

# Manual del programador

Sistema: **XperiencUML: herramienta web para creación de objetos de aprendizaje con empaquetado CMI-5**

## Contenido

Introducción.....	3
Objetivo.....	3
Alcance.....	3
Requerimientos técnicos .....	4
De software.....	4
De hardware.....	7
Instalación de la aplicación en Symphony .....	8
Obtención y ubicación de la solución. ....	8
Descripción de la estructura de archivos .....	9
Ejecución de los servicios de Apache y MySQL.....	10
Actualización de componentes.....	11
Creación de la base de datos.....	12
Inicializar el servidor.....	16
Diseño de la aplicación .....	17
Actores de la aplicación .....	17
Módulos de la aplicación.....	17
Autenticación.....	17
Usuarios.....	18
Cursos.....	18
Actividades .....	19
Reportes.....	19
Declaraciones.....	19
Creación de Diagramas.....	20
Descripción e implementación de la aplicación .....	20
Autenticación.....	21
Gestión de usuarios.....	27
Gestión de cursos.....	37
Gestión de actividades.....	51
Gestión de diagramas UML.....	66
Gestión de diagramas para Docentes.....	71
Gestión de diagramas para Estudiantes .....	87
Gestión de declaraciones.....	100
Gestión de reportes .....	103
Otros ajustes.....	106

## Introducción

Este documento es una guía para los programadores de la herramienta XPerienceUML. Este manual describe los pasos necesarios para la instalación y administración del aplicativo creado para la creación de objetos de aprendizaje con empaquetado CMI5. Es importante considerar los requisitos mínimos de hardware y software para la correcta instalación del aplicativo.

## Objetivo

El presente manual tiene con objetivo servir como guía práctica para la compresión y descripción de la arquitectura de la herramienta. Está dirigido a programadores, con o sin experiencia. Para los programadores sin experiencia es una ayuda práctica para adquirir conocimientos en la construcción de la herramienta. Mientras que para los programadores con experiencia, este funcionará como documento de consulta.

## Alcance

Está dirigido a programadores del sistema XPerienceUML: herramienta Web para la creación de objeto de aprendizaje con empaquetado CMI5.

En el documento se describen los pasos necesarios para la instalación y administración del aplicativo. También se consideran algunos aspectos como los requisitos mínimos de hardware y software para la correcta instalación y funcionamiento de la herramienta.

El manual contiene los siguientes capítulos: Introducción con objetivo y alcance de la aplicación, Requerimientos técnicos tanto de hardware como de software, Instalación de una aplicación en Symfony, Diseño de la aplicación, así como Descripción e implementación de la solución. Descripción general del sistema, Arquitectura y Aspectos de diseño e implementación.

## Requerimientos técnicos

La aplicación fue realizada como una aplicación Symfony, que es un proyecto PHP, de software libre, que permite crear aplicaciones y sitios web rápidos y seguros de forma profesional.

Para la ejecución de la aplicación luego es necesario tener en cuenta los siguientes requisitos de software y de hardware.

Tecnologías utilizadas			
Nº	Nombre	Versión	Descripción
1	XAMPP	7.3	Software libre que permite gestionar base de datos como MySql, servidores web como Apache.
2	PHP	7.3	Lenguaje de programación orientado al desarrollo web
3	Composer	2.6	Gestor de paquetes para uso de dependencias y librerías de PHP
4	GoJs	2.3	Es una biblioteca de JavaScript que permite la creación de recursos interactivos en aplicaciones web.

### De software

Para la ejecución del sistema es necesario tener instalado:

- Composer como gestor de dependencias para PHP. Se emplea para descargar y gestionar las librerías y dependencias del proyecto PHP de manera automatizada.
- XAMPP versión mayor a 7.3, como Gestor de Base de Datos.
- Microsoft Visual Studio Code, como editor de códigos php, javascript, css.

La instalación de Composer puede descargarse de la página oficial <https://getcomposer.org/>. En la Figura 1 se muestra el diseño de esta página.

Figura 1 Página Composer



La instalación del XAMPP puede descargarse de la página <https://www.apachefriends.org/es/index.html>. En la Figura 2 puede verse la página oficial.

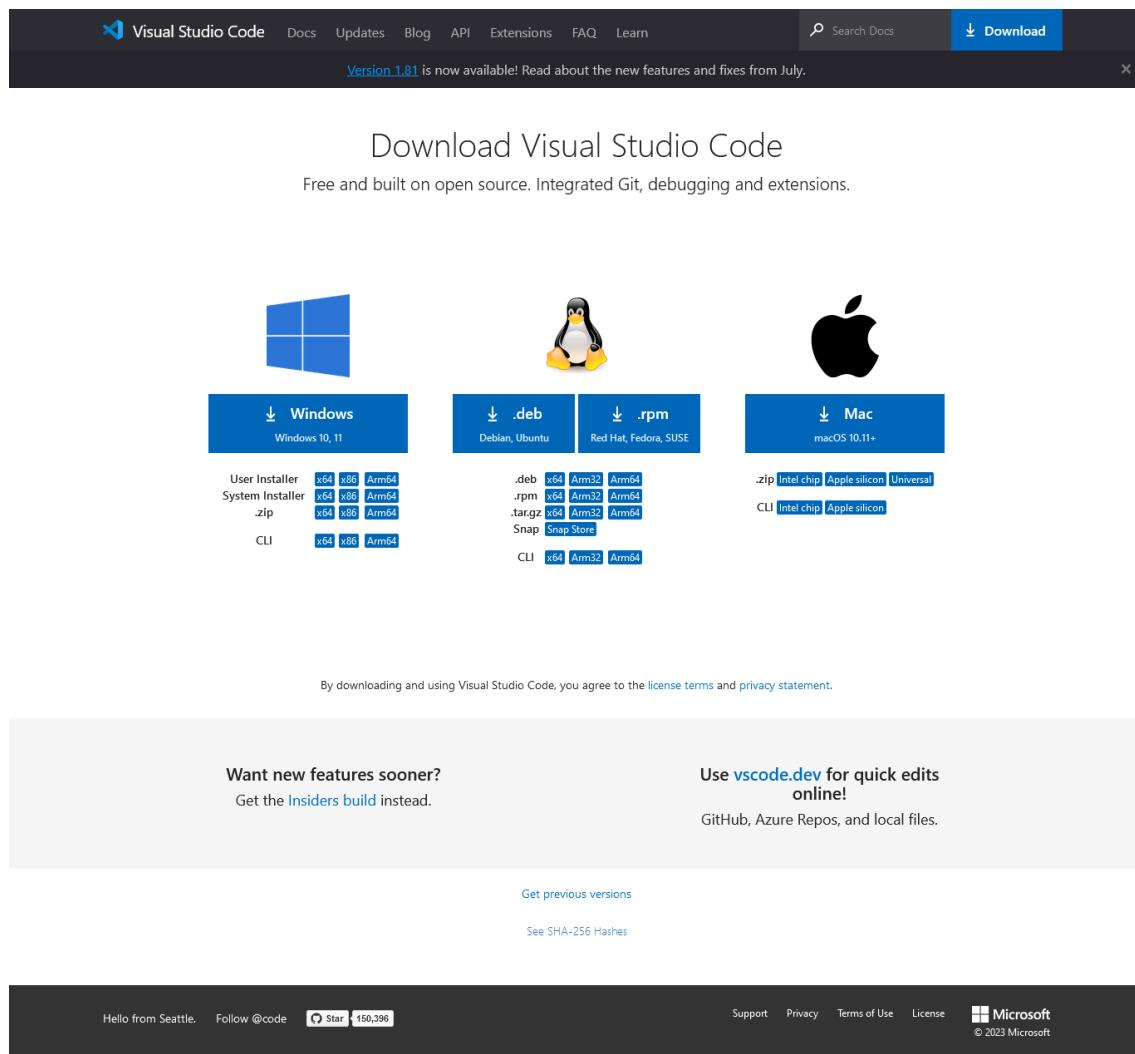
Figura 2 Página de XAMPP

The screenshot shows the Apache Friends XAMPP website. At the top, there's a dark blue header with links for 'Apache Friends', 'Descargar', 'Alojamiento', 'Comunidad', and 'Acerca de'. A search bar and a language dropdown set to 'ES' are also at the top. On the right side of the header is a vertical list of language options: DE, EN (selected), ES, FR, HU, IT, JP, PL, PT BR, RO, RU, TR, UR, ZH CN, and ZH TW. The main content area features a large 'XAMPP Apache + MariaDB + PHP + F' logo. Below it is a section titled '¿Qué es XAMPP?' with a brief description. To the right is a large orange square icon with a play button and the word 'XAMPP' below it. Below this are download links for Windows (8.2.4), Linux (8.2.4), and OS X (8.2.4). A green arrow-shaped button on the left says 'Descargar' and 'Pulsa aquí para otras versiones'. A central box announces the 'New XAMPP release 8.2.4, 8.1.17 and 8.0.28'. The footer contains social media links (Twitter, Facebook), copyright information ('Copyright (c) 2023, Apache Friends'), and links to 'Blog', 'Política de privacidad', and 'CDN proporcionado por fastly'.

Fuente: (Macas, 2023)

El IDE Microsoft Visual Studio Code para la gestión de los códigos puede descargarse de su página oficial <https://code.visualstudio.com/download>, la cual puede verse en la Figura 3.

Figura 3 Vista de la página del instalador del Visual Studio



Fuente: (Macas, 2023)

## De hardware

De los softwares necesarios utilizados, el que más recursos de hardware necesita es el Microsoft Visual Studio Code.

Para su instalación los requerimientos mínimos son:

- Requiere 1 GB de RAM (1,5 GB si se ejecuta en una máquina virtual)
- 1 GB de espacio disponible en el disco duro.
- Resolución de pantalla de 1024 por 768 o superior.

De ahí que estos sean los requerimientos mínimos de hardware que necesite nuestra aplicación.

## Instalación de la aplicación en Symphony

Symfony es uno de los mejores frameworks de PHP por su flexibilidad y que requiere menos tiempo de desarrollo. Se utiliza para crear software para los usuarios según sus necesidades.

Luego de instalados los softwares descritos en los requerimientos de software, para la instalación de la aplicación Symfony en cuestión dentro del servidor web, deben seguirse los siguientes pasos.

1. Obtención y ubicación de la solución.
2. Ejecución de los servicios de Apache y MySQL
3. Actualización de componentes
4. Creación de la base de datos
5. Inicializar el servidor

A continuación describiremos cada uno de estos pasos

### Obtención y ubicación de la solución.

Esta deberá ponerse en la carpeta **htdocs** del servidor web. Es decir, para el sistema operativo Windows, quedará en **C:\xampp\htdocs**.

