UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

UBULog 1.1

Presentado por Oscar Fernández Armengol
en Universidad de Burgos — 8 de enero de 2018
Tutor: Raúl Marticorena Sánchez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Oscar Fernández Armengol, con DNI 71346020-C, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de enero de 2018

Vº. Bº. del Tutor:

D. nombre tutor

Resumen

Este proyecto trata de crear una aplicación de escritorio, para docentes de la universidad de brugos, aunque se puede extender a cualquiera que tuviera un entorno de Moodle por lo menos de la versión 3.3.

La aplicación le permitirá loguearse en la plataforma de Moodle, utilizando el correo web, su contraseña y el dominio, en nuestro caso es https://ubuvirtual.ubu.es. Una vez iniciada la sesión sabremos las asignaturas en las que está matriculado como profesor o alumno, una vez elegida la asignatura sabremos quien está matriculado en ella. Una vez aquí podremos ver las interacciones de los log de los alumnos de manera sencilla.

Nuestra aplicación consulta y guarda información de los cursos y los usuarios, a través del web service con una respuesta JSON.

A lo largo de la memoria se explicará todos los pasos del desarrollo con los problemas ocasionados.

Este proyecto es una continuación del proyecto UBUGrades 1.0 pero en vez de sacar estadísticas de las notas, se sacará las interacciones de los diferentes usuarios para mostrarlo de forma amigable a los usuarios.

Descriptores

Moodle, registros, servicios web, JSON, JavaFX, e-learning, análisis del aprendizaje, web scriping.

Abstract

This project tries to create a desktop application for teachers of the University of Burgos, although it can be extended to anyone who has a Moodle environment at least of version 3.3.

The application will allow you to login to the Moodle platform, using webmail, your password and the domain, in our case it is https://ubuvirtual.ubu.es. Once the session begins, we will know the subjects in which he / she is enrolled as a teacher or student, once the subject is chosen, we will know who is enrolled in it. Once here we can see the interactions of the students' log in a simple way.

Our application consults and saves information about courses and users, through the web service with a JSON response.

Throughout the memory all the steps of the development will be explained with the problems caused.

This project is a continuation of the UBUGrades 1.0 project but instead of taking statistics from the notes, the interactions of the different users will be taken out in order to show it in a friendly way to the users.

Keywords

Moodle, logs, web services, JSON, JavaFX, e-learning, analysis of learning, web scripting.

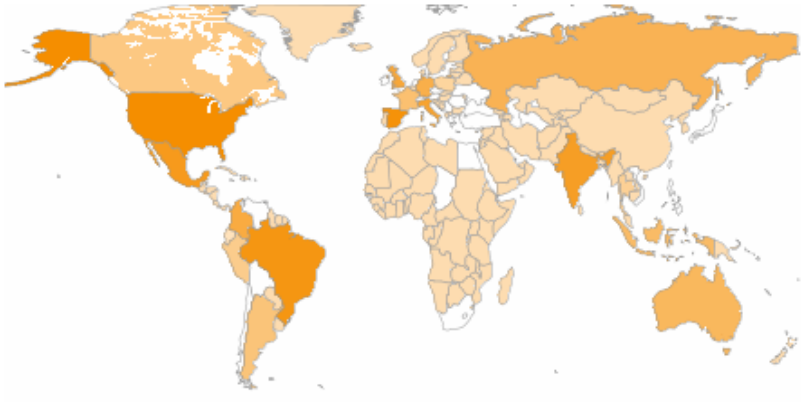
Índice general

Índice de figuras

Índice de tablas

Introducción

Los profesores, hoy en día, utilizan plataformas para poder facilitar su trabajo, proporcionando recursos, haciendo exámenes o teses, de manera on-line. No es que se quede anticuada la forma tradicional de impartir clases, si no que evoluciona, para facilitar el trabajo a ellos mismos y a los alumnos. La más utilizada y a la vez la que estamos utilizando en la Universidad de Burgos es Moodle, en concreto la versión 3.3.



Country	Registrations
Estados Unidos	10,811
España	8,145
Brasil	5,358
México	5,254
Italia	3,965
India	3,899
Reino Unido	3,705
Colombia	3,087
Alemania	2,865
Federación Rusa	2,529

Figura 1.1: Estadística de ciudades que más utilizan Moodle.

Registered sites	93,290
Countries	234
Courses	14,315,902
Users	121,930,657
Enrolments	494,331,813
Forum posts	249,257,628
Resources	128,597,657
Quiz questions	706,158,526

Figura 1.2: Datos de uso de Moodle.

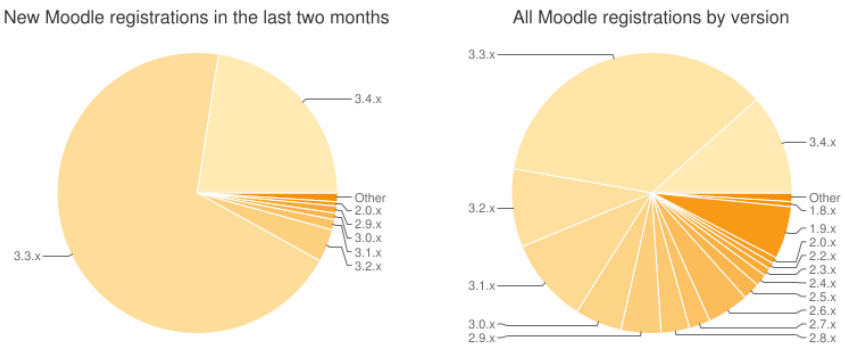


Figura 1.3: Versiones más utilizadas.

Lo que queremos conseguir aprovechando el potencial de Moodle, es, una aplicación de escritorio, en la que podremos pasar el log correspondiente, que nos lo generan los registros de Moodle en cada asignatura y el profesor puede acceder a ellos sin ningún problema, para facilitarle su interpretación, con un muestreo de los datos en formato de gráficas, filtrando por usuarios de esa asignatura (contando con el usuario administrador, el usuario sistema, usuario invitado y desconocido, como no sabemos los usuarios exactos que hay en los Moodle de producción, los colocaremos en desconocidos para controlar su interacción en la asignatura) y los tipos de evento realizado. Con esto podremos permitir al profesor un control de su asignatura, casi completo. Por ejemplo, podrá saber cuántas veces un alumno ha entrado en calificaciones, ha subido un trabajo, hasta si un administrador ha estado visitando su asignatura o creado nuevos elementos en ella.

1.1. Estructura de la memoria

La memoria sigue la siguiente estructura:

- **Introducción:** breve descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que persigue el proyecto.
- **Conceptos teóricos:** breve explicación de los conceptos teóricos clave para la comprensión de la solución propuesta.
- **Técnicas y herramientas:** listado de técnicas metodológicas y herramientas utilizadas para gestión y desarrollo del proyecto.

- **Aspectos relevantes del desarrollo:** exposición de aspectos destacables que tuvieron lugar durante la realización del proyecto.
- **Conclusiones y líneas de trabajo futuras:** conclusiones obtenidas tras la realización del proyecto y posibilidades de mejora o expansión de la solución aportada.

Junto a la memoria se proporcionan los siguientes anexos:

- **Plan de proyecto:** análisis de la viabilidad del proyecto y su planificación.
- **Requisitos:** exposición de los requisitos que debería de tener nuestra aplicación.
- **Diseño:** exposición del diseño de la aplicación.
- **Manual del programador:** aspectos relevantes del código fuente.
- **Manual del usuario:** guía detallada del manejo de la aplicación.

