiento de Moodle a la memoria caché. Ya que esta función también debe velar que el límite para la caché establecido por el administrador no se exceda se usa la instrucción **info**, en este caso con el argumento **"MEMORY"** para obtener el espacio utilizado actualmente.

Función **retrieve_files:**

```
function retrieve_files($recommendations) {
  global $DB;

  $redis = new Redis();
  $redis->connect('127.0.0.1', '6379');

  $info = $redis->info("MEMORY");

  $usedmemory = $info[used_memory];
  $memorylimit = get_cache_limit();

  var_dump($memorylimit);
  var_dump($usedmemory);

  for ($i=0; $i < count($recommendations); $i++) {
    $id = $recommendations[$i];

    // Retrieve the file from the Files API.
    $fs = get_file_storage();
    $file = $fs->get_file_by_id($id);
    $filesize = $file->get_filesize();

    var_dump($filesize);

    if (($usedmemory+$filesize) > $memorylimit)
      continue;
```

```
    if (!$file) {
      echo "File with id ".$id." not found.\xA"; // The file does not
          exist.
    } else {
      $contents = $file->get_content();

      if (!redis_save_file($id, $contents)) {
        echo "There was a problem saving file ".$id." in Redis.\xA";
      } else {
        $info = $redis->info("MEMORY");
        $usedmemory = $info[used_memory];
        var_dump($usedmemory);
        echo "File saved in Redis successfully.\xA";
      }
    }
  }
}
```

La función **get_cache_limit** obtiene el tamaño de la memoria RAM del computador y retorna el límite de la memoria caché en función a la anterior. Para este proyecto se estimó usar la tercera parte del total de la memoria física del computador (aproximadamente 800MB en este caso) pero este valor puede ser modificado a conveniencia del administrador.

Función **get_cache_limit:**

```
function get_cache_limit() {
  $fh = fopen('/proc/meminfo','r');
  $mem = 0;

  while ($line = fgets($fh)) {
    $pieces = array();
    if (preg_match('/^MemTotal:\s+(\d+)\skB$/', $line, $pieces)) {
      $mem = $pieces[1];
      break;
    }
  }

  fclose($fh);

  return ($mem*0.3333)*1024;
```

```
}
```

La rutina **retrieve_files** itera sobre cada identificador contenido en el arreglo de recomendaciones recibido y para cada caso usa la **API de archivos** (File API[3]) de Moodle, que es la interfaz de programación de aplicaciones que Moodle provee para manejar los archivos en la plataforma, con el fin de obtener los metadatos del archivo en cuestión junto con su contenido. Si no existe espacio disponible suficiente para almacenar este archivo, se procede con el siguiente y así sucesivamente. Si por el contrario el espacio disponible es suficiente se utiliza la función **redis_save_file** que recibe como argumentos el identificador del archivo y el contenido del mismo y lo almacena en la memoria caché usando la instrucción **set** de **PhpRedis**. De ser satisfactorio el proceso, se actualiza la variable que contiene el valor de la memoria utilizada y se procede con el siguiente archivo. Por el contrario, si el proceso falla, se retorna un mensaje de error a la consola y se procede de igual forma con el siguiente archivo.

Función **redis_save_file:**

```
function redis_save_file($id, $file) {
  $redis = new Redis();

  if ($redis->connect('127.0.0.1', '6379')) {

    if ($redis->exists($id)) {
      echo "File ".$id." is already cached.\xA";
      return false;
    } else {

      if (!$redis->set($id, $file)) {
        return false;
      }

      return true;
    }
  } else {
    echo "Can't connect with Redis.\xA";
```

[3]Documentación del File API: https://docs.moodle.org/dev/File_API

```
    return false;
  }
}
```

4.1.3. Descargas

Para implementar el uso del componente en el flujo normal de descargas de Moodle se requirió realizar una modificación al código fuente de Moodle. La rutina ***pluginfile.php*** mencionada en el capítulo anterior es la encargada de gestionar las descargas de todos los módulos de la plataforma.