La estructura de carpeta quedará como se muestra en la siguiente figura. (Ver Figura 4)

Figura 4 Estructura de carpeta de la aplicación

Nombre	Fecha de modificación	Tipo	Tamaño
📁 .idea	01/07/2024 21:30	Carpeta de archivos	
📁 .vs	01/07/2024 21:30	Carpeta de archivos	
📁 assets	01/07/2024 21:30	Carpeta de archivos	
📁 bin	25/10/2023 8:33	Carpeta de archivos	
📁 config	01/07/2024 21:30	Carpeta de archivos	
📁 migrations	25/10/2023 8:33	Carpeta de archivos	
📁 node_modules	01/07/2024 21:33	Carpeta de archivos	
📁 public	01/07/2024 21:36	Carpeta de archivos	
📁 src	01/07/2024 21:36	Carpeta de archivos	
📁 temp	22/05/2023 22:59	Carpeta de archivos	
📁 templates	01/07/2024 21:36	Carpeta de archivos	
📁 tests	01/07/2024 21:36	Carpeta de archivos	
📁 translations	01/07/2024 21:36	Carpeta de archivos	
📁 var	01/07/2024 21:37	Carpeta de archivos	
📁 vendor	02/07/2024 8:41	Carpeta de archivos	
📄 .env	05/09/2023 22:03	Archivo ENV	2 KB
📄 .htaccess	06/12/2023 21:24	Archivo HTACCESS	1 KB
📄 .user.ini	01/10/2023 15:29	Opciones de confi...	1 KB
📄 composer.json	02/07/2024 8:41	JSON File	4 KB
📄 composer.lock	02/07/2024 8:41	Archivo LOCK	349 KB
📄 composer.phar	20/01/2023 8:30	Archivo PHAR	2.154 KB
📄 error_log	10/12/2023 21:54	Archivo	469 KB
📄 index.php	17/09/2023 11:01	Archivo de origen ...	60 KB
📄 info.php	15/09/2023 16:50	Archivo de origen ...	1 KB
📄 package.json	16/01/2024 12:22	JSON File	1 KB
📄 package-lock.json	17/08/2023 21:00	JSON File	1.042 KB
📄 phpunit.xml.dist	20/01/2023 8:30	Archivo DIST	2 KB
📄 README.md	19/06/2023 14:11	Archivo de origen ...	1 KB
📄 symfony.lock	20/01/2023 8:30	Archivo LOCK	16 KB
📄 xperienc_uml.sql	05/09/2023 10:40	Archivo de origen ...	27 KB

Fuente: (Macas, 2023)

### Descripción de la estructura de archivos

A continuación, explicaremos para qué se emplean cada uno de los principales directorios.

- bin: Dentro de este directorio podemos ejecutar los correspondientes comandos de Symfony por consola para la ejecución de crear nuevos controladores, entidades tareas con bases de datos etc.

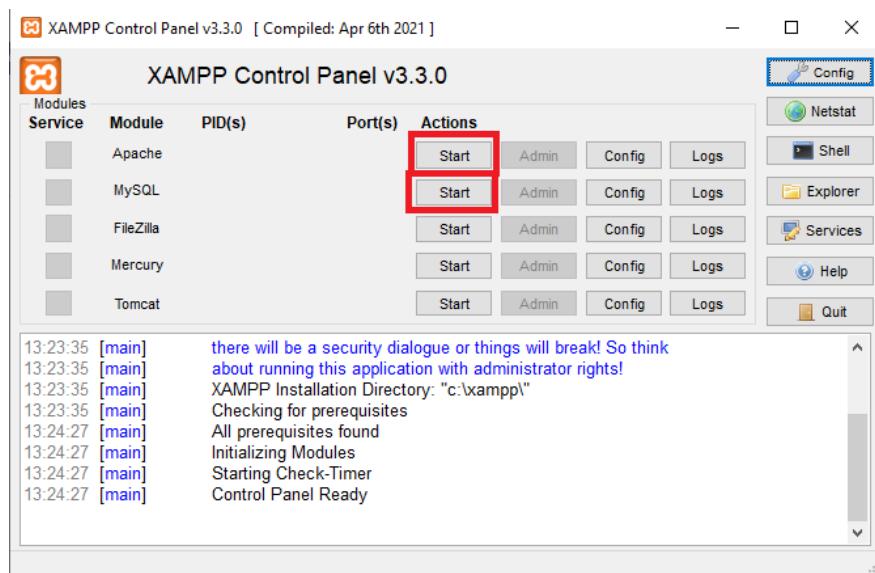
- config: Se guardan las rutas de nuestra aplicación en el archivo *routes.yaml* y los servicios en el archivo *services.yaml*
- public: En este directorio es donde se ve el contenido de la aplicación lo ideal es tener aquí guardada toda la parte de archivos correspondientes en cuanto css, javascript imágenes etc...
- src: Dentro de esta carpeta es donde se guardan todo aquello relacionado con lo que hacemos mediante consola o directamente como controladores, repositorios, entidades formularios etc...
- templates: Se almacenan las plantillas correspondientes que se son llamadas desde el controlador para la visualización de resultados usando el lenguaje plantillas twig.
- vendor: Este directorio es el principal de toda la aplicación de Symfony por que es donde está toda la configuración del framework por defecto, junto con todos los paquetes que se van emplear.

En la raíz del directorios, cabe destacar la existencia de dos ficheros muy importantes para el funcionamiento del framework, tal como son el *composer.json* y el archivo *.env*. El archivo *composer.json* nos va declarando todos los componentes que se van a emplear en la solución. Mientras que el archivo *.env* nos indica el entorno de desarrollo o producción del framework junto con la configuración de la base de datos.

### [Ejecución de los servicios de Apache y MySQL](#)

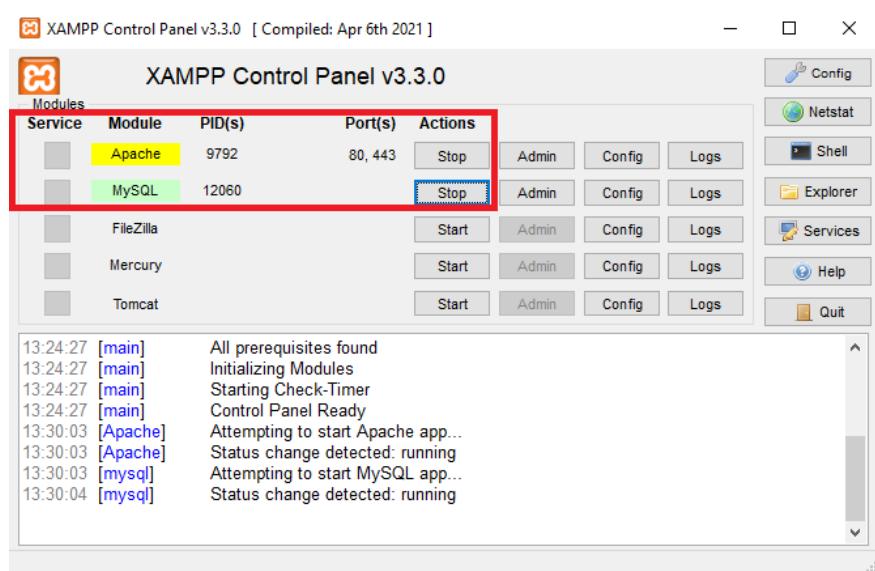
Una vez instalado el servidor XAMPP, se tiene varios servicios listos para correr que complementan el código de nuestra aplicación. En nuestro caso, necesitaremos de los servicios de Apache y MySQL. Estos servicios lo activaremos a través del panel de control de XAMPP.

Figura 5 Panel de Control de XAMPP



Una vez inicializados los servicios se mostrará el panel de control como se muestra en la Figura 6

Figura 6 Servicios Apache y MySQL inicializados en panel de control de XAMPP



Fuente: (Macas, 2023)

### Actualización de componentes

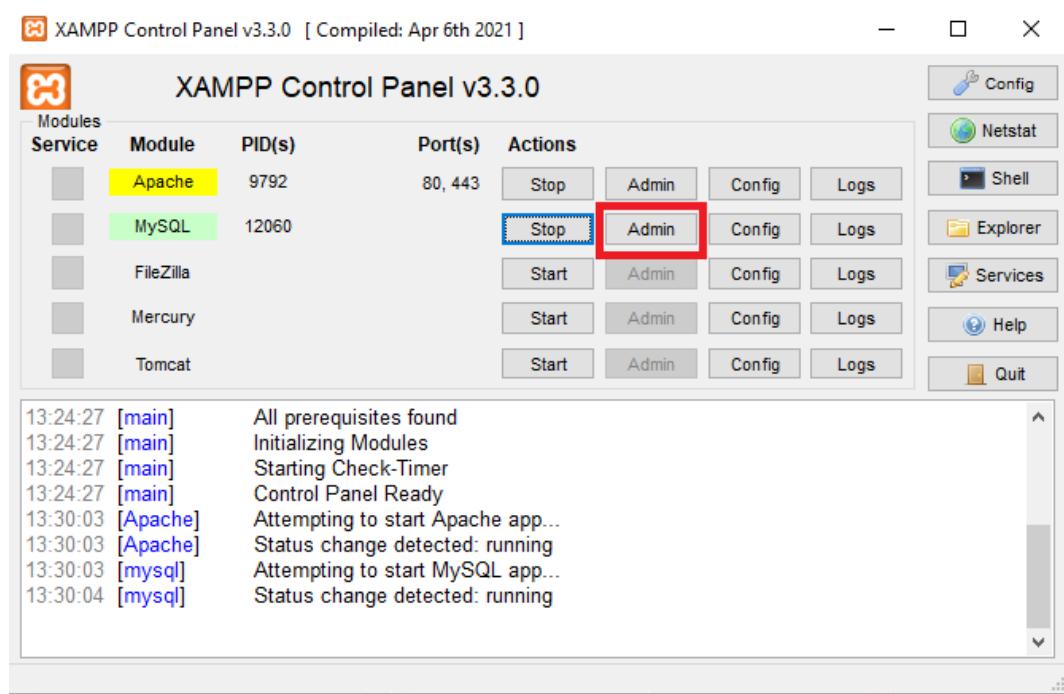
La aplicación está formada con una serie de componentes que son necesarios para su funcionamiento. Para realizar la actualización de los componentes se debe borrar la carpeta *vendor* y ejecutar el comando *composer update*. Antes de ejecutar este comando debemos asegurarnos de que la versión de php que usaremos esté correcta. Para ello verificaremos con el comando *php -v* este valor y comprobaremos ese resultado con el mostrado en el fichero *composer.json*

Al ejecutarse el comando ***composer update***, la carpeta ***vendor*** se generará nuevamente con los componentes listos para ser empleados.