1.2. Materiales adjuntos

Los materiales que se adjuntan con la memoria son:

- Aplicación UBULog 1.1

Además, los siguientes recursos están accesibles a través de internet:

- Código público en el repositorio de GitHub.
- Análisis de la calidad del código con SonarQube.
- Integración continua del proyecto con Travis CI.

Objetivos del proyecto

A continuación, se detallan los diferentes objetivos que han motivado la realización del proyecto.

2.1. Objetivos generales

- Desarrollar una aplicación de escritorio para poder visualizar las interacciones realizadas en el Moodle de la Universidad de Burgos, por los diferentes usuarios, en las distintas asignaturas.
- Facilitar la interpretación de los datos recogidos mediante representaciones gráficas.
- Aportar información extra a los profesores de la asignatura, para saber la interacción que tienen sus alumnos con ella y otros usuarios como puede ser el administrador de Moodle.

2.2. Objetivos técnicos

- Desarrollar aplicación de escritorio con Java FX.
- Desarrollar lógica de programación en entorno Java 8
- Utilización de API WebService de Moodle para la obtención de diferentes datos.
- Parseo de documentos CSV.
- Utilizar Git como sistema de control de versiones distribuido junto con la plataforma GitHub.
- Aplicar la metodología ágil Scrum en el desarrollo del software.
- Utilizar ZenHub como herramienta de gestión de proyectos.
- Utilizar Mendeley para almacenamiento de bibliografía.

2.3. Objetivos personales

- Utilizar librería chart.js para generar los gráficos.
- Utilizar Gradle como herramienta para automatizar el proceso de construcción de software.
- Hacer uso de herramientas de integración continua como Travis y SonarQube en el repositorio.
- Aumentar los tipos de parseo de documentos.

Conceptos teóricos

Vamos a hablar de los conceptos teóricos aportando nueva información contando la información del tfg anterior de Claudia Martínez Herrero (página 6-13, año 2017) [17].

3.1. Token

Token para la autenticación en web, que es como lo estamos utilizando nosotros. Es un código único, en algunos casos temporal, que se le asocia a un usuario para mantenerle su sesión abierta, mientras ese token sea correcto. En nuestro caso, se genera un token permanente, dado que estamos en pruebas. el token que nos genera Moodle es: "9a5e85d1e61c1c42509d77b34f26643a"

3.2. Json

JavaScript Object Notation (*Json*) es una estructura en formato texto, con el cual, podemos enviar y recibir datos de un servidor, de forma rápida y sencilla.

Su estructura seria tal que así:

Figura 3.4: Estructura Json.

La imagen representa una respuesta del servidor de Moodle haciéndole una petición a su WebService.

Para su tratamiento de los datos es similar a un array de objetos, nos podemos mover por él y almacenar esos datos para nuestro uso futuro, igualmente podemos crear un Json y enviarlo al servidor para que lo trate el, en el caso de que estuviera permitido.

3.3. Web Scraping

Web Scraping es una manera de interaccionar con una web y automatizar las interacciones. Su uso habitual es para el testeo de webs, probar campos, botones, etc.... Pero también podemos utilizarlo para el tratamiento de datos. Hay veces que la web no proporciona una api rest o un archivo con el cual podamos tratar ciertos datos para hacer alguna cosa concreta. En nuestro caso, lo tenemos que utilizar para la descarga automática de un log, en el que si no estamos logueados, no podríamos descargar ningún documento.

El usuario podría ir al registro sin ningún problema y descargarlo, pero también nos interesa, que si el usuario no sabe dónde está el registro o le resulta dificultoso el proceso de descargarlo manualmente, podamos automatizarlo para él.

Lo que hemos hecho es automatizar el proceso que haría el usuario de forma manual.

1. Nos iremos a la URL de logueo.
2. Rellenamos usuario y contraseña e inmediatamente hacemos clic en aceptar.
3. Construimos la URL para ir al registro con los filtros para que salgan todos, se puede poner en la URL porque Moodle, esa búsqueda, hace una petición GET.
4. Una vez estemos en la ventana con todos los log del registro hacemos clic en descargar, como por defecto el archivo descarga es CSV no consideramos ese movimiento.
5. Recogemos la respuesta que nos da la web y la convertimos en String, en este paso ya tenemos todo el registro almacenado en nuestra aplicación.

Aunque en nuestra aplicación ya nos logueamos al principio, para este proceso hay que volver a loguearse, ya que, es token que nos proporciona el web service de Moodle, no es el mismo.

Es importante destacar, que este proceso, para el usuario, es transparente. Lo único que va a notar, es que, le cuesta cargar el log más. En el equipo que se ha desarrollado el proyecto un log de 100 registros es imperceptible, pero hacer que se lo descargue automáticamente incrementa el tiempo de forma considerable.

Técnicas y herramientas

4.1. Metodologías

Scrum

Scrum es un marco de trabajo para el desarrollo de *software* que se engloba dentro de las metodologías ágiles. En él se define un conjunto de prácticas y roles durante el desarrollo del proyecto. Es una estrategia de trabajo iterativa e incremental a través de iteraciones (*sprints*) y revisiones.

Durante el proyecto se han llevado a cabo reuniones semanales con el tutor (*sprints*) y según se iban terminando las tareas (*issue*) se sincronizaba con el repositorio, lo que conlleva, que el tutor y el alumno tuvieran siempre el mismo código y se pudieran solucionar los problemas de forma rápida [18].

4.2. Patrones de diseño

Model-View-Presenter (MVP)

MVP es un patrón de arquitectura derivado del *Model–View–Controller* (MVC). Fue de los primeros patrones relacionados con las vistas gráficas de interfaz de usuario. Permite separar los datos de nuestra aplicación en un modelo y enlazarlos con un controlador a las vistas que serán lo que ve el usuario [16].

Se compone de las tres capas con las que lleva su nombre:

- ***Modelo***: Los diferentes tipos de datos en los que se compone la aplicación.
- ***View***: Como el usuario va a visualizar los datos, una vista no tiene porqué ser general para todos, cada usuario puede tener la suya propia.
- ***Controlador***: Comunica la capa modelo con las vistas. Comunica datos del modelo para mostrárselos al usuario.