Los cambios realizados al archivo comienzan con la inclusión de la librería de nuestro componente. De ser encontrada se procede a revisar si la tabla encargada de almacenar las métricas se encuentra disponible. El método tradicional de las descargas directas se insertó en la función **serve_file_from_storage**. De esta forma, se mantendría a disposición en caso de ser requerido.

Luego de establecer una conexión con **Redis** se toman los argumentos del archivo solicitado y se hace uso de la **API de archivos** de Moodle para extraer la información necesaria para mapear el archivo en la memoria caché.

Se consiguió un error con las imagenes de visualización de los cursos. Debido a que estos archivos no son encontrados por la **API de archivos**, el flujo normal ocasionaba que estas imagenes no fueran servidas correctamente. Por ende se agregó una condicional en la que de no encontrarse el archivo requerido a ***pluginfile.php*** automáticamente se desviara el flujo a la función **serve_file_from_storage**.

De encontrarse el archivo se verifica su existencia en la base de datos de **Redis**. Si este se encuentra en la misma, se procede a invocar la función **serve_file_from_cache** que toma como argumentos el identificador del archivo en la base de datos, su nombre y su tamaño para extraerlo desde la memoria caché y servirlo al usuario. En caso de no existir el archivo en la memoria se procede a servirse de forma tradicional.

Una vez completado el proceso se actualizan las métricas en la base de datos. Como se detalló en la planificación, si el archivo fue servido desde la memoria caché representaría un acierto que se suma a la cantidad de aciertos registrada en la tabla. En caso contrario, si el archivo debió ser servido desde el almacenamiento, se registra

un fallo en la recomendación agregandose en la tabla.

Para finalizar el flujo se escribe en la base de datos, especificamente en la tabla **download_optimizer_logs** los logs generados en el proceso, como el tiempo tomado para servir el archivo en su totalidad. Cabe resaltar que este tiempo no es el tiempo de descarga, el cual se ve afectado por otras condiciones.

Función **serve_file_from_cache:**

```
function serve_file_from_cache($id, $filename, $filesize){
  $redis = new Redis();
  $redis->connect('127.0.0.1', '6379');

  $content = $redis->get($id);

  header('Content-Disposition: attachment; filename='.$filename);
  header('Content-Type: application/force-download');
  header('Content-Length: ' . $filesize);
  header('Connection: close');

  echo $content;
}
```


Rutina **pluginfile.php:**

```
<?php

// This file is part of Moodle - http://moodle.org/
//
// Moodle is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// Moodle is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with Moodle. If not, see <http://www.gnu.org/licenses/>.
```

```
/**
 * This script delegates file serving to individual plugins
 *
 * @package   core
 * @subpackage file
 * @copyright 2008 Petr Skoda (http://skodak.org)
 * @license   http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later
 */

// Disable moodle specific debug messages and any errors in output.
define('NO_DEBUG_DISPLAY', true);

require_once('config.php');
require_once('lib/filelib.php');
require_once($CFG->dirroot.'/cache/stores/download_optimizer/lib.php');

if (function_exists('serve_file_from_cache')) {
  check_metrics_availability();

  $table = 'download_optimizer_metrics';

  $redis = new Redis();
    $redis->connect('127.0.0.1', '6379');

    $args = explode('/', ltrim(get_file_argument(), '/'));

  $fs = get_file_storage();
    $file = $fs->get_file($args[0], $args[1], $args[2], $args[3], '/',
        $args[4]);

    if (!$file) {
      serve_file_from_storage();
      return;
    }

    $id = $file->get_id();
    $filename = $file->get_filename();
    $filesize = $file->get_filesize();

  if ($redis->exists($id)){
    serve_file_from_cache($id, $filename, $filesize);

    $success = $DB->get_field($table, 'value', ['metric' => 'success']);
```

```
    $DB->set_field($table, 'value', ++$success, ['metric' => 'success']);
  }
  else{
    $fail = $DB->get_field($table, 'value', ['metric' => 'fail']);
    $DB->set_field($table, 'value', ++$fail, ['metric' => 'fail']);

    serve_file_from_storage();
  }

} else {
  serve_file_from_storage();

}

function serve_file_from_storage() {
  $relativepath = get_file_argument();
  $forcedownload = optional_param('forcedownload', 0, PARAM_BOOL);
  $preview = optional_param('preview', null, PARAM_ALPHANUM);
  // Offline means download the file from the repository and serve it,
     even if it was an external link.
  // The repository may have to export the file to an offline format.
  $offline = optional_param('offline', 0, PARAM_BOOL);

  file_pluginfile($relativepath, $forcedownload, $preview, $offline);
}
```