## Creación de la base de datos

Para la creación de la base de datos, en el servicio MySQL de XAMPP del panel de control, se presiona el botón Admin para ejecutar en phpMyAdmin, software embedido que permite la gestión de la base de datos, como puede verse en la Figura 7.

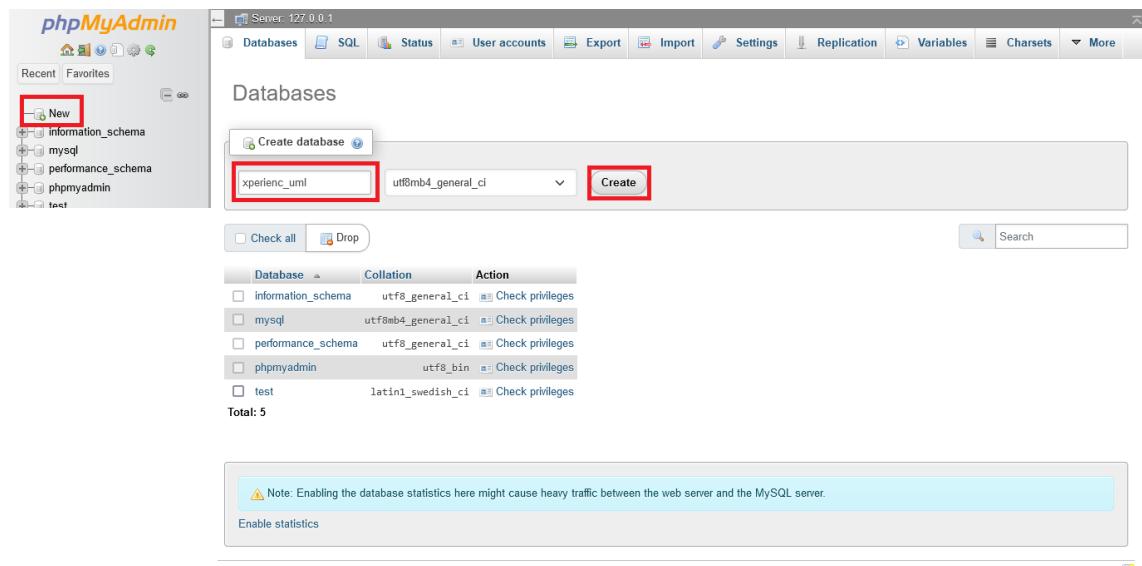
Figura 7 Botón Admin para abrir phpMyAdmin



Fuente: (Macas, 2023)

Una vez abierto del sistema de gestión de Bases de Datos, crearemos la base de datos con nombre ***xperienc\_uml*** desde el botón ***New***. (Ver Figura 8)

Figura 8 Vista de phpMyAdmin para crear la base de datos



Es importante verificar que para la base de datos se cuenta con el usuario *xperienc*. Este se crea de la siguiente manera mediante comandos SQL.

```
CREATE USER 'xperienc'@'localhost' IDENTIFIED BY 'XXXXXXXX';
```

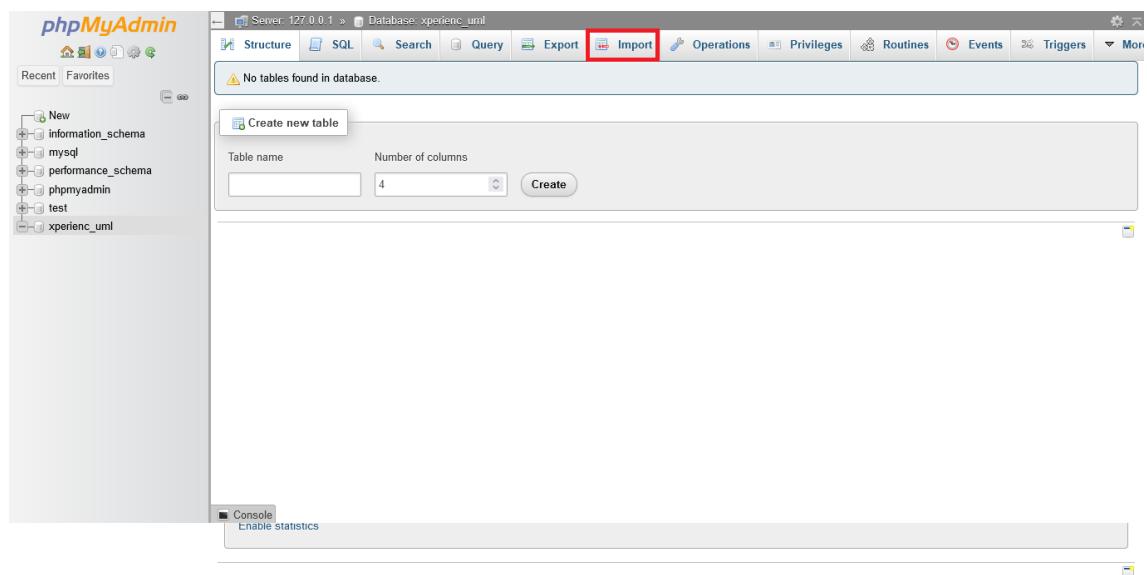
```
GRANT ALL PRIVILEGES ON * . * TO 'xperienc'@'localhost';
```

Aquí '**XXXXXXXX**', se refiere a la contraseña del usuario *'xperienc'*.

Otra parte importante a chequear es el fichero *.env*, también ubicado en la raíz de la solución. En este fichero se tiene la acreditación de la base de datos, que debe coincidir con el usuario y la contraseña que se registra en la creación de la base de datos.

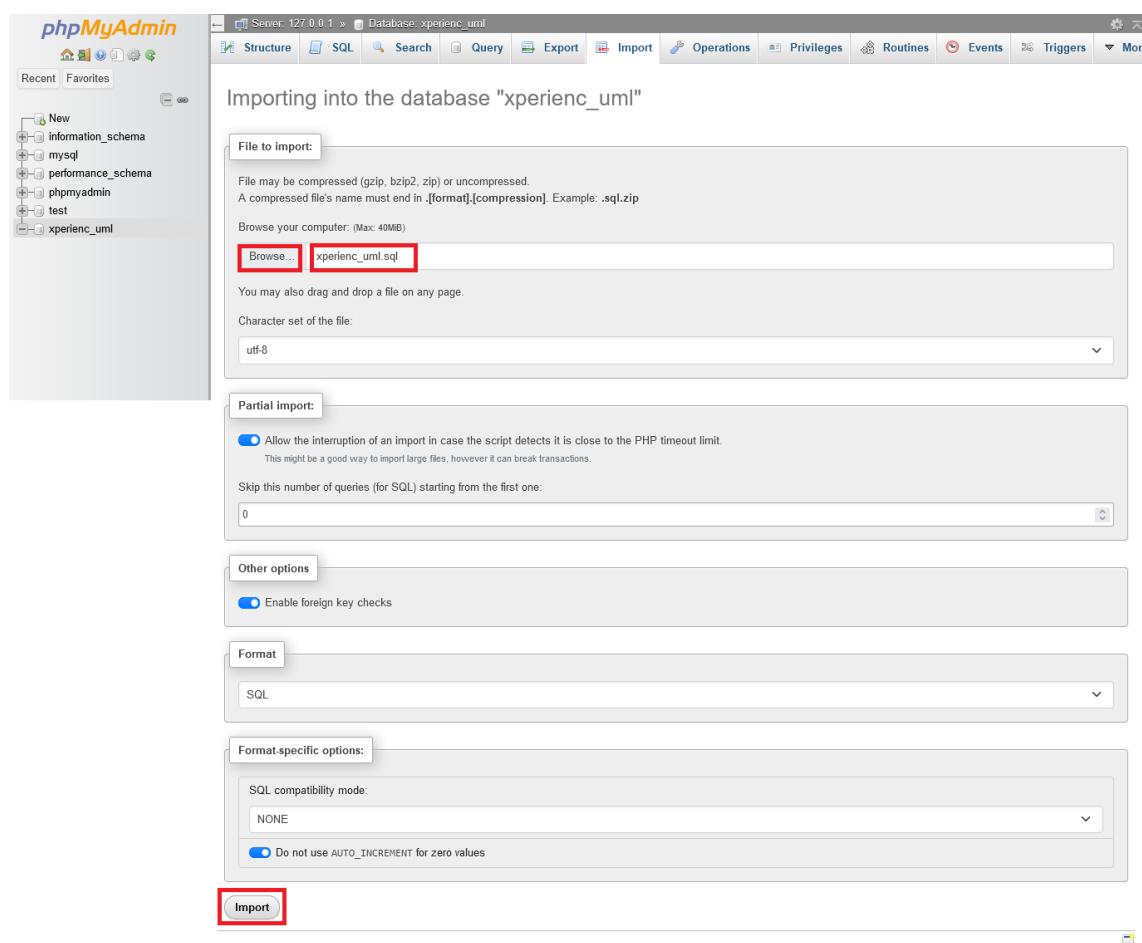
Después se procederá a importar las tablas y algunos datos a través de la pestaña importar (. Ver Figura 9). Estos datos se cargarán desde el fichero *xperienc\_uml.sql*, ubicado en la raíz del directorio de la solución.

Figura 9 Pestaña para Importar datos



Luego debe abrirse base de datos **xperienc\_uml**, e ir a la opción **Import**. En esta opción se debe buscar a través del botón **Browse** el fichero *xperienc\_uml.sql* para finalmente seleccionar la opción **Import** al final de la página y ejecutar la operación. (Ver Figura 10)

Figura 10 Importar datos desde un fichero



Esto creará las tablas de nuestra aplicación e insertará algunos datos a esas tablas. A continuación mostramos las tablas

Tabla 1 Tablas de la base de datos

Tablas	Descripción
T-01	user
T-02	course
T-03	course_user
T-04	pclass
T-05	stud_pclass
T-06	activity
T-07	nta
T-08	datatype
T-09	Atribute

---

T-10	operation
T-11	pclass_atribute
T-12	pclass_operation
T-13	stud_pclass_atribute
T-14	stud_pclass_operation
T-15	success_code
T-16	success_code_codeline
T-17	codeline
T-18	eval_code
T-19	eval_code_codeline
T-20	model_diagram_test
T-21	model_diagram_success
T-22	statement
T-23	doctrine_migration_versions

---

Fuente: elaboración propia.

## Inicializar el servidor

Una vez llegado a este paso solo queda echar a andar la aplicación en el servidor. Para ello debe ejecutarse el comando:

***php -S localhost:8080***

En este punto se puede abrir el proyecto en el navegador y ver la aplicación funcionando.

La dirección a la que podrá accederse para ver nuestra aplicación ejecutándose es:

***http://localhost:8080***

Hasta aquí de forma genérica los pasos a seguir para la creación de una aplicación web mediante Symfony.

A continuación describiremos nuestra aplicación.