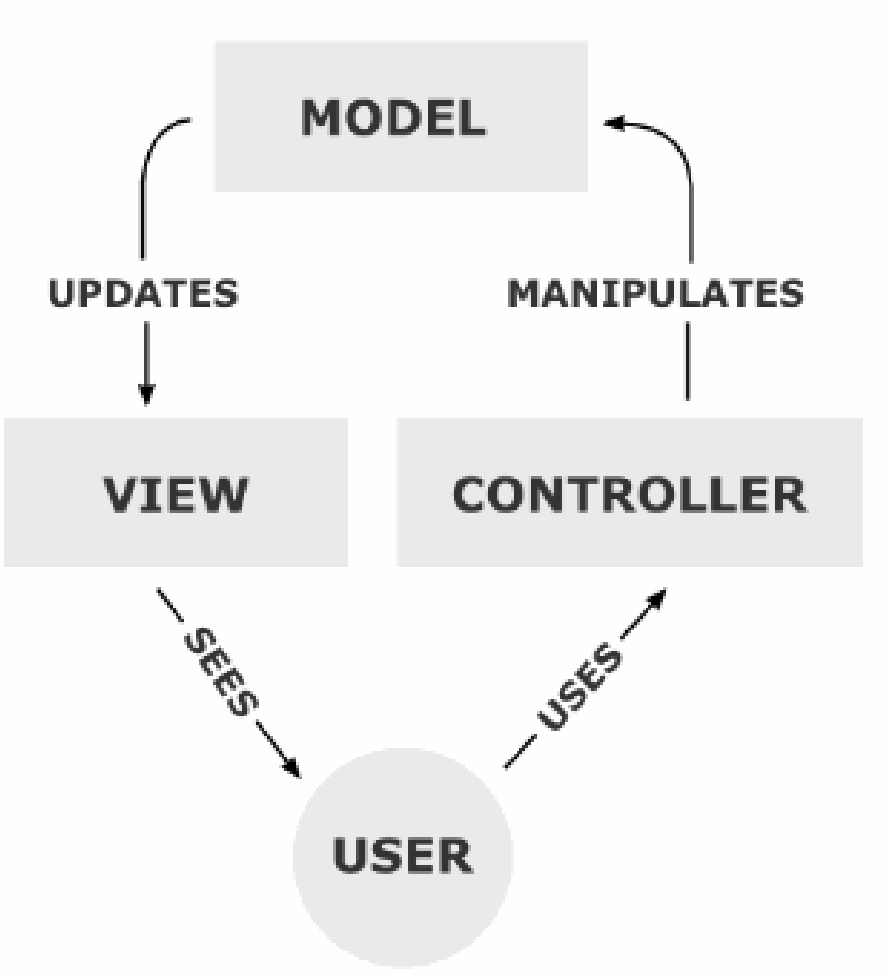


Figura 4.5: Patrón MVP.

4.3. Control de versiones

- Herramientas consideradas: Git y Subversion.

- Herramienta elegida: Git.

Git es un sistema de control de versiones distribuido. Nos permite tener dos repositorios, un repositorio local y otro repositorio remoto, los dos tendrán como rama principal la *master*, en la cual, cuando consideremos que los cambios de nuestro repositorio *local* pueden subirse al *master*, podremos subirlo para compartirlo con el equipo. No nos hace falta estar permanentemente conectados a una red como con Subversion. Como no requiere de conexión, nos podríamos retraer a un commit anterior, ya que tenemos el historial de log en nuestro repositorio *local*, sin internet y de forma local [5].

Git se distribuye bajo la licencia de *software* libre GNU LGPL v2.1.

4.4. *Hosting* del repositorio

- Herramientas consideradas: GitHub, Bitbucket y GitLab.
- Herramienta elegida: GitHub.

GitHub es una plataforma web donde podemos hospedar nuestro repositorios del proyecto. Nos permite todas las funcionalidades de Git, revisión de commit, revisión de código, documentación, gestión de tareas.... Es gratuita cuando los proyectos son *open source*.

Con el plugin de Chrome Zedhub, en GitHub, nos permite tener la funcionalidad de canvas para la organización de tareas, como la plataforma Trello. Como estamos utilizando servicios de integración continua, teniendo el repositorio en GitHub, nos permite la sincronización solo con nuestra cuenta de GitHub [15].

4.5. Gestión del proyecto

- Herramientas consideradas: ZenHub, Trello
- Herramienta elegida: ZenHub.

ZenHub es un plugin desarrollado para exploradores web para integrar una nueva vista, más organizativa, en GitHub.

Proporciona un tablero canvas en donde cada tarea representada se corresponde con un *issue* o tarea, nativo de GitHub. Cada tarea puede ser organizada en el *sprint* al que pertenece con la asignación de ella, su tiempo

estimado de realización, etc. Si el proyecto tiene menos de 5 colaboradores es *open source* [14].

4.6. Entorno de desarrollo integrado (IDE)

Java

eclipse neon

Es una aplicación con un entorno de desarrollo integrado (*IDE*), nos facilita el trabajo a la hora de programar, nos proporciona un auto completar de métodos y clases y nos enseña, si lo tiene, una descripción de los métodos. Esto nos ayuda en un primer momento si no se conoce la librería a utilizar o los métodos. En nuestro caso lo utilizamos para programación Java, aunque nos permite otro lenguages, como puede ser PHP. Eclipse también nos permite funciones de caracterización que nos ahorran horas de trabajo y podemos conectarlo con Git directamente para hacer commit. y ver de forma directa los cambios hechos o los ficheros que no están commiteados [3].

LaTeX

TexStudio

TexStudio es un editor para crear documentos LaTeX. Nos permite compilar el documento, previsualizarlo, autocompletarlo, corrección de faltas. Se puede instalar en diferentes sistemas operativos. En el entorno Ubuntu 16.04, la instalación es sencilla, aparte del texStudio solo hay que instalar los paquetes de idiomas y ya podremos empezar a trabajar [12].

JavaScript y HTML

Sublime Text

Es una aplicación con un entorno de desarrollo integrado (*IDE*), nos permite desarrollar proyectos en diferentes lenguajes, como Python, PHP , Java, HTML , JavaScript, etc. Como eclipse no tiene plugins para JavaScript hemos optado por programar con sublime para HTML y JavaScript [11].

4.7. Servicios de integración continua

Compilación y testeo

TravisCI

Travis es una herramienta de integración continua que está alojada en la nube y que podemos sincronizarla con GitHub. Esto nos permite, cada vez que hacemos un commit, montar nuestro proyecto y ejecutar las pruebas de forma automatizada, y cuando termine nos enviara un informe del resultado [13].

Calidad del código

SonarQube

SonarQube es una plataforma que tiene versión de escritorio y también está alojada en la nube, de código abierto. Nos permite la sincronización con nuestro repositorio en GitHub y nos evalúa el código. Permite detectar bugs, código smell, código duplicado, etc. Es muy útil para mejorar el mantenimiento de tu proyecto [4].

4.8. Sistemas de construcción automática del *software*

Gradle

Gradle Es una herramienta para la automatización de build, puede ser instalada en el sistema operativo o como plugin en eclipse, el plugin tiene ciertas limitaciones, como por ejemplo la especificación de errores. hay veces que no te los dice y tienes que ejecutarlo desde consola, por eso, en nuestro proyecto, se ha utilizado los dos [6].

4.9. Librerías

JavaFX

JavaFX es una librería para la creación de interfaces gráficas en Java.

Log4j

Log4j es una librería de Java desarrollada por la compañía Apache Software Fundation que nos permite en nuestra aplicación escribir mensajes de registro, como pueden ser errores o warnings. También nos permite configurar el filtrado de mensajes para que el programador pueda ver los tipos de errores que desee [9].

HttpClient

HttpClient es una librería de Google diseñada para Java, lo que nos permite es hacer peticiones HTTP. Con esta librería haremos las peticiones a UBUVirtual para que nos de los datos correspondientes [1].

Commons-CSV

Commons-CSV es una librería creada por Apache Software Fundation para Java, que nos facilita la obtención de datos de un CSV [2].

HtmlUnit

- Herramientas consideradas: Jsoup, Selenium y HtmlUnit.
- Herramienta elegida: HtmlUnit.

HtmlUnit es una librería para Java, que nos permite la interacción en páginas web o *web scripting*, nos permite rellenar formularios, hacer clic sobre botones, incluso recoger archivos descargados de la web [7].