4.2. Pruebas

4.2.1. Uso de las instrucciónes var_dump() y echo para verificar el estado del proceso

La instrucción **var_dump()** de PHP muestra información estructurada sobre una o más expresiones incluyendo su tipo y valor, esta se implementó como herramienta para conocer información de utilidad en la ejecución del proceso de limpiado y llenado de la memoria caché, como por ejemplo el espacio disponible al almacenar o remover un elemento de la misma.

Por otro lado, **echo** se define como un constructor del lenguaje, su función es imprimir un texto dado. Su implementación se aplicó para imprimir mensajes que

explicaran de forma rápida y clara lo que estaba ocurriendo en dicho momento, un ejemplo serían los mensajes de éxito y/o error en algún paso de la ejecución.

4.2.2. Pruebas de rendimiento

Con el fin de conocer el rendimiento de nuestra extensión y realizar las respectivas comparaciones con el flujo normal que provee la plataforma por defecto, se implementó un sistema de logs que son plasmados en la base de datos en una tabla que lleva por nombre **download_optimizer_logs**. La información capturada en esta corresponde al tiempo en microsegundos que se toma la plataforma en servir un archivo especificado con su propio identificador, ya sea que fuese servido desde la memoria caché o directamente desde el almacenamiento.

4.3. Resultados

A continuación se muestran los resultados obtenidos tabulados para ambos casos. Como se puede apreciar, se utilizaron archivos PDF, FLAC y MP4 de tamaños variados con el fin de probar la mayor cantidad posible de comportamientos de la plataforma. Estas pruebas de rendimiento fueron ejecutadas en un equipo que contaba con un procesador Intel Core 2 Duo, un disco duro mecánico de 256GB de almacenamiento y una memoria de acceso aleatorio DDR de 2GB.

4.3.1. Resultados para archivos PDF

- **Archivo 1:** Archivo .pdf de 12.74 MB.
- **Archivo 2:** Archivo .pdf de 126.12 MB.
- **Archivo 3:** Archivo .pdf de 395.89 MB.

Archivo 1		Archivo 2		Archivo 3	
Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)
0.0013147	0.0712364	0.0391349	0.0450909	0.1339281	0.7795469
0.0006478	0.0071478	0.0508111	0.0482569	0.1019279	0.0440559

0.0008741	0.0004878	0.0425971	0.0337379	0.1095259	0.0783770
0.0014752	0.0016478	0.0404260	0.0339191	0.1336439	0.0623741
0.0004318	0.0013712	0.0525000	0.0977330	0.1212389	0.0561211
0.0001687	0.0004048	0.0446059	0.0707841	0.1059069	0.0951399
0.0013548	0.0009379	0.0130169	0.0695179	0.1041231	0.0578520
0.0017493	0.0017519	0.0858491	0.0737338	0.0945940	0.1038241
0.0001975	0.0009794	0.0460801	0.0814800	0.1542399	0.0923815
0.0001647	0.0002159	0.0125771	0.0513611	0.1144018	0.0686631

Tabla 4.1: Tabla de tiempos de ejecución registrados para archivos PDF.

4.3.2. Resultados para archivos FLAC

- **Archivo 1:** Archivo .flac de 23.20 MB.
- **Archivo 2:** Archivo .flac de 125.92 MB.
- **Archivo 3:** Archivo .flac de 329.74 MB.