## Diseño de la aplicación

A continuación describiremos la aplicación así como los pasos llevados a cabo en su implementación. Primeramente identificaremos los actores (usuarios) de la misma. Luego los módulos de dicha aplicación.

### Actores de la aplicación

La identificación de los actores es una de las primeras tareas al iniciar un proyecto. Para esta aplicación se han definido como actores:

- Usuarios con rol Docente
- Usuarios con rol Estudiante.

Los usuarios con rol Docente serán encargados de la gestión del sitio, es decir gestionar los usuarios, cursos y actividades.

Los usuarios con rol Estudiante solo resolverán las actividades de los cursos de los cuales son miembros y verán esas actividades realizadas por ellos.

### Módulos de la aplicación

Las funciones principales del aplicativo se encuentran agrupadas en los siguientes módulos, los cuales están disponibles en la sección web:

- Módulo de Gestión de Autenticación
- Módulo de Gestión de Usuarios
- Módulo de Gestión de Cursos
- Módulo de Gestión de Actividades
- Módulo de Reportes

A continuación describiremos cada uno de estos módulos.

#### Autenticación

Este módulo permite identificar y verificar los valores o campos proporcionados para garantizar un acceso correcto a la aplicación. En un escenario inicial, se proporcionará un administrador

por defecto, que posteriormente podrá ser actualizado con los datos correspondientes del usuario final. También se dará la posibilidad de realizar el autoregistro de un usuario, al cual se le asignará el rol Estudiante.

## Usuarios

Este módulo permite la autenticación de usuarios mediante la asignación de roles, lo que garantiza un enrutamiento adecuado de las vistas. Solo los usuarios administradores tienen acceso y control en este módulo, lo que reduce la posibilidad de accesos no autorizados a la aplicación, con el propósito de obtener soluciones de ejercicios o actividades programadas de manera ilícita. Para el registro de un usuario se tiene en cuenta el correo electrónico, rol que se le va a asignar (es decir, Estudiante o Docente), clave o contraseña y nombre de usuario. Para la gestión de usuarios las acciones que se realizan son Ver y Editar los datos de un usuario. Cuando se estén mostrando los datos de un usuario se podrá además eliminar el usuario que este visualizándose en ese momento. Para los usuarios con rol de Estudiante, solo podrán visualizar sus datos registrados en la aplicación, sin posibilidad alguna de modificación, edición o eliminación. Una vez registrados los usuarios, podrán comenzar a realizar sus actividades académicas asignadas.

## Cursos

En este módulo se lleva a cabo la gestión de cursos, los cuales se definen mediante atributos como el nombre, la fecha de inicio y la fecha de culminación, así como los miembros participantes del mismo. El propósito de gestionar cursos es categorizar la información en bloques, permitiendo que un determinado número de alumnos o estudiantes tengan una base conceptual y puedan aprender a su propio ritmo de manera proporcional. Cuando se crea un curso se incorporan los estudiantes que van a ser miembros del mismo, por tanto, para gestionar los cursos, debe haberse gestionado antes los usuarios. Este módulo es solo manejado por usuarios con rol Docente. Las acciones permitidas para la gestión de los cursos son Ver y Editar. Cuando se está visualizando un curso en específico, se dará la posibilidad también de

Eliminar el curso en cuestión. Por otra parte cuando se está visualizándose un curso en específico se dará la posibilidad de gestionar las actividades relacionadas con el curso.

## Actividades

Este módulo ofrece la posibilidad de crear una o varias actividades, donde se pueden definir diversas propiedades. Entre estas propiedades se incluyen el título principal de la actividad, una descripción detallada del ejercicio o encabezado, la solución técnica proporcionada para la resolución de la actividad y el mensaje que se mostrará en caso de realizar la actividad de forma satisfactoria, lo que permite la personalización de cada actividad. De esta manera, se brinda una estructura clara y organizada para cada actividad, permitiendo su correcta realización y evaluación. Como se había mencionado antes, a este módulo se llega a través del módulo de Gestión de Cursos.

## Reportes

En este módulo, se almacenarán los datos del xAPI a través de un Learning Record Store (LRS), lo que nos permitirá tener un control adecuado de las actividades realizadas por los usuarios estudiantes en el sistema. Estos datos recopilados incluirán información relevante sobre el progreso, el rendimiento y el comportamiento de los estudiantes en relación con las actividades y los recursos del sistema. Esta funcionalidad nos brindará una visión detallada de la interacción de los estudiantes con la plataforma y nos ayudará a realizar un seguimiento y una evaluación efectiva de su aprendizaje.

## Declaraciones

En este módulo, se encuentran los verbos que describen la actividad de los Estudiantes, utilizados por el módulo de Reportes para calificar el estado de cada actividad implementada por los mismos.

Los verbos que se implementan son

- Launched
- Initialized

- Passed
- Waived
- Failed
- Terminated
- Completed

Este último se genera automáticamente cuando el estudiante culmina la actividad y se evalúa satisfactoriamente.

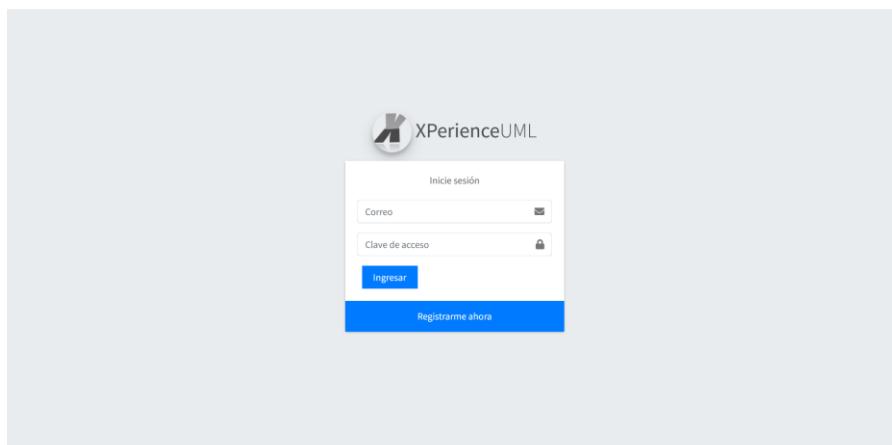
## Creación de Diagramas

En este módulo, los usuarios tienen la capacidad de crear diagramas UML. Tanto los estudiantes como los administradores pueden resolver las actividades mediante componentes interactivos que se pueden mover dentro de la ventana del navegador. Este entorno de trabajo proporciona una interfaz intuitiva que permite a los usuarios diseñar y modificar diagramas UML de manera eficiente. Los componentes interactivos facilitan la manipulación de los elementos del diagrama, lo que brinda flexibilidad y facilidad de uso a la hora de resolver las actividades propuestas.

## Descripción e implementación de la aplicación

La aplicación está concebida para ser usada por usuarios registrados. A continuación se muestra la pantalla inicial. (Ver Figura 11).

*Figura 11 Pantalla inicial del sistema*



Fuente: (Macas, 2023)

## Autenticación

Para iniciar en la aplicación hemos destinado un usuario con rol Docente, para la gestión de todo lo necesario para el uso y explotación de la aplicación.

Figura 12 Código del frontend de la autenticación

```
52. <div class="card">
53.   <div class="card-body login-card-body">
54.     <p class="login-box-msg">Inicie sesión</p>
55.     <div class="input-group mb-3">
56.       <input type="email" value="{{ last_username }}" name="email"
57. id="inputEmail" placeholder="Correo" class="form-control" autocomplete="email"
58. required autofocus>
59.       <div class="input-group-append">
60.         <div class="input-group-text">
61.           <span class="fas fa-envelope"></span>
62.         </div>
63.       </div>
64.     </div>
65.     <div class="input-group mb-3">
66.       <input type="password" name="password" id="inputPassword"
67. placeholder="Clave de acceso" class="form-control" autocomplete="current-
68. password" required>
69.       <div class="input-group-append">
70.         <div class="input-group-text">
71.           <span class="fas fa-lock"></span>
72.         </div>
73.       </div>
74.     </div>
75.     <input type="hidden" name="_csrf_token" value="{{
76. csrf_token('authenticate') }}">
77.     <div class="col-4">
78.       <button type="submit" class="btn btn-primary btn-block btn-
79. flat">Ingresar</button>
80.     </div>
81.   </div>
82.   <p class="mb-0">
83.     <button type="submit" class="btn btn-primary btn-block btn-flat"><a
84. href="{{ path('app_register') }}" class="btn btn-primary btn-block btn-
85. flat">Registrarme ahora</a></button>
86.   </p>
87. </div>
88. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 13 Código del backend de la autenticación. Redireccionamiento luego de ingresar un usuario al sistema.

```
92. public function onAuthenticationSuccess(Request $request, TokenInterface $token,
93. string $providerKey)
94. {
95.     if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {
96.         return new RedirectResponse($targetPath);
97.     }
98.     $userRole = $token->getUser()->getRoles()[0];
99.     switch ($userRole) {
100.         case 'ROLE_ADMIN':
101.             return new RedirectResponse($this->urlGenerator-
102. >generate('app_course_index'));
103.         case 'ROLE_USER':
104.             return new RedirectResponse($this->urlGenerator-
105. >generate('app_model_diagram_test_index2', ['id'=>$token->getUser()->getId()]));
106.         default:
107.             throw new \Exception("Rol desconocido: ".$userRole);
108.     }
109. }
110. return new RedirectResponse($this->urlGenerator->generate('app_begin'));
111. throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
112. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la autenticación se emplea el fichero *AppCustomAuthenticator.php*, que es una clase en Symfony que se utiliza para implementar un mecanismo de autenticación personalizado. En esta clase la tabla de credenciales no se define directamente, sino que se utiliza Doctrine para recuperar las credenciales de una entidad personalizada, con las que se genera un token de autenticación en caso de éxito. Esto permite extender la funcionalidad de autenticación predeterminada de Symfony para adaptarla a las necesidades específicas de la aplicación.