4.10. Otras herramientas

Moodle 3.3

Moodle es una plataforma de gestión de aprendizaje que lo utilizan muchos centros dedicados a la enseñanza, entre ellos, la universidad de Burgos.

Mendeley

Mendeley es un gestor de citas bibliográficas. Con el podremos guardar todas las referencias que estemos utilizando y exportarlas a BibTex para LaTeXsin mucho esfuerzo.

JsonView

Json este plugin de Chrome nos facilita la lectura de los Json formateando y coloreándolo.

Scene Builder

Scene Builder es una aplicación para diseñar la interfaz de usuario de una manera sencilla. Scene builder es de Oracle con ella podremos generar los archivos .fxml que serán los que cargaremos para ver las vistas del usuario.

Pinta

Pinta es una aplicación para Ubuntu que se encarga de la creación y edición de imágenes.

MySQL WorkBench

MySQL WorkBench es una aplicación para desarrolladores o diseñadores de bases de datos o que tengan que trabajar con bases de datos. Nos permite importar o exportar BD, ver su entidad/relación para saber como se comunican las tablas, saber su estructura, etc [10].

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, así como las decisiones tomadas y su repercusión.

5.1. Inicio del proyecto

El proyecto comenzó el 3 de octubre del 2017. La idea surgió de un tfg anterior el de Claudia Martínez Herrero [17] en el cual se iniciaba sesión en UBUVirtual y tenía que sacar con gráficas las notas de los diferentes alumnos. Al tutor Raúl Marticorena Sánchez, se le ocurrió dar una vuelta a esa idea y partiendo de ese tfg, queríamos sacar las estadísticas de las interacciones de los diferentes usuarios de UBUVirtual en una asignatura concreta para poder mostrarlo al profesor de forma más amigable.

Figura 5.6: Logo UBULog.

5.2. Metodologías

En la primera reunión se decidió trabajar con metodologías ágiles, en concreto Scrum. Durante el proyecto se han llevado a cabo reuniones semanales con el tutor (*sprints*) y según se iban terminando las tareas (*issue*) se sincronizaba con el repositorio, lo que conlleva, que el tutor y el alumno tuvieran siempre el mismo código y se pudieran solucionar los problemas de forma rápida.

5.3. Formación

Algunas tecnologías utilizadas en el proyecto requerían formación antes de empezar a trabajar con ellas.

- Aplicación UBUGrades, se tuvo que probar y refactorizar el código para facilitar su lectura y saber que partes nos valían y cuales podríamos desechar.
- Librería Common-csv-1.5, se investigó su JavaDoc para saber si nos podía parsear nuestros documentos.
- Librería HtmlUnit, Se tuvo que investigar la librería y hacer diferentes pruebas antes de su implementación
- Librería Calendar, se investigó su JavaDoc antes de empezar con la implementación, en un primer momento no funciono y se tuvo que analizar el código debuggeando, se vio que el JavaDoc estaba mal documentado y se vio como utilizar esta librería de forma correcta.
- Librería chart.js, se miró la documentación [8] y se analizó su código para su implementación.

5.4. Decisiones técnicas

Este proyecto ha sido una continuación parcial de proyecto de Claudia Martínez Herrero [17] con lo cual, en los puntos coincidentes, explicaremos las diferencias existentes y añadiremos nueva información.

Utilización de Moodle y sus servicios web

La única diferencia en este punto es, que nosotros no hemos generado calificaciones, ya que nosotros no lo necesitamos, porque vamos a mirar las interacciones de los usuarios con el log que genera Moodle (la descarga del log se explicara en la guía de usuario).

Funciones utilizadas

Nosotros hemos descartado algunas llamadas de las originales, ya que no las necesitamos.

- gradereport_user_get_grades_table.
- mod_assign_get_assignments.
- mod_quiz_get_quizzes_by_courses.

Obtención del token de usuario

No se aporta nada nuevo.

Extracción de datos en JSON

No se aporta nada nuevo.

Generar gráficos

Para generar los gráficos, ya que se ha optado por la librería chart.js que es de JavaScript, necesitaremos un HTML para poder ejecutar ese javaScript y mostrarlo en la etiqueta HTML5 <canvas>. En un primer momento, se consideró que el .html iría en resources y el .js que esta creado en tiempo de ejecución igual. Todo funcionaba de forma correcta hasta que se generó el .jar, dado que en el .jar no se puede escribir, nos empezó a dar excepciones de que ese archivo no existía, entonces optamos por generarlos en tiempo de ejecución todos los archivos. Así solucionamos los problemas de ejecución en el .jar.

Cabe destacar que cuando se cierra la aplicación nos aseguramos de eliminar los archivos generados.

A continuación explicaremos los métodos correspondientes para generar los gráficos.

Constructor

Con el crearemos el objeto Chart.

```
44    /**
45     * Constructor de clase.
46     */
47    public Chart() {
48        dates = new ArrayList<>();
49        label = new HashMap<>();
50        setTypeChart("bar");
51        createHTML();
52        createCSS();
53        createUtils();
54        generarGrafica();
55    }
56
```

Figura 5.7: Constructor Chart.

HTML

El HTML necesario para las gráficas, se genera en tiempo de ejecución. El método con el cual se genera es el siguiente.

```
/**
 * Metodo que crea el html de chart.
 */
private void createHTML() {
    try (FileWriter ficheroJS = new FileWriter("./chart.html"); PrintWriter pw = new PrintWriter(ficheroJS)) {
        pw.println("<!doctype html><html><head>" + "<meta charset=\"utf-8\">"
                + "<title>Gráfico de interacción en UbuVirtual</title>"
                + "<link rel=\"stylesheet\" href=\"./chart.css\">"
                + "<script type=\"text/javascript\" src=\"https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.1/Chart.bundle.js\"></script>"
                + "<script type=\"text/javascript\" src=\"https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.1/Chart.js\"></script>"
                + "<script type=\"text/javascript\" src=\"./Chart.js\"></script> "
                + "<script type=\"text/javascript\" src=\"./utils.js\"> </script>"
                + "<style>canvas {-moz-user-select: none;-webkit-user-select: none;-ms-user-select: none;}</style>"
                + "</head><body>" + "<div id=\"container\" style=\"width: 100%;\">"
                + " <canvas id=\"canvas\"></canvas>" + "</div></body></html>");

    } catch (Exception e) {
        logger.error("Error al generar html. {}", e);
    }
}
```

Figura 5.8: Código HTML para el gráfico.