Archivo 1		Archivo 2		Archivo 3	
Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)
0.0037560	0.1219440	0.6077996	0.5212248	0.1339281	0.5204261
0.0089550	0.0028629	0.5650714	0.6311146	0.1019279	0.6689912
0.0036220	0.0064800	0.4697466	0.4330288	0.1095259	0.8216092
0.0030441	0.0026130	0.5227754	0.3726597	0.1336439	0.6835562
0.0046251	0.0021999	0.4451844	0.5112276	0.1212389	0.6589940
0.0028789	0.0038180	0.4551816	0.6686376	0.1059069	0.7788810
0.0037561	0.0027082	0.3066165	0.5357898	0.1041231	0.8164040
0.0051291	0.0027139	0.3669856	0.5888186	0.0945940	0.5807952
0.0044241	0.0085749	0.6025944	0.6738428	0.1542399	0.7365850

0.0039530	0.0047879	0.5345924	0.6006356	0.1144018	0.7484020

Tabla 4.2: Tabla de tiempos de ejecución registrados para archivos FLAC.

4.3.3. Resultados para archivos MP4

- **Archivo 1:** Archivo .mp4 de 23.20 MB.
- **Archivo 2:** Archivo .mp4 de 125.92 MB.
- **Archivo 3:** Archivo .mp4 de 329.74 MB.

Archivo 1		Archivo 2		Archivo 3	
Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)
0.0004151	0.0434270	0.5390871	0.8578963	0.7674171	0.8148802
0.0010800	0.0008411	0.4963589	0.5342097	0.8146889	0.6295554
0.0006000	0.0008581	0.4410341	0.7198433	0.6293641	0.5916842
0.0004342	0.0009189	0.4940629	0.7728721	0.5914929	0.7676084
0.0004079	0.0013142	0.5138969	0.6933227	0.6948019	0.7844012
0.0004609	0.0010289	0.3864691	0.4253199	0.7047991	0.7049904
0.0004429	0.0009379	0.3753290	0.8367132	0.5562340	0.5564253
0.0006721	0.0254129	0.2982731	0.6041239	0.6166031	0.6167944
0.0010161	0.0009000	0.5338819	0.6526911	0.7422029	0.7423942
0.0006471	0.0009441	0.4658799	0.6047307	0.7842099	0.6949932

Tabla 4.3: Tabla de tiempos de ejecución registrados para archivos MP4.

En la siguiente tabla podemos ver una comparación entre las medias de los tiempos de respuesta del servidor capturados en ambos casos.

Archivo 1		Archivo 2		Archivo 3	
Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)
0.0008379	0.0086181	0.0427598	0.0605615	0.1173530	0.1438336

Tabla 4.4: Tabla de medias de tiempos de ejecución registrados para archivos PDF.

Archivo 1		Archivo 2		Archivo 3	
Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)
0.0044143	0.0158703	0.4876548	0.5536980	0.6993443	0.7014644

Tabla 4.5: Tabla de medias de tiempos de ejecución registrados para archivos FLAC.

Archivo 1		Archivo 2		Archivo 3	
Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)	Cache(ms)	Servidor(ms)
0.0006176	0.0076583	0.4544273	0.6701723	0.6901814	0.6903727

Tabla 4.6: Tabla de medias de tiempos de ejecución registrados para archivos MP4.

En general podemos notar que en las pruebas para los distintos tipos de archivo se percibió una mejora. Se pudo apreciar que el rendimiento en los archivos más pequeños fue mejor que en los archivos con mayor tamaño, alcanzando mejoras de 91.28 % en archivos PDF, 72.19 % en archivos de audio FLAC y 91.94 % en archivos de video MP4. También podemos notar que el rendimiento en general de la plataforma para servir archivos de audio y video reduce con mayor proporción en

comparación con archivos de texto PDF a medida que aumenta el tamaño de los mismos.

Capítulo 5

Conclusiones y Recomendaciones

En este proyecto se logró diseñar e implementar una extensión en la plataforma Moodle con el fin de mejorar los tiempos de respuesta de la plataforma al momento de servir recursos a los usuarios.

Para llevar esto a cabo, se aplicó una metodología de desarrollo basada en funcionalidades, a partir de la cual se pudo organizar el trabajo en 5 procesos: Desarrollar un modelo global, construir una lista de características, planificar, diseñar e implementar.

Luego de desarrollar un modelo global óptimo, se construyó una lista de características enfocada en satisfacer las necesidades planteadas en el proceso anterior con el fin de cumplir con los objetivos planteados al inicio del proyecto. Posteriormente se realizó la planificación y el diseño de cada funcionalidad por separada pero manteniendo la integridad del sistema para finalmente realizar la implementación.