En código mostrado Figura 13 se ejecuta luego de ingresado el usuario representado por un email y la contraseña. En este código inicialmente se verifica si hay una ruta guardada para redirigir al usuario después de iniciar sesión. Si la encuentra, la función devuelve una respuesta especial que indica al navegador que mueva al usuario a esa ruta específica. De lo contrario, se obtiene el rol. Luego en dependencia del rol del usuario se ejecutan solo una de las opciones siguiente:

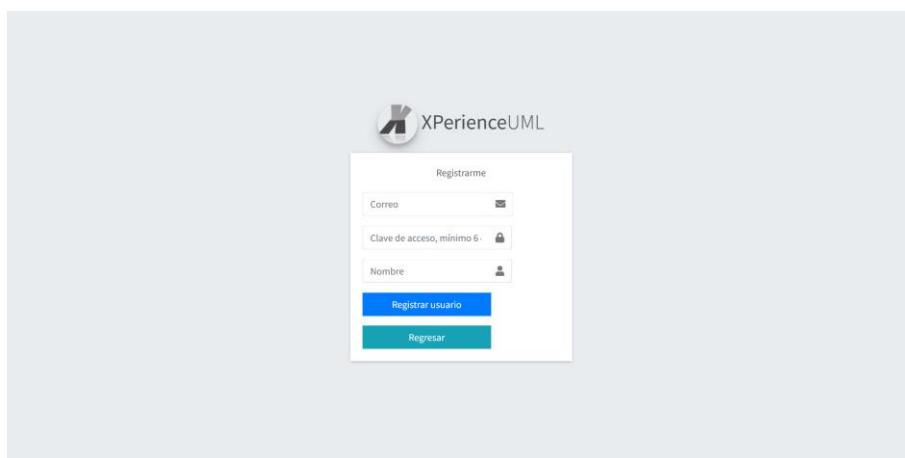
- Se le redirecciona a la ruta *app\_course\_index* , la cual se corresponde con la página del listado de cursos la cual es una página específica para usuarios administradores, es decir, para usuarios con rol Docente (visto internamente en código como **ROLE\_ADMIN**),

- Se le redirecciona a la ruta *app\_model\_diagram\_test\_index2*, la cual se corresponde con la página del listado de soluciones del estudiante en caso de que el rol identificado sea Estudiante (visto internamente como ROLE\_USER). Además, se agrega un parámetro id a la URL de redireccionamiento con el valor del identificador del usuario obtenido del token (*\$token->getUser()->getId()*). Esto permite acceder a información específica del usuario en la página de destino.
- En caso de que ninguno de los roles mencionados sea identificado, entonces se lanza una excepción indicando que se encontró un rol desconocido y se redirecciona al usuario a la página de inicio, la cual está indicada en la ruta *app\_begin*.

Como se ve en la Figura 11, los usuarios pueden acceder al sistema ingresando un correo y una contraseña previamente registrada, o realizando un autoregistro mediante el botón **Registrarme ahora**.

Si presiona este botón se mostrará la pantalla que se ve en la Figura 14, donde el usuario proporcionará un correo, una clave de al menos 6 caracteres y un nombre. El correo que proporcione no podrá estar registrado en el sistema, de lo contrario se mostrará un mensaje y no se procederá al registro del usuario. Otra causa por la que puede no realizarse el registro del usuario, es por no poner una clave de forma correcta o no proporcionar un nombre de usuario.

*Figura 14 Autoregistro de un usuario*



Fuente: (Macas, 2023), pantalla de la aplicación.

Figura 15 Código del frontend de la autenticación. Autoregistro.

```
41. {{ form_start(registrationForm) }}
42. <div class="input-group mb-3">
43.   {{ form_row(registrationForm.email, {'attr': {'class': 'form-
44. control', 'autocomplete': 'off'}})}}
45.   <div class="input-group-append">
46.     <div class="input-group-text">
47.       <span class="fas fa-envelope"></span>
48.     </div>
49.   </div>
50. </div>
51. <div class="input-group mb-3">
52.   {{ form_row(registrationForm.plainPassword, {'attr': {'class': 'form-control',
53. 'autocomplete': 'new-password'}})}}
54.   <div class="input-group-append">
55.     <div class="input-group-text">
56.       <span class="fas fa-lock"></span>
57.     </div>
58.   </div>
59. </div>
60. <div class="input-group mb-3">
61.   {{ form_row(registrationForm.name, {'attr': {'class': 'form-control'}})}}
62.   <div class="input-group-append">
63.     <div class="input-group-text">
64.       <span class="fas fa-user"></span>
65.     </div>
66.   </div>
67. </div>
68. <div class="modal" id="confirmContra" name="confirmContra" style="display:none;">
69.   <div class="modal-dialog" role="document">
70.     <div class="modal-content">
71.       <div class="modal-header">
72.         <h5 class="modal-title" id="confirmModalLabel">Verificacion de
73. datos</h5>
74.         <button type="button" class="close" data-dismiss="modal" aria-
75. label="Close">
76.           <span aria-hidden="true">&times;</span>
77.         </button>
78.       </div>
79.       <div class="modal-body" id="modalResult" name="modalResult">
80.       </div>
81.       <div class="modal-footer">
82.         <button type="button" class="btn btn-info" data-dismiss="modal"
83. id="cerrar" onclick="ocultarDiv()">Verificar</button>
84.       </div>
85.     </div>
86.   </div>
87. </div>
88. <div class="row">
89.   <div class="col-8">
90.     <button type="submit" class="btn btn-primary btn-block btn-flat"
91. onclick="mostrarDiv()">Registrar usuario</button>
92.   </div>
93. </div>
94. {{ form_end(registrationForm) }}
95. <div class="row">
96.   <div class="col-8">
97.     <a href="{{ path('app_login') }}><button class="btn btn-info btn-block btn-
98. flat">Regresar</button></a>
99.   </div>
100. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 16 Código del backend de la autenticación. Botón Registrar usuario.

```
30. /**
31. * @Route("/register", name="app_register")
32. */
33. public function register(Request $request,
34.                         UserPasswordEncoderInterface $passwordEncoder,
35.                         GuardAuthenticatorHandler $guardHandler,
36.                         AppCustomAuthenticator $authenticator): Response
37. {
38.     $form = $this->createForm(RegistrationFormType::class);
39.     $form->handleRequest($request);
40.
41.
42.     if ($form->isSubmitted() && $form->isValid())
43.     {
44.         $user = $form->getData();
45.
46.         // Asignar el rol al usuario
47.         $user->setRoles(['ROLE_USER']);
48.
49.         // encode the plain password
50.         $user->setPassword(
51.             $passwordEncoder->encodePassword(
52.                 $user,
53.                 $form->get('plainPassword')->getData()
54.             )
55.         );
56.         $entityManager = $this->getDoctrine()->getManager();
57.         $users = $entityManager->getRepository(User::class);
58.         if ($users->findBy(['email' => $user->getEmail()]))
59.         {
60.             $this->addFlash('error', 'Email ya registrado !!!');
61.             return $this->render('registration/register.html.twig', [
62.                 'registrationForm' => $form->createView(),
63.                 'error' => 'Email ya registrado. Rectifique y vuelva a
64. intentarlo !!!',
65.             ]);
66.         }
67.         $entityManager = $this->getDoctrine()->getManager();
68.         $entityManager->persist($user);
69.         $entityManager->flush();
70.         // do anything else you need here, like send an email
71.
72.         // Autenticar al usuario después del registro
73.         return $guardHandler->authenticateUserAndHandleSuccess(
74.             $user,
75.             $request,
76.             $authenticator,
77.             'main' // firewall name in security.yaml
78.         );
79.     }
80.
81.     return $this->render('registration/register.html.twig', [
82.         'registrationForm' => $form->createView(),
83.         'error' => '',
84.     ]);
85. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para el autoregistro de un usuario se emplea el fichero *RegistrationController.php*, que es una clase controladora de Symfony. Este fichero consta de una sola función (*register*), que es una función que gestiona el registro de un nuevo usuario, donde se define una ruta para la acción de

registro. Cualquier solicitud HTTP a la URL `/register` será manejada por esta función. La función `register` tiene los siguientes parámetros:

- Request \$request: Objeto que representa la solicitud HTTP.
- UserPasswordEncoderInterface \$passwordEncoder: Servicio para encriptar contraseñas.
- GuardAuthenticatorHandler \$guardHandler: Servicio para manejar la autenticación.
- AppCustomAuthenticator \$authenticator: Autentificador personalizado. La función devuelve un objeto Response.

Inicialmente se crea un formulario de registro basado en la clase `RegistrationFormType` y procesa los datos enviados en la solicitud HTTP. Se comprueba si el formulario ha sido enviado y si los datos son válidos. Se obtienen los datos del formulario y los asigna a un objeto `User`. Asigna el rol de usuario al nuevo usuario (ROLE\_USER). Se encripta la contraseña del usuario utilizando el servicio `$passwordEncoder` que estableció en los parámetros. Se comprueba si el email del usuario ya existe en la base de datos. Si existe, muestra un mensaje de error. Se persiste el usuario en la base de datos, guardándose los datos. Luego se autentica al usuario recién registrado y redirige al usuario a la página de inicio de sesión. Si el formulario no es válido o si hay errores se renderiza el formulario de registro. En resumen, el código crea un formulario de registro, valida los datos, encripta la contraseña, verifica la existencia del email, persiste el usuario en la base de datos y finalmente autentica al usuario.

Es importante señalar que nunca se debe almacenar la contraseña o clave de un usuario directamente en la base de datos. En su lugar, se almacena una versión codificada (encriptada) de la contraseña. De esta forma, si alguien tuviera acceso a tu base de datos, no podría ver las contraseñas reales de los usuarios.

A continuación mostramos la pantalla luego de haberse autenticado un Docente

Figura 17 Vista Inicial al registrarse un usuario con rol Docente. Listado de Curso

The screenshot shows the XperienceUML application's main dashboard. On the left, there is a sidebar with the following navigation options:

- Gestión de cursos
  - Cursos (selected)
  - Usuarios
  - Actividades del Curso
  - Soluciones de Actividades
- Reportes
  - Experiencia de aprendizaje
- Estudiantes
  - Actividades Resueltas
  - Responder actividades

In the center, the main content area is titled "Lista de Cursos". It displays a table with three rows of course information:

Nombre	Fecha Inicio	Fecha Fin	Acciones
El mundo de la programación orientada a objetos	2024-06-01	2024-06-30	
Herencia y Polimorfismo en Programación Orientada a Objetos	2024-06-01	2024-06-30	
Introducción a la Programación Orientada a Objetos	2024-01-01	2024-08-31	

At the bottom of the main content area, there is a footer with the text "Mostrando página 1 de 1" and navigation buttons for "Anterior" and "Próximo". Below the footer, there are copyright and version information:

Copyright © 2024 XperienceUML. Todos los derechos reservados. Version 1.0

Fuente: (Macas, 2023)

Cuando veamos el apartado de cursos analizaremos brevemente el código de esta página.

## Gestión de usuarios

Para la correcta funcionalidad del sitio, los docentes deberán primero gestionar los usuarios.

La gestión de usuarios se basa en la creación, modificación y eliminación de usuarios. Solo está permitido para usuarios con rol Docente.

El docente ingresa en la aplicación web y se autentica. Luego seleccionará la opción de Gestión de cursos, Usuarios, la que listará todos los usuarios registrados para el uso de la aplicación.