CSS

El CSS necesario para las gráficas, se genera en tiempo de ejecución. El método con el cual se genera es el siguiente.

```
/**
 * Crear css.
 */
private void createCSS() {
    try (FileWriter ficheroCSS = new FileWriter("./chart.css"); PrintWriter pw = new PrintWriter(ficheroCSS)) {
        pw.println("@import url('https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css');"
                + "@import url('https://fonts.googleapis.com/css?family=Lato:100,300,400,700,900');"
                + "body, html {color: #333538;font-family: 'Lato', sans-serif;line-height: 1.6;padding: 0;margin: 0;}a {color: #f27173;text-decoration: none;}a:hover {color: #e25f5f;text-decoration: underline;
                + "}.content {  max-width: 800px;   margin: auto;   padding: 16px 32px;}.header {text-align: center;    padding: 32px 0;}.wrapper { min-height: 400px;"
                + "padding: 16px 0;position: relative;}.wrapper.col-2 { display: inline-block;  min-height: 256px;  width: 49%;}@media (max-width: 400px) { .wrapper.col-2 {width: 100% }}"
                + ".wrapper canvas {    -moz-user-select: none;  -webkit-user-select: none;   -ms-user-select: none;}"
                + ".toolbar {display: flex;}.toolbar > * {  margin: 0 8px 0 0;}.btn {   background-color: #aaa; border-radius: 4px; color: white;   padding: 0.25rem 0.75rem;}"
                + ".btn .fa {   font-size: 1rem;}.btn:hover {   background-color: #888; color: white;   text-decoration: none;}"
                +

                ".btn-chartjs { background-color: #f27173; }.btn-chartjs:hover { background-color: #e25f5f; }.btn-docs:hover { background-color: #2793db; }.btn-docs { background-color: #36A2EB; }.btn-docs:hove
                +

                ".btn-on {  border-style: inset;}.chartjs-title {   font-size: 2rem;    font-weight: 600;   white-space: nowrap;}.chartjs-title::before {   background-image: url(logo.svg); "
                + " background-position: left center;background-repeat: no-repeat;background-size: 40px;content: 'Chart.js | ';color: #f27173;font-weight: 600;"
                + " padding-left: 48px;}.chartjs-caption {  font-size: 1.2rem;}.chartjs-links { display: flex;  justify-content: center;    padding: 0px 0;}"
                +

                ".chartjs-links a {align-items: center;display: flex;   font-size: 0.9rem;  margin: 0.2rem;}.chartjs-links .fa:before { margin-right: 0.5em;}"
                +

                ".samples-category {    display: inline-block;  margin-bottom: 32px;    vertical-align: top;    width: 25%;}.samples-category > .title {    color: #aaa;"
                + "font-weight: 300;    font-size: 1.5rem;}.samples-category:hover > .title {   color: black;}.samples-category > .items {  padding: 8px 0; "
                + "}.samples-entry {padding: 8 0 4px 0;}.samples-entry > .title {   font-weight: 700;}@media (max-width: 640px) {   .samples-category { width: 33%; }}"
                +

                "@media (max-width: 512px) {    .samples-category { width: 50%; }}@media (max-width: 420px) {    .chartjs-caption { font-size: 1.05rem; }    .chartjs-title::before { content: ''; }"
                + ".chartjs-links a { flex-direction: column; }    .chartjs-links .fa { margin: 0 }    .samples-category { width: 100%; }}.analyser table {"
                + " color: #333;    font-size: 0.9rem;  margin: 8px 0;  width: 100%}.analyser th {  background-color: #f0f0f0;  padding: 2px;}"
                + ".analyser td {   padding: 2px;   text-align: center;}");

    } catch (Exception e) {
        logger.error("Error al generar html. {}", e);
    }

}
/**
```

Figura 5.9: Código CSS del gráfico.

Utils.js

El JavaScript necesario para generar gráficas, se genera en tiempo de ejecución. El método con el cual se genera es el siguiente.

```
    /**
     * Crear utils.
     */
    private void createUtils() {
        try (FileWriter ficheroCSS = new FileWriter("./utils.js"); PrintWriter pw = new PrintWriter(ficheroCSS)) {
            pw.println("'use strict';" +
"'use strict';" +
"" +
"window.chartColors = {" +
"   red: 'rgb(255, 99, 132)'," +
"   orange: 'rgb(255, 159, 64)'," +
"   yellow: 'rgb(255, 205, 86)'," +
"   green: 'rgb(75, 192, 192)'," +
"   blue: 'rgb(54, 162, 235)'," +
"   purple: 'rgb(153, 102, 255)'," +
"   grey: 'rgb(201, 203, 207)'" +
"};" +
"" +
"(function(global) {" +
"   var Months = [" +
"       'January'," +
"       'February'," +
"       'March'," +
"       'April'," +
"       'May'," +
"       'June'," +
"       'July'," +
"       'August'," +
"       'September'," +
"       'October'," +
"       'November'," +
"       'December'" +
"   ];" +
"" +
"   var COLORS = [" +
"       '#4dc9f6'," +
"       '#f67019'," +
"       '#f53794'," +
"       '#537bc4'," +
"       '#acc236'," +
"       '#166a8f'," +
"       '#00a950'," +
"       '#58595b'," +
"       '#8549ba'" +
"   ];" +
"" +
"   var Samples = global.Samples || (global.Samples = {});" +
"   var Color = global.Color;" +
"" +
"   Samples.utils = {" +
"       // Adapted from http://indiegamr.com/generate-repeatable-random-numbers-in-js/" +
"       srand: function(seed) {" +
"           this._seed = seed;" +
```

Figura 5.10: Código utils.js necesario para el gráfico.

chart.js

El JavaScript necesario para las gráficas, se genera en tiempo de ejecución. El método con el cual se genera es el siguiente.

```
    * Método para crear el javaScript que nos cargara la gráfica.
    */
   public void generarGrafica() {

       try (FileWriter ficheroJS = new FileWriter("./Chart.js"); PrintWriter pw = new PrintWriter(ficheroJS)) {
           pw.println(
                   "var MONTHS = [\"January\", \"February\", \"March\", \"April\", \"May\", \"June\", \"July\", \"August\", \"September\", \"October\", \"November\", \"December\"];");
           pw.println("var colorNames = [\"red\", \"orange\", \"yellow\", \"green\", \"blue\", \"purple\", \"grey\"];");
           pw.println("var color = Chart.helpers.color;");
           pw.println("var barChartData = {");

           pw.print("\tlabels: [");
           for (int i = dates.size(); i > 0; i--) {
               if (0 != i - 1) {
                   pw.print("\"" + dates.get(i - 1) + "\",");

               } else {
                   pw.print("\"" + dates.get(i - 1) + "\"");
               }

           }
           pw.println("],");
           pw.print("\t\tdatasets: [");

           setDataSetJavaScript(pw);

           pw.println("\t]");
           pw.println("};");

           pw.println("window.onload = function() {");
           pw.println("\tvar ctx = document.getElementById(\"canvas\").getContext(\"2d\");");
           pw.println("\twindow.myBar = new Chart(ctx, {");
           pw.println("\t\ttype: '" + typeChart + "',");
           pw.println("\t\tdata: barChartData,");
           pw.println("\t\toptions: {");
           pw.println("\t\t\tresponsive: true,");
           pw.println("\t\t\tlegend: {");
           pw.println("\t\t\t\tposition: 'top',");
           pw.println("\t\t\t},");
           pw.println("\t\t\ttitle: {");
           pw.println("\t\t\t\tdisplay: true,");
           pw.println("\t\t\t\ttext: 'Gráfico de interacción en UbuVirtual'");
           pw.println("\t\t\t}");
           pw.println("\t\t}");
           pw.println("\t});");
           pw.println("};");

           pw.println("function colorDataSet( num){");
           pw.println("\treturn colorNames[num % colorNames.length];");
           pw.println("};");

       } catch (Exception e) {
           logger.error("Error al generar gráfica. {}", e);
       }
   }
```

Figura 5.11: Código chart general.js.