Con la extensión implementada, se realizaron las pruebas de funcionamiento y rendimiento correspondientes, en las cuales se determinó que el trabajo realizado cumplía con los objetivos planteados resultando en una implementación completamente funcional del modelo diseñado como primer paso en este proyecto.

En base a los resultados obtenidos, se puede comprobar que efectivamente la extensión aquí implementada mejorará la experiencia del usuario dentro de la plataforma al momento de acceder a algún recurso de alta demanda optimizando los tiempos de respuesta de la misma dependiendo de las propiedades los archivos.

Como recomendaciones encontradas durante este estudio, hemos planteado las siguientes:

- Desplegar el cliente de Moodle que hará uso de este *plugin* en un sistema operativo basado en Linux.
- Configurar el crontab del sistema operativo para que ejecute el proceso **cron** de Moodle en intervalos de menos de un (1) minuto. Para garantizar que se mantenga un listado de recomendaciones actualizado.
- Configurar un límite seguro desde la consola de Redis además del límite establecido desde la extensión con el fin de evitar eventuales desbordamientos.
- Modificar el límite de memoria principal utilizada por la extensión, por defecto la extensión utilizará una tercera parte (1/3) de la memoria que se posee en el servidor.
- Integrar este esquema con el sistema recomendador para optimizar la gestión de almacenamiento de Moodle.

Apéndice A: Instalación de un *Plugin*

Instalación directa desde el directorio de *plugins* de Moodle

1. Ingrese a su sitio como administrador y vaya a Administración >Administración del sitio >*Plugins* >Instalar *plugins*. (Si Usted no puede encontrar este lugar, esto es debido a que en su sitio está prohibido instalar *plugins*).

2. Elija el botón que dice Instalar *plugins* desde el directorio de *plugins* de Moodle.

3. Busque un *plugin* que tenga un botón para instalar (Install) que asegura que es compatible con su versión de Moodle), elija el botón para instalar (Install) y luego elija continuar (Continue).

4. Revise que aparezca el mensaje de que pasó la validación (Validation passed!) y después elija el botón para instalar el *plugin* (Install add-on).

Instalación mediante archivo ZIP subido al sitio

1. Vaya al Moodle *plugins* directory[1], seleccione su versión actual de Moodle (2.5/2.6/3.0/...), después elija un *plugin* que tenga un botón para Descargar (Download) y descargue el archivo ZIP.

2. Ingrese a su sitio Moodle como administrador y vaya a Administración >Administración del sitio >*Plugins* >Instalar *plugins*.

[1]Directorio de *plugins*: https://moodle.org/plugins

3. Suba el archivo ZIP, seleccione el tipo apropiado de *plugin*, acepte la casilla de aceptación, después elija el botón para Ínstalar un *plugin* desde un archivo ZIP'.

4. Revise que aparezca el mensaje de que pasó la validación (Validation passed!) y después elija el botón para Instalar el *plugin* (Install add-on).

Instalación manual en el servidor

En primer lugar, establezca el sitio correcto dentro del árbol de directorios de Moodle en donde debe de ir el tipo de *plugin*. Las localizaciones comunes son:

- /ruta/a/moodle/theme/ - temas gráficos
- /ruta/a/moodle/mod/ - recursos y módulos de actividad
- /ruta/a/moodle/blocks/ - bloques que van a un lado
- /ruta/a/moodle/question/type/ - tipos de preguntas
- /ruta/a/moodle/course/format/ - formatos de curso
- /ruta/a/moodle/admin/report/ - reportes administrativos

1. Vaya al Moodle *plugins* directory[2], seleccione su versión actual de Moodle (2.5/2.6/3.0/...), después elija un *plugin* que tenga un botón para Descargar (Download) y descargue el archivo ZIP.

2. Súbalo o cópielo a su servidor Moodle.

3. Descomprima (unzip) el archivo al lugar apropiado para el tipo de *plugin* (o siga las instrucciones del *plugin*).

4. En su sitio Moodle (como administrador) vaya a Configuraciones >Administración del sitio >Notificaciones (para la mayoría de los *plugins*, Usted debería de ver un mensaje que le diga que el *plugin* está instalado).