Figura 18 Listado de usuarios

Correo electrónico	Roles	Nombre	Acciones
admin@cmi5.com	Profesor	Administrador	
crisestudiante@utpl.com	Estudiante	Cristian Estudiante	
estudiante10@utpl.com	Estudiante	Estudiante 10	
estudiante11@utpl.com	Estudiante	Estudiante 11	
estudiante12@utpl.com	Estudiante	Estudiante 12	
estudiante13@utpl.com	Estudiante	Estudiante 13	
estudiante14@utpl.com	Estudiante	Estudiante 14	
estudiante15@utpl.com	Estudiante	Estudiante 15	
estudiante16@utpl.com	Estudiante	Estudiante 16	
estudiante17@utpl.com	Estudiante	Estudiante 17	

Fuente: (Macas, 2023)

La información que se listarán específicamente serán el correo electrónico, el rol y el nombre de cada usuario registrado.

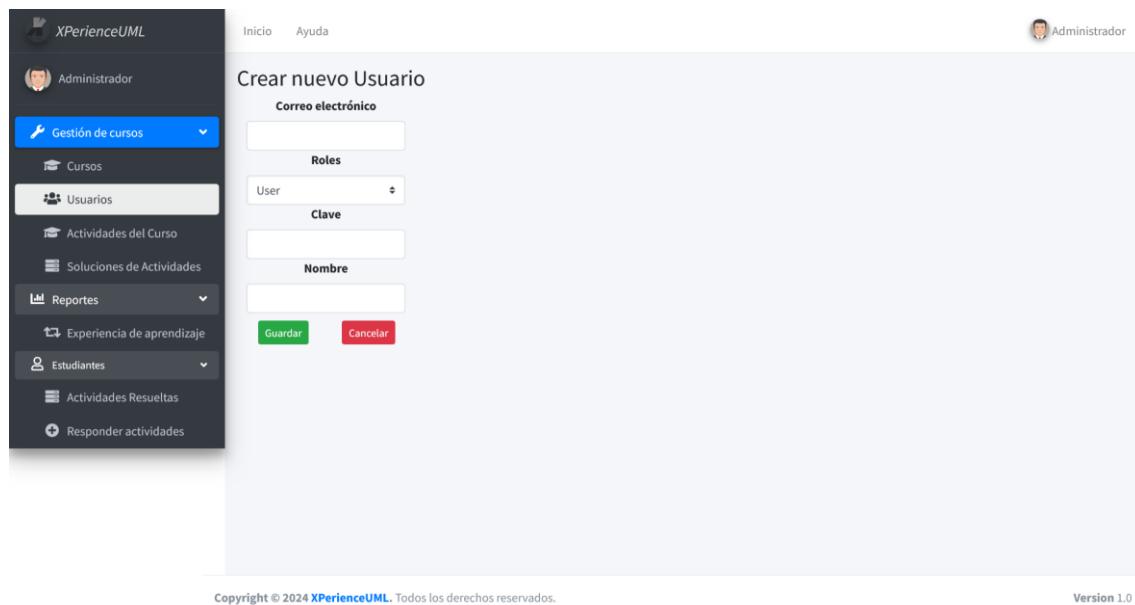
Figura 19 Código del frontend. Listado de usuarios.

```
29. <table id="example1" class="table table-bordered table-striped dataTable"
30. role="grid" aria-describedby="example1_info">
31.   <thead>
32.     <tr>
33.       <th>Correo electr&acute;nico</th>
34.       <th>Roles</th>
35.       <th>Nombre</th>
36.       <th>Acciones</th>
37.     </tr>
38.   </thead>
39.   <tbody>
40.     {% for user in users %}
41.       <tr>
42.         <td>{{ user.email }}</td>
43.         <td>{{ user.roles[0]=='ROLE_USER' ? 'Estudiante' : 'Profesor' }}</td>
44.         <td>{{ user.name }}</td>
45.         <td>
46.           <a href="{{ path('app_user_show', {'id': user.id}) }}"><i class="fa
47. fa-eye" title="Ver"></i></a>
48.           {% if is_granted('ROLE_ADMIN')%}
49.             {% if ( user.email != "admin@cmi5.com" ) %}
50.               <a href="{{ path('app_user_edit', {'id': user.id}) }}"><i
51. class="fa fa-edit" title="Editar"></i></a>
52.               {% if ( app.user.username == "admin@cmi5.com" ) %}
53.                 <a href="{{ path('app_user_delete', {'id': user.id}) }}"><i
54. class="fa fa-trash" title="Borrar"></i></a>
55.               {% endif %}
56.               {% endif %}
57.             {% endif %}
58.           </td>
59.         </tr>
60.     {% else %}
61.       <tr>
62.         <td colspan="6">no hay datos</td>
63.       </tr>
64.     {% endfor %}
65.   </tbody>
66. </table>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Mediante la opción *Agregar usuario*, podrá crear nuevos usuarios para el sistema. A continuación mostramos la vista de la página para crear un nuevo usuario.

Figura 20 Vista de creación de usuarios.



Fuente: (Macas, 2023)

Figura 21 Código del frontend de Crear nuevo Usuario.

```
2. {{ form_row(form.email) }}
3. {{ form_row(form.roles) }}
4. {{ form_row(form.password) }}
5. {{ form_row(form.name) }}
6. <div class="row" style="margin-top:10px">
7.   <div class="col-6">
8.     <button class="btn btn-success btn-sm">{{ button_label|default('Guardar') }}</button>
9.   </div>
10.  <div class="col-6">
11.    <button type="button" class="btn btn-danger btn-sm"
12.      onclick="mostrarDivUser()">Cancelar</button>
13.  </div>
14. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 22 Código del backend de la autenticación. Botón Guardar usuario.

```
36. /**
37. * @Route("/new", name="app_user_new", methods={"GET", "POST"})
38. */
39. public function new(Request $request, UserRepository $userRepository):
40. Response
41. {
42.     $user = new User();
43.     $form = $this->createForm(UserType::class, $user);
44.     $form->handleRequest($request);
45.
46.     if ($form->isSubmitted() && $form->isValid()) {
47.         /* código para codificar clave */
48.         $plainpwd = $user->getPassword();
49.         $encoded = $this->passwordEncoder->encodePassword($user,
50. $plainpwd);
51.         $user->setPassword($encoded);
52.         $userRepository->add($user);
53.         $this->addFlash("success", 'Insertado satisfactoriamente!!! ');
54.         $form = $this->createForm(UserType::class, $user);
55.         return $this->redirectToRoute('app_user_index', [],
56. Response::HTTP_SEE_OTHER);
57.     }
58.
59.     return $this->render('user/new.html.twig', [
60.         'user' => $user,
61.         'form' => $form->createView(),
62.     ]);
63. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la gestión de los usuarios desde el punto de vista de código se emplea el fichero *UserController.php*, es una clase controladora de Symfony. En esta clase se implementan los métodos para listar, crear, modificar y eliminar un usuario.

El código de la Figura 22 gestiona la creación de un nuevo usuario con esta clase controladora, donde se define una ruta */new* para esta acción, accesible tanto por métodos GET como POST. Los parámetros que recibe son:

- Request \$request: Objeto que representa la solicitud HTTP.
- UserRepository \$userRepository: Repositorio para gestionar usuarios.

Mientras que devuelve una respuesta HTTP.

En la implementación de la función inicialmente se crea un Objeto *User*. Luego se crea un formulario basado en la clase *UserType* y lo vincula al objeto *\$user*, para después procesar los datos enviados en la solicitud HTTP. Se comprueba si el formulario ha sido enviado y si los

datos son válidos. Entonces se codifica la contraseña. Esto es, obtener la contraseña sin codificar, utilizar el servicio `$passwordEncoder` para codificarla que se establece en el constructor de la clase y asignar la contraseña codificada al usuario. Seguido se utiliza el repositorio `userRepository` para persistir el usuario en la base de datos. Se muestra un mensaje de éxito. Se redirige a la ruta `app_user_index` después de guardar el usuario. Si el formulario no es válido o si ha ocurrido algún error se renderiza la plantilla `user/new.html.twig` pasando el objeto `$user` y el formulario como variables. En resumen, este código crea un formulario para crear un nuevo usuario, valida los datos, codifica la contraseña, guarda el usuario en la base de datos y redirige a una lista de usuarios.

Cuando se listan los usuarios hay un conjunto de acciones que son permitidas al usuario con Rol Docente, para realizar con cada uno de los usuarios a excepción del usuario **Administrador**. Estas son *Ver, Editar y Borrar*.

La opción Ver, visualiza el registro de un usuario. A continuación mostramos la página de visualización de datos de un usuario.

Figura 23 Visualización del registro de un usuario

The screenshot shows the XPerienceUML application interface. On the left is a sidebar with navigation links: 'Gestión de cursos' (selected), 'Cursos', 'Usuarios' (highlighted in blue), 'Actividades del Curso', 'Soluciones de Actividades', 'Reportes' (dropdown menu), 'Experiencia de aprendizaje', 'Estudiantes' (dropdown menu), 'Actividades Resueltas', and 'Responder actividades'. At the top right are 'Inicio' and 'Ayuda' buttons, and a profile picture for 'Administrador'. The main content area is titled 'Usuario' and displays the following information:

Correo	crisestudiante@utpl.com
Roles	Estudiante
Nombre	Cristian Estudiante

A blue 'Editar' button is located at the bottom left of the main content area. At the very bottom of the page, there is a copyright notice: 'Copyright © 2024 XPerienceUML. Todos los derechos reservados.' and 'Version 1.0'.

Fuente: (Macas, 2023)

Figura 24 Código del frontend de Ver información de un Usuario.

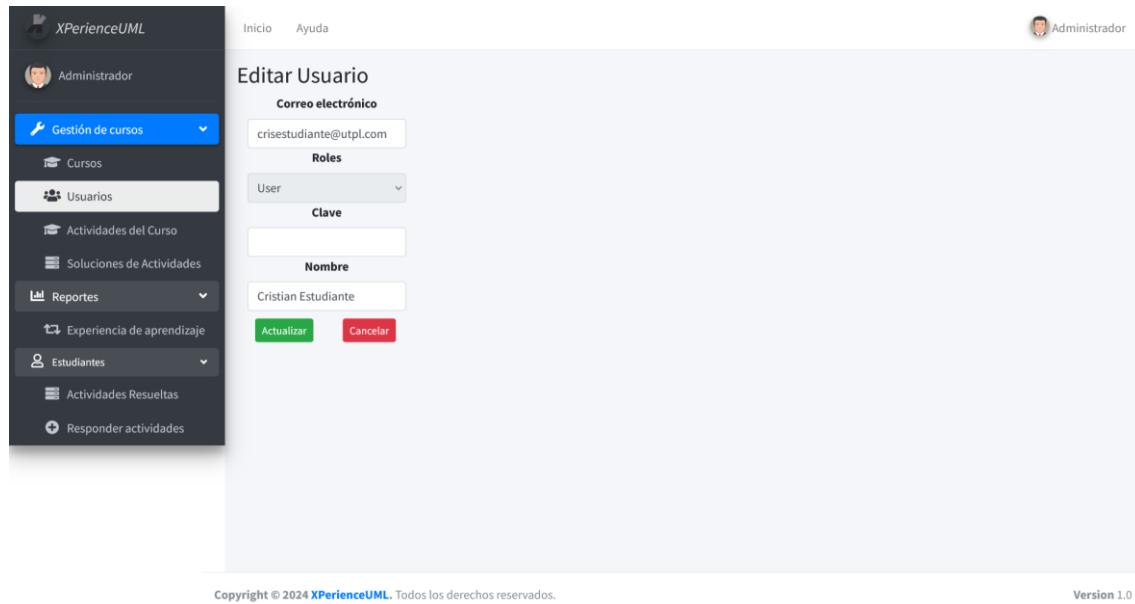
```
9. <section class="content-header">
10.   <div class="container-fluid">
11.     <h1>Usuario</h1>
12.     <table class="table">
13.       <tbody>
14.         <tr>
15.           <th>Correo</th>
16.           <td>{{ user.email }}</td>
17.         </tr>
18.         <tr>
19.           <th>Roles</th>
20.           <td>{{ user.roles[0]=='ROLE_USER' ? 'Estudiante' : 'Profesor' }}</td>
21.         </tr>
22.         <tr>
23.           <th>Nombre</th>
24.           <td>{{ user.name }}</td>
25.         </tr>
26.         <tr><td>
27.           <a class="btn btn-info btn-sm" href="{{ path('app_user_edit1', {'id':
28. user.id}) }}">Editar</a>
29.         </td>
30.         <td>
31.           </td></tr>
32.         </tbody>
33.       </table>
34.     </div>
35.   </section>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Si fuera necesario, el Docente puede cambiar los datos del usuario mediante la opción Editar, a la cual tiene acceso mediante el botón Editar, o a través del ícono de edición en la página que lista los usuarios del sistema