En este código lo que hacemos es poner la configuración del gráfico, el tipo, y los datos. Las fechas las insertamos en este método, para insertar los datos que queremos mostrar las gráficas llamamos al método setDataSetJavaScript pasándole el objeto PrinterWriter, el código es el siguiente.

```
463    /**
464     * Método para añadir los datos al javaScript.
465     *
466     * @param pw,
467     *            para escribir en el fichero.
468     */
469    private void setDataSetJavaScript(PrintWriter pw) {
470        int tam = label.size();
471        int cont = 0;
472        for (String dataset : label.keySet()) {
473            pw.println("{");
474            pw.print("\t\t\tlabel:");
475            pw.println("'" + dataset + "',");
476
477            pw.println("\t\t\tbackgroundColor: colorDataSet(" + cont
478                    + "),");
479            pw.println("\t\t\tborderColor: colorDataSet(" + cont
480                    + "),");
481            pw.println("\t\t\tborderWidth: 1,");
482
483            pw.println("\t\t\tdata: [");
484            for (int i = label.get(dataset).size(); i > 0; i--) {
485                if (0 != i - 1) {
486                    pw.println("\t\t\t\t" + label.get(dataset).get(i - 1) + ",");
487
488                } else {
489                    pw.println("\t\t\t\t" + label.get(dataset).get(i - 1));
490                }
491
492            }
493            cont++;
494            pw.println("\t\t\t]");
495            if (cont != tam) {
496                pw.print("\t\t},");
497
498            } else {
499                pw.print("\t\t}");
500            }
501        }
502    }
```

Figura 5.12: Código chart datos.js.

En este código generamos tantos DataSet como tengamos ya almacenamos con el número de interacciones correspondientes. Para obtener estos datos el método que almacena los datos es el siguiente.

```
    /**
     * Metodo para meter los diferentes eventos con participantes y el numero de
     * interacciones con ellos.
     *
     * @param selectedParticipants,
     *            participantes seleccionados.
     * @param selectedEvents,
     *            eventos seleccionados.
     * @param filterLogs,
     *            logs.
     */
    public void setLabel(ObservableList<EnrolledUser> selectedParticipants, ObservableList<Event> selectedEvents,
            ArrayList<Log> filterLogs) {
        int cont = 0;
        String fechaLog = null;
        asignedUserMonth(filterLogs);

        if (selectedEvents.isEmpty()) {
            for (EnrolledUser participant : selectedParticipants) {
                ArrayList<Integer> cantidad = new ArrayList<>();
                for (int i = 0; i < dates.size(); i++) {
                    for (Log log : filterLogs) {
                        fechaLog = MONTH[log.getDate().get(Calendar.MONTH)] + " " + log.getDate().get(Calendar.YEAR);
                        if (log.getUser().getFullName().equals(participant.toString())
                                && fechaLog.equals(dates.get(i))) {
                            cont += 1;
                        }
                    }
                    cantidad.add(cont);
                    cont = 0;
                    this.label.put(participant.getFullName(), cantidad);

                }

            }
        } else {
            if (selectedParticipants.isEmpty()) {

                for (Event event : selectedEvents) {
                    ArrayList<Integer> cantidad = new ArrayList<>();
                    for (int i = 0; i < dates.size(); i++) {
                        for (Log log : filterLogs) {
                            fechaLog = MONTH[log.getDate().get(Calendar.MONTH)] + " "
                                    + log.getDate().get(Calendar.YEAR);
                            if (log.getEvent().equals(event.toString()) && fechaLog.equals(dates.get(i))) {
                                cont += 1;
                            }
                        }
                        cantidad.add(cont);
                        cont = 0;
                        this.label.put(event.getNameEvent(), cantidad);

                    }

                }
            } else {
```

Figura 5.13: Código que obtiene datos para el gráfico 1.

```
                }
            } else {
                for (EnrolledUser participant : selectedParticipants) {
                    for (Event event : selectedEvents) {
                        ArrayList<Integer> cantidad = new ArrayList<>();
                        for (int i = 0; i < dates.size(); i++) {
                            for (Log log : filterLogs) {
                                fechaLog = MONTH[log.getDate().get(Calendar.MONTH)] + " "
                                        + log.getDate().get(Calendar.YEAR);
                                if (log.getEvent().equals(event.toString())
                                        && log.getUser().getFullName().equals(participant.toString())
                                        && fechaLog.equals(dates.get(i))) {
                                    cont += 1;
                                }
                            }
                            cantidad.add(cont);
                            cont = 0;
                            this.label.put(participant.getFullName() + "/" + event.getNameEvent(), cantidad);

                        }

                    }
                }
            }
        }
    }
```

Figura 5.14: Código que obtiene datos para el gráfico 2.

Este método tiene 3 condiciones principales, si solo ha seleccionado participantes, si solo ha seleccionado eventos o si selecciona ambas.

En la condición más compleja, que es seleccionar ambas, lo que ocurre, es:

- Iteramos cada participante seleccionado.
- Con cada participante seleccionado iteramos cada evento seleccionado.
- Con cada evento seleccionado iteramos cada set de fechas de los log.
- Con cada set de fechas de los log iteramos cada uno de los logs.
- Si el mes y el año, el evento y participante corresponde con el log incrementamos el contador, si no, no. Almacenamos los valores en una lista para saber el número de veces que un participante, en un evento concreto, en esa fecha.

Generar tabla logs

En esta parte podremos ver la información concreta de los log, se implementaron unos filtros con los cuales podremos desgranar más el log y volver a generar una gráfica más específica con los datos resultantes de la tabla.

Cabe destacar que cuando se cierra la aplicación nos aseguramos de eliminar los archivos generados.

Con este HTML tenemos el mismo problema que en la generación de gráficas en la ejecución de .jar, con lo cual lo generamos en tiempo de ejecución de la misma manera.

HTML

```
public void generarTablaLogs(List<Log> list) {

    try(FileWriter ficheroHTML = new FileWriter("./tablelogs.html");
        PrintWriter pw = new PrintWriter(ficheroHTML); ) {
        String initRow ="\t\t<th>";
        String finalRow = "</th>";
        pw.println("<!DOCTYPE html> \n " + "<html> \n " + "<title>tabla logs</title> \n"
                + "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\"> \n"
                + "<meta charset=\"utf-8\"> \n"
                + "<link rel=\"stylesheet\" href=\"https://www.w3schools.com/w3css/4/w3.css\">\n" + "<body>\n"
                + "<div class=\"w3-container\"> \n" + "\t<h2>Tabla de logs</h2>");

        pw.println("\t<table class=\"w3-table-all w3-margin-top\" id=\"myTable\">");
        pw.println("\t\t<tr> \n" + initRow + "Fecha " + finalRow + "");
        pw.println(initRow + "Nombre completo del usuario" + finalRow + "");
        pw.println(initRow + "Usuario afectado" + finalRow + "");
        pw.println(initRow + "Contexto del evento" + finalRow + "");
        pw.println(initRow + "Componente" + finalRow + "");
        pw.println(initRow + "Nombre evento" + finalRow + "");
        pw.println(initRow + "Descripción" + finalRow + "");
        pw.println(initRow + "Origen" + finalRow + "");
        pw.println(initRow + "Dirección IP" + finalRow + "");
        pw.println("\t</tr>");

        for (Log log : list) {
            dataTableLog(pw, log);
        }

        pw.println("\t</table>");
        pw.println("</div> \n" + "</body>\n" + "</html>");

    } catch (Exception e) {
        logger.error("Error al generar tabla logs. {}", e);
    }
}
```