[2]Directorio de *plugins*: https://moodle.org/plugins

Apéndice B: Instalación Redis en Linux

Instalación usando la línea de comandos

Actualice el cache de su apt e instale Redis escribiendo en la consola:

```
$ sudo apt update
$ sudo apt install redis-server
```

De esta forma se descargará e instalará Redis y sus dependencias. Seguido de esto, hay un cambio importante en las configuraciones que se debe realizar en el archivo de configuración de Redis, el cual fue generado automaticamente durante la instalación.
Abra este archivo con el editor de texto de su preferencia:

```
$ sudo nano /etc/redis/redis.conf
```

Dentro del archivo, busque la instrucción **supervised**. Esta instrucción le permite declarar un sistema de arranque para manejar Redis como un servicio, proporcionando al administrador un mayor control sobre sus operaciones. La instrucción está establecida en **no** por defecto. En sistemas basados en Debian, cambie esta configuración a **systemd**.

Archivo **/etc/redis/redis.conf**

```
. . .

# If you run Redis from upstart or systemd, Redis can interact with your
# supervision tree. Options:
#   supervised no      - no supervision interaction
```

```
#   supervised upstart - signal upstart by putting Redis into SIGSTOP mode
#   supervised systemd - signal systemd by writing READY=1 to
    $NOTIFY_SOCKET
#   supervised auto  - detect upstart or systemd method based on
#                      UPSTART_JOB or NOTIFY_SOCKET environment variables
# Note: these supervision methods only signal "process is ready."
#       They do not enable continuous liveness pings back to your
    supervisor.
supervised systemd

. . .
```

Guarde y cierre el archivo y recargue el archivo de servicio de Redis para reflejar los cambios realizados en el archivo de configuración.

Instalación de PhpRedis

El método recomendado para instalar PhpRedis es usando **pecl**.

Instalar **pecl**:

```
$ sudo apt install pkg-php-tools
```

Instalar **PhpRedis**:

```
$ sudo pecl install redis
```

Referencias

Scott W. Ambler. Feature driven development (fdd) and agile modeling. 2002. URL http://agilemodeling.com/essays/fdd.htm.

R. Bello Díaz. Educación virtual: Aulas sin paredes. *Ciudades Virtuales Latinas*, 2005.

Mark Newnham Damien Regad y The ADOdb Community. Adodb - database abstraction layer for php. 2014. URL https://adodb.org/dokuwiki/doku.php.

Francisco Hidrobo Gabriel Barrios, Yaneth Moreno. Entendiendo el funcionamiento de moodle: un enfoque basado en un marco de modelado. *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação*, (20):327–337, 2019.

Guillermo García. Design and implementation of a mobile application based on cordova framework and web technologies for a university intranet. *Telecomunicación*, 2017.

Sam Hemelryk. Alternative php cache (apc). 2014. URL https://moodle.org/plugins/cachestore_apc.

Raúl Ivorra Oltra y Sergio Luján-Mora. Ampliación de moodle: Creación de módulo actividad. 2009.

Eric Merrill. Memcache cluster. 2014. URL https://moodle.org/plugins/cachestore_memcachecluster.

Mohamed A Mgheder y Mick J Ridley. Automatic generation of web user interfaces in php using database metadata. En IEEE, ed., *Internet and Web Applications*

and Services, 2008. ICIW'08. Third International Conference on, págs. 426–430. 2008.

Moodle. Acerca de moodle. 2019. URL https://docs.moodle.org/all/es/Acerca_de_Moodle.

Daniel Sanchez, Oscar Mendez, y Hector Florez. Applying the 3-layer model in the construction of a framework to create web applications. En IIIS, ed., *The 8th International Multi-Conference on Complexity, Informatics and Cybernetics, 2017.*, pág. 364"369. 2017.

Hunt Tim. A basic introduction to the moodle architecture. 2010. URL https://www.slideshare.net/tjh1000/a-basic-introduciton-to-the-moodle-architecture-5442122.