La opción *Editar* se emplea para actualizar o completar los datos del usuario. Cuando se selecciona *Editar* nos abre la pantalla que a continuación mostramos.

Figura 25 Vista de la edición de los datos del registro de un usuario



Fuente: (Macas, 2023)

Figura 26 Código del frontend de editar información de un usuario.

```
1.  {% extends 'base.html.twig' %}  
2.  
3.  {% block title %}Editar Usuario{% endblock %}  
4.  
5.  {% block body %}  
6.  
7.      <div class="content-wrapper" style="transform: translateX(30px); width:  
8.      calc(100% - 280px);">  
9.          <!-- Content Header (Page header) -->  
10.         <section class="content-header">  
11.             <div class="container-fluid">  
12.                 <div class="row mb-2">  
13.                     <div class="col-sm-6">  
14.                         <h1>Editar Usuario</h1>  
15. <div class="btn btn-group-horizontal">  
16. <table>  
17.     <tr><td>  
18.         {{ include('user/_form.html.twig', {'button_label': 'Actualizar'}) }}  
19.     </td></tr>  
20. </table>  
21. </div>  
22. </div>  
23. </div>  
24. </div>  
25. </section>  
26. </div>  
27. {% endblock %}
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 27 Código del backend de la actualizar la información de un usuario.

```
72. /**
73. * @Route("/{id}/edit", name="app_user_edit", methods={"GET", "POST"})
74. */
75. public function edit(Request $request, User $user, UserRepository
76. $userRepository): Response
77. {
78.     $form = $this->createForm(UserType::class, $user);
79.     $form->handleRequest($request);
80.     if ($form->isSubmitted() && $form->isValid()) {
81.         /* código para codificar clave      */
82.         $plainpwd = $user->getPassword();
83.         $encoded = $this->passwordEncoder->encodePassword($user, $plainpwd);
84.         $user->setPassword($encoded);
85.         $userRepository->add($user);
86.         $this->addFlash('success', 'Datos actualizados satisfactoriamente !!!');
87.         return $this->redirectToRoute('app_user_index', [], Response::HTTP_SEE_OTHER);
88.     }
89.     return $this->render('user/edit.html.twig', [
90.         'user' => $user,
91.         'form' => $form->createView(),
92.     ]);
93. }
94. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Como habíamos mencionado anteriormente la edición de un usuario se lleva a cabo por la parte del código desde el fichero *UserController.php*. La función *edit* de dicho fichero, que gestiona la edición de un usuario existente, define una ruta */id/edit* para esta acción, donde *id* es un parámetro dinámico que representa el ID del usuario. La ruta acepta tanto métodos GET como POST. Los parámetros que recibe son:

- Request \$request: Objeto que representa la solicitud HTTP.
- User \$user: Entidad User correspondiente al ID de la ruta.
- UserRepository \$userRepository: Repositorio para gestionar usuarios.

Mientras que devuelve una respuesta HTTP.

En la implementación de esta función se crea un formulario basado en la clase *UserType* y se le vincula al usuario existente. Se procesa los datos enviados en la solicitud HTTP. Se comprueba si el formulario ha sido enviado y si los datos son válidos. Entonces se codifica la contraseña. Esto es, obtener la contraseña sin codificar, utilizar el servicio *\$passwordEncoder* establecido en el constructor de la clase para codificarla, y asignar la contraseña codificada al usuario.

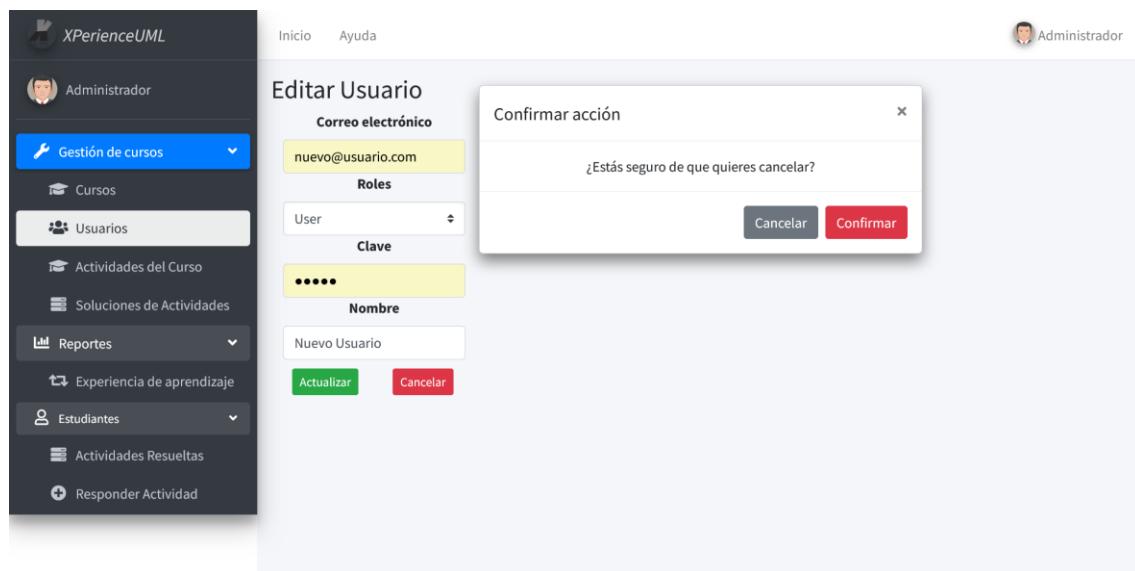
Luego se guarda el usuario, utilizando el repositorio `$userRepository` para persistir los cambios del usuario en la base de datos. Muestra un mensaje de éxito. Finalmente se redirige a la ruta `app_user_index` después de actualizar el usuario. En caso de que el formulario no fuera valido o hubiera un error en el, se renderiza la plantilla `user/edit.html.twig` pasando el objeto `$user` y el formulario como variables.

Como puede verse en la Figura 25, se consigue acceder a todos las propiedades de un usuario. Se puede *Actualizar* los datos, o no hacer nada (*Cancelar*) y regresar al listado de usuario.

En el caso de *Cancelar* la edición, se mostrará un cuadro de dialogo, para confirmar la acción que se ha escogido (Cancelar la edición). Si se cancela la cancelación el usuario puede continuar editando los datos de los usuarios. Si se *Confirma* la cancelación, la aplicación vuelve a la página donde se muestra el listado de usuarios.

Cuando se editan los datos y se presiona el botón *Actualizar*, los datos son grabados en la base de datos y se vuelve al listado de usuarios registrados para utilizar el sistema.

Figura 28 Cancelación de una acción de edición



Fuente: (Macas, 2023)