Figura 5.15: Código que genera la tabla log 1.

```
    /**
     * Método para construir la tabla de logs.
     * @param pw, printerWriter.
     * @param log, logs.
     */
    private void dataTableLog(PrintWriter pw, Log log) {
        String initRow ="\t\t<td>";
        String finalRow = "</td>";
        pw.println("\t<tr>");

        pw.println(initRow + log.getDate().get(Calendar.DAY_OF_MONTH) + "/" + (log.getDate().get(Calendar.MONTH)+1) + "/"
                + log.getDate().get(Calendar.YEAR) + " "+ log.getDate().get(Calendar.HOUR_OF_DAY)+ ":"+log.getDate().get(Calendar.MINUTE)+  finalRow);
        pw.println(initRow + log.getNameUser() + "</td>");
        pw.println(initRow + log.getUserAffected() + finalRow);
        pw.println(initRow + log.getContext() + finalRow);
        pw.println(initRow + log.getComponent() + finalRow);
        pw.println(initRow + log.getEvent() + finalRow);
        pw.println(initRow + log.getDescription() + finalRow);
        pw.println(initRow + log.getOrigin() + finalRow);
        pw.println(initRow + log.getIp() + finalRow);

        pw.println("\t</tr>");
    }
```

Figura 5.16: Código que genera la tabla log 2.

Creamos las cabeceras y una vez hecho eso recorremos cada log para obtener sus datos y poder crear la tabla completa.

Web Scripting

Con Web Scripting lo que queremos conseguir es la automatización de la descarga de logs.

Hemos observado que con la descarga de logs considerablemente grandes(con 20.000 instancias) el tiempo se incrementa considerablemente, es muchísimo más rápido la descarga manual del log. Se ha observado que, si se intenta descargar con una conexión wifi, la aplicación puede llegar a colgarse de tal manera que haya que hacer un kill.

Hablaremos del código en las siguientes sub-secciones.

Constructor

En el cargaremos la web de UBUVirtual e iniciaremos sesión.

```
    /**
     * Constructor de clase.
     */
    public WebScripting() {
        try {
            client = new WebClient(BrowserVersion.getDefault());
            client.getOptions().setTimeout(0);
            page = client.getPage(UBULog.getHost() + "/login/index.php");
            HtmlForm form = (HtmlForm) page.getElementById("login");

            form.getInputByName("username").setValueAttribute(UBULog.getSession().getUserName());
            form.getInputByName("password").setValueAttribute(UBULog.getSession().getPassword());

            page.getElementById("loginbtn").click();

        } catch (FailingHttpStatusCodeException | IOException e) {
            logger.error(e.getMessage());

        }
    }
```

Figura 5.17: Constructor Web Scripting.

Responsive Web

Una vez iniciado sesión cargamos la URL correspondiente con todos los log, sin ningún tipo de filtro, del curso en el que estamos.

Recogemos la respuesta que nos da al hacer clic en el input de descarga y guardamos la respuesta en un CSV temporal.

El código correspondiente es el siguiente.

```
58    /**
59     * Metodo que pide y guarda la respuesta de log de la web.
70     */
71    public void getResponsiveWeb() {
72        try {
73            page = client.getPage(UBULog.getHost() + "/report/log/index.php?chooselog=1&showusers=0&showcourses=0&id="
74                    + UBULog.getSession().getActualCourse().getId()
75                    + "&user=&date=&modid=&modaction=&origin=&edulevel=-1&logreader=logstore_standard");
76
77            DomNodeList<DomElement> inputs = page.getElementsByTagName("input");
78            int valbtn = -1;
79            for (int i = 0; i < inputs.getLength(); i++) {
30                if (inputs.get(i).getAttribute("type").equals("submit")) {
31                    valbtn = i;
32                }
33            }
34
35            WebResponse dataDownload = inputs.get(valbtn).click().getWebResponse();
36            setResponsive(dataDownload.getContentAsString());
37
38        } catch (FailingHttpStatusCodeException | IOException e) {
39            logger.error(e.getMessage());
30        }
31    }
```

Figura 5.18: Método getResponsiveWeb.

Decisiones en cuanto a la interfaz

El proyecto a partido del diseño del TFG de Claudia Martínez Herrero [17] con algunas modificaciones.

Ventana de bienvenida

No se aporta nada nuevo.

Barra de progreso

No se aporta nada nuevo.

Ventana de principal

Esta ventana, se ha tenido que crear desde 0, porque no se redimensionaba de forma correcta, aunque se ha seguido el diseño que ya estaba, se ha cambiado botones y añadido funcionalidades nuevas, al igual que se han suprimido otras.

el resultado es el siguiente.

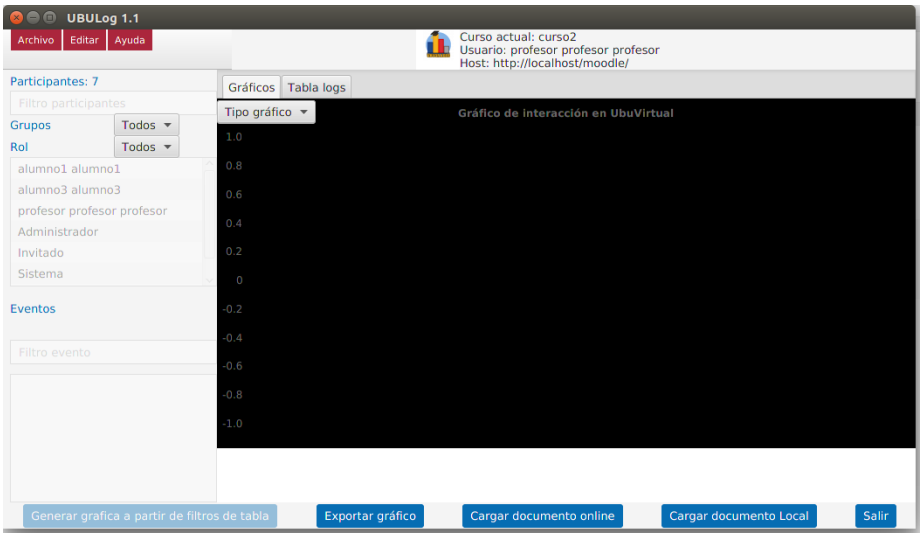


Figura 5.19: Ventana de principal UBULog.

El primer cambio importante que se observa es que, si no se tiene un log cargado, los filtros, las selecciones de eventos y participantes quedan deshabilitadas hasta que se carga e log.

Se añade la foto del profesor que ha iniciado sesión.

Nuevos botones para la carga de log cuando se tiene el registro almacenado en local o si se quiere descargar de forma automática.

Un nuevo botón para generar el gráfico con referencia a los resultados de la tabla de logs.

Gráfica logs

Una vez tengamos el log cargado, podremos seleccionar eventos y participantes, el resultado sería el siguiente.

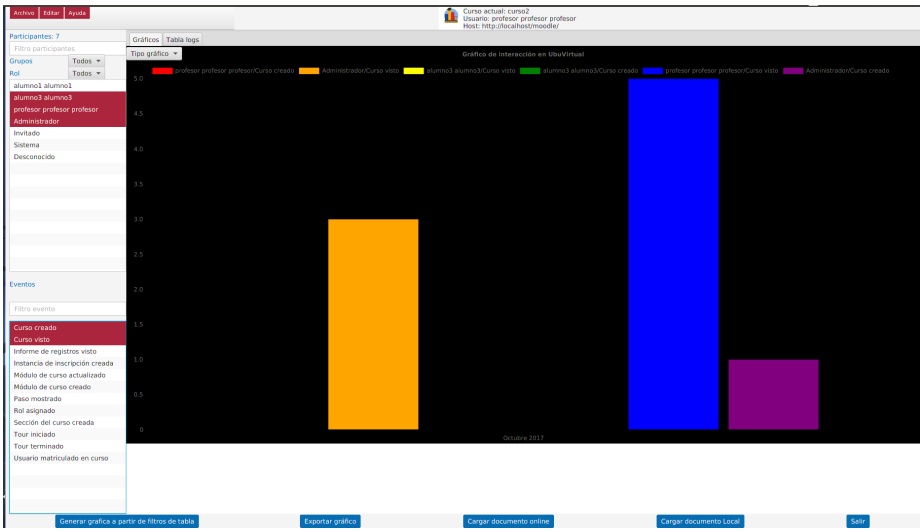


Figura 5.20: Vista de gráfica.

En la gráfica podemos ver en la leyenda las combinaciones seleccionadas, y en el gráfico los resultados. Si no vemos bien en número de interacciones en el gráfico a simple vista, podremos ir con el ratón, colocarnos encima de la barra deseada y nos dará la información correspondiente.

Si deseamos cambiar el tipo de gráfico podremos pinchar en el selector de la gráfica y seleccionar uno diferente.

Tabla logs

Una vez tengamos el log cargado, podremos seleccionar eventos y participantes, el resultado sería el siguiente en la tabla de logs.

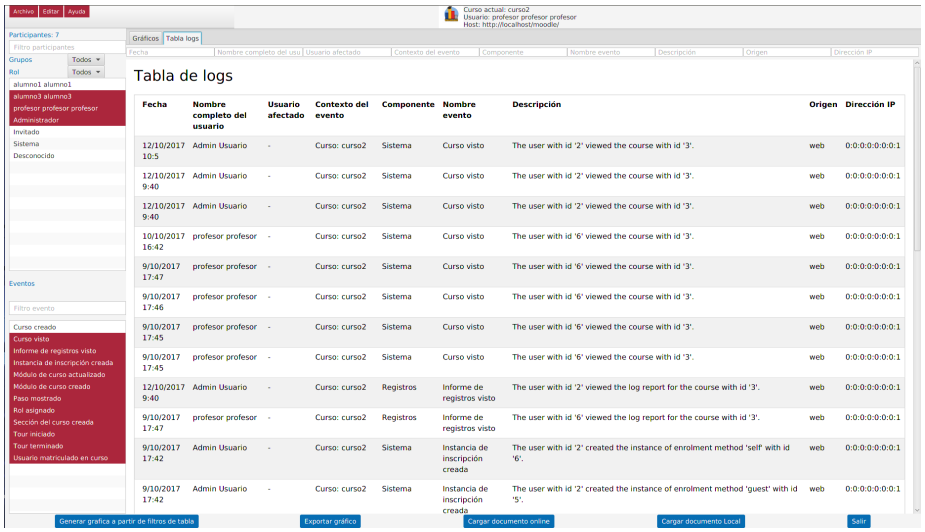


Figura 5.21: Vista tabla de logs.

En la tabla de logs, podremos ver los log de forma individual, con sus características.

En esta tabla, proporcionamos unos filtros, de los cuales, cada uno corresponde con una columna de la tabla. En el caso de que queramos ver los log más desgranados, podremos filtrarlos y cuando muestre el resultado pinchar en el botón generar gráfica a partir de datos de tabla y nos mostrara los datos en la gráfica resultantes.

Conclusiones y Líneas de trabajo futuras

En esta última sección expondremos las conclusiones del trabajo realizado y las líneas futuras que se podrán seguir.

7.1. Conclusiones

En la finalización del proyecto podemos sacar las siguientes conclusiones.

- El objetivo general del proyecto se ha cumplido satisfactoriamente. Ahora los profesores cuentan con una aplicación para poder ver las interacciones de los diferentes usuarios, y hacer comprobaciones entre equipos de alumnos para saber quién ha interaccionado con UBUVirtual y quién no, para tener un análisis más detallado.
- El proyecto ha sido desarrollado con un claro pensamiento de mantenibilidad. Esto ha conllevado que, al hacer algún cambio de implementación, el tiempo invertido en ello era mínimo. Esta última consideración está ligada también a una nueva implementación.
- Se ha observado la importancia de las herramientas como Travis o SonarQube para el desarrollo.
- El editor de LaTeX ha sido de gran ayuda en la documentación, aunque sea de mayor costumbre utilizar Word o Writer, ha sido bueno el aprendizaje de esta tecnología para futuros usos en las empresas privadas.
- Hay que tener cuidado con la documentación que existe en internet, aunque sea la oficial, en alguna librería utilizada la documentación

conducía a errores de implementación y hay que observar el propio código.
- Al utilizar la metodología ágil scrum, se ha desarrollado el proyecto de la forma más profesional posible, sufriendo la presión de las entregas, lo que será habitual en el entorno laboral.
- Es complicado calcular la duración de los issue, se puede decir, que he sido optimista en el cálculo, excepto en un par de issue los demás se finalizaban antes de lo esperado.

7.2. Líneas de trabajo futuras

- Buscar funciones de la API Web Service de Moodle para dar información más concreta al usuario, diciéndole, por ejemplo, información concreta de cuestionarios, preguntas, contenido, etc....
- Investigar la mejora de tiempos del Web Scripting o incluso buscar una librería mejor para este proceso. El tiempo que tarda en descargar los log es muy grande y cuánto más grande es el log aumenta, de tal forma, que puede llegar a bloquear la aplicación.
- Como en los log tenemos la IP por donde se han conectado a UBUVirtual, podríamos hacer estadísticas de por donde se conectan las personas o incluso hacer minería con estos datos para saber cuándo se conectan más.
- Añadir más tipos de gráficos. Hemos metido 3, pero convendría poner más tipos ya que hay muchos tipos de usuarios y para diferentes datos habrá gustos diferentes de gráficos.
- Al descargar el log, hay unos select que nos permite elegir ciertas opciones para filtrar los log y después descargar esos log. Habría que implementar para que el usuario se pueda descargar el log automáticamente modificando esas opciones.
- Implementar traducciones para tener una aplicación en diferentes idiomas.
- Ampliar formatos de lectura de registros a .xlsx, .html, .jsom, .odt.

Bibliografía

[1] Biblioteca de clientes HTTP para Java | Desarrolladores de Google.

[2] CSVParser (Apache Commons CSV 1.5.1-SNAPSHOT API).

[3] Eclipse Neon.

[4] GII-17.1B-UBULog-1.0 - trona85.

[5] Git - Documentation.

[6] Gradle Build Tool.

[7] HtmlUnit – Welcome to HtmlUnit.

[8] Libreria para generar los graficos - Chart.js · Cocumentación.

[9] Log4j – Apache Log4j 2 - Apache Log4j 2.

[10] MySQL :: MySQL Workbench.

[11] Sublime Text 3 - IDE para javaScript y html.

[12] TeXstudio.

[13] Travis CI - Integración continua.

[14] zenhub.

[15] GitHub, 2017.

[16] modelo vista controlador — Wikipedia, La enciclopedia libre. 2017.

[17] Claudia Martínez Herrero. *Memoria tfg, 2017* .

[18] Ken. Schwaber. *Agile project management with Scrum.* Microsoft Press, 2004.