Figura 29 Código del frontend. Confirmación de la acción de cancelación.

```
15. <div class="modal" id="confirmModalUser" name="confirmModalUser"
16. style="display:none;">
17.   <div class="modal-dialog" role="document">
18.     <div class="modal-content">
19.       <div class="modal-header">
20.         <h5 class="modal-title" id="confirmModalLabel">Confirmar
21.         acción</h5>
22.         <button type="button" class="close" data-dismiss="modal" aria-
23.         label="Close">
24.           <span aria-hidden="true">&times;</span>
25.         </button>
26.       </div>
27.       <div class="modal-body">
28.         ¿Estás seguro de que quieres cancelar?
29.       </div>
30.       <div class="modal-footer">
31.         <button type="button" class="btn btn-secondary" data-
32.         dismiss="modal" id="cerrar" onclick="ocultarDivUser()">Cancelar</button>
33.         <a href="{{ path('app_user_index') }}" class="btn btn-
34.         danger">Confirmar</a>
35.       </div>
36.     </div>
37.   </div>
38. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En el listado de los usuarios, el docente puede elegir también la opción *Ver*, para visualizar el registro del usuario. A continuación mostramos la página de visualización de datos de un usuario.

Antes de crearse un curso o una actividad, los docentes tienen que crear los usuarios, debido a que cuando se crea un curso se seleccionan los estudiantes que lo cursaran y las actividades son creadas a partir de los cursos.

## Gestión de cursos

Luego de gestionar los usuarios, se deben gestionar los cursos.

Se basa en la creación, modificación y eliminación de cursos. Solo está permitido para usuarios con rol Docente.

El docente ingresa en la aplicación web, se autentica y selecciona la opción de gestión de cursos. Se visualiza entonces el listado de los cursos creados hasta el momento. (Ver Figura 30)

Figura 30 Vista Gestión de Cursos. Listado de Cursos

The screenshot shows the XperienceUML application interface. On the left is a sidebar with navigation links: 'Gestión de cursos' (selected), 'Cursos' (highlighted), 'Usuarios', 'Actividades del Curso', 'Soluciones de Actividades', 'Reportes' (selected), 'Experiencia de aprendizaje', 'Estudiantes' (selected), 'Actividades Resueltas', and 'Responder actividades'. At the top right is a user profile for 'Administrador'. The main content area is titled 'Lista de Cursos' and displays a table of three courses:

Nombre	Fecha Inicio	Fecha Fin	Acciones
El mundo de la programación orientada a objetos	2024-06-01	2024-06-30	
Herencia y Polimorfismo en Programación Orientada a Objetos	2024-06-01	2024-06-30	
Introducción a la Programación Orientada a Objetos	2024-01-01	2024-08-31	

Below the table, it says 'Mostrando página 1 de 1'. At the bottom left is a blue button 'Agregar curso +', and at the bottom right are buttons for 'Anterior' (with page number 1) and 'Próximo'. The footer contains copyright information: 'Copyright © 2024 XperienceUML. Todos los derechos reservados.' and 'Versión 1.0'.

Fuente: (Macas, 2023)

La información que se listarán específicamente serán el nombre del curso, y las fechas de inicio y fin de ellos.

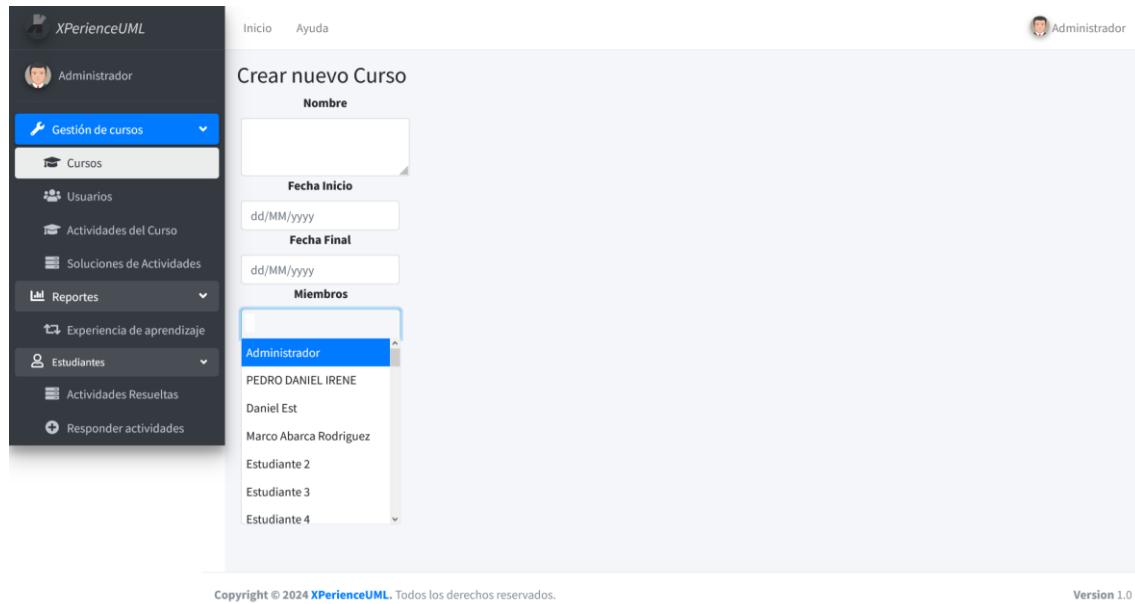
Figura 31 Código del frontend. Listado de cursos.

```
28. <div class="card-body table-responsive p-0" style="height: auto;">
29.     <table id="example1" class="table table-bordered table-striped dataTable"
30.         role="grid" aria-describedby="example1_info">
31.         <thead>
32.             <tr>
33.                 <th>Nombre</th>
34.                 <th>Fecha Inicio</th>
35.                 <th>Fecha Fin</th>
36.                 <th>Acciones</th>
37.             </tr>
38.         </thead>
39.         <tbody>
40.             {% for course in courses %}
41.             <tr>
42.
43.                 <td>{{ course.name }}</td>
44.                 <td>{{ course.initialDate ? course.initialDate|date('Y-m-d') :
45. : '' }}</td>
46.                 <td>{{ course.fdate ? course.fdate|date('Y-m-d') : '' }}</td>
47.                 <td>
48.                     <a href="{{ path('app_course_show', {'id': course.id}) }}"
49. {"} " title="Ver">
50.                         <i class="fa fa-eye"></i>
51.                     </a>
52.
53.                     {% if is_granted('ROLE_ADMIN') %}
54.                         <a href="{{ path('app_course_edit', {'id':
55. course.id}) }}" title="Editar">
56.                             <i class="fa fa-edit"></i>
57.                         </a>
58.                     <a href="{{ path('app_course_delete', {'id': course.id}) }}"
59. {"} " title="Borrar">
60.                         <i class="fa fa-trash"></i>
61.                     </a>
62.                     {% endif %}
63.                 </td>
64.             </tr>
65.             {% else %}
66.             <tr>
67.                 <td colspan="5">no hay datos</td>
68.             </tr>
69.             {% endfor %}
70.         </tbody>
71.     </table>
72. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Mediante la opción *Agregar curso*, podrá crear nuevos cursos para el sistema. A continuación mostramos la vista de la página para crear un nuevo curso.

Figura 32 Gestión de cursos. Crear nuevo curso



Fuente: (Macas, 2023)

Figura 33 Código del frontend de formulario para crear nuevo curso.

```
2. <div class="row">
3.   <div class="col-14">
4.     {{form_row(form.name, {'attr': {'class': 'form-control'}})}}
5.   </div>
6. </div>
7.
8. <div class="row">
9.   <div class="col-14">
10.    {{ form_row(form.initialDate, {'attr': {'class': 'form-control'}})}}
11.   </div>
12. </div>
13. <div class="row">
14.   <div class="col-14">
15.     {{ form_row(form.fdate, {'attr': {'class': 'form-control'}})}}
16.   </div>
17. </div>
18. <div class="row" >
19. <div class="col-14" >
20.   {{ form_row(form.members, {'attr': {'class': 'form-control'}})}}
21. </div>
22. </div>
23.
24. <div class="row" style="margin-top:10px">
25.   <div class="col-6">
26.     <button class="btn btn-success btn-sm">{{ button_label|default('Guardar') }}</button>
27.   </div>
28.   <div class="col-6">
29.     <button type="button" class="btn btn-danger btn-sm "
30. onclick="mostrarDivCourse()>Cancelar</button>
31.   </div>
32. </div>
33. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 34 Código del backend de guardar la información de un curso.

```
28. /**
29. * @Route("/new", name="app_course_new", methods={"GET", "POST"})
30. */
31. function new (Request $request, CourseRepository $courseRepository): Response {
32.     $course = new Course();
33.     $form = $this->createForm(CourseType::class, $course);
34.     $form->handleRequest($request);
35.     if ($form->isSubmitted() && $form->isValid()) {
36.         $courseRepository->add($course);
37.         $this->addFlash("success", 'Insertado satisfactoriamente!!! ');
38.         return $this->redirectToRoute('app_course_index', []),
39. Response::HTTP_SEE_OTHER);
40.     }
41.     return $this->render('course/new.html.twig', [
42.         'course' => $course,
43.         'form' => $form->createView(),
44.     ]);
45. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la gestión de los cursos desde el punto de vista de código se emplea el fichero *CourseController.php*, es una clase controladora de Symfony. En esta clase se implementan los métodos para listar, crear, modificar y eliminar un curso.

En el caso de la gestión de la creación de un nuevo curso, se emplea la función *new*, la cual define una ruta */new* para esta acción, accesible tanto por métodos GET como POST. Los parámetros que recibe son:

- Request \$request: Objeto que representa la solicitud HTTP.
- CourseRepository \$courseRepository: Repositorio para gestionar cursos.

Mientras que devuelve una respuesta HTTP.

En la implementación de esta función inicialmente se crea un nuevo objeto *Course*. Luego se crea un formulario basado en la clase *CourseType* y se le vincula al objeto *\$course*. Se procesan los datos enviados en la solicitud HTTP. Se comprueba si el formulario ha sido enviado y si los datos son válidos, entonces se guarda el curso, utilizando el repositorio *\$courseRepository* para persistir el curso en la base de datos. Se muestra un mensaje de éxito. Finalmente se redirige a la ruta *app\_course\_index* después de guardar el curso. En caso de que el formulario no sea válido

o tenga algún error se renderiza la plantilla *course/new.html.twig* pasando el objeto *\$course* y el formulario como variables.

Los datos que se recopilan para cada curso son nombre, miembros, fecha de inicio y fecha de finalización.

Figura 35 Código del backend. Formulario de cursos.

```
23. public function buildForm(FormBuilderInterface $builder, array $options): void
24. {
25.     $builder
26.         ->add('name', TextareaType::class,
27.             array('disabled'=>$options['isEdit'],
28.                 'label' => 'Nombre',
29.                 'attr' => array('class' => 'form_control')
30.             )
31.         )
32.         ->add('initialDate', DateTimeType::class,
33.             array('disabled'=>$options['isEdit'],
34.                 'label' => 'Fecha Inicio',
35.                 'html5' => false,
36.                 'format' => 'dd/MM/yyyy',
37.                 'widget' => 'single_text',
38.                 'attr' => array('class' => 'form-control', 'placeholder' =>
39. 'dd/MM/yyyy'),
40.             )
41.         )
42.         ->add('fdate', DateType::class,
43.             array('disabled'=>$options['isEdit'],
44.                 'label' => 'Fecha Final',
45.                 'html5' => false,
46.                 'format' => 'dd/MM/yyyy',
47.                 'widget' => 'single_text',
48.                 'attr' => array('class' => 'form-control', 'placeholder' =>
49. 'dd/MM/yyyy'),
50.             )
51.         )
52.         ->add('members', EntityType::class, array(
53.             'label' => 'Miembros',
54.             'class' => User::class,
55.             'expanded' => false,
56.             'multiple' => true,
57.             'attr' => array('style' => 'height: 150px; overflow-y: scroll;', 'class' =>
58. 'form_control'),
59.             'required'=>false,
60.         ))
61.     ;
62. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código presentado en la Figura 35 define una función *buildForm* que construye un formulario para un objeto *Course*. El formulario incluye campos para el nombre, fecha de inicio, fecha final y una selección múltiple de miembros. La función utiliza el *FormBuilderInterface* para agregar los campos al formulario. Para esta función los parámetros son:

- *FormBuilderInterface \$builder*: Objeto utilizado para construir el formulario.

- array \$options: Opciones adicionales para el formulario.

En la implementación primero se Agrega un campo de tipo *TextareaType* para el nombre del curso. El campo está deshabilitado si la opción *isEdit* es *true*, mientras que si se pone a *false* significa que el campo será editable. Se establece la etiqueta del campo como 'Nombre'. Se aplica la clase CSS *form\_control* al campo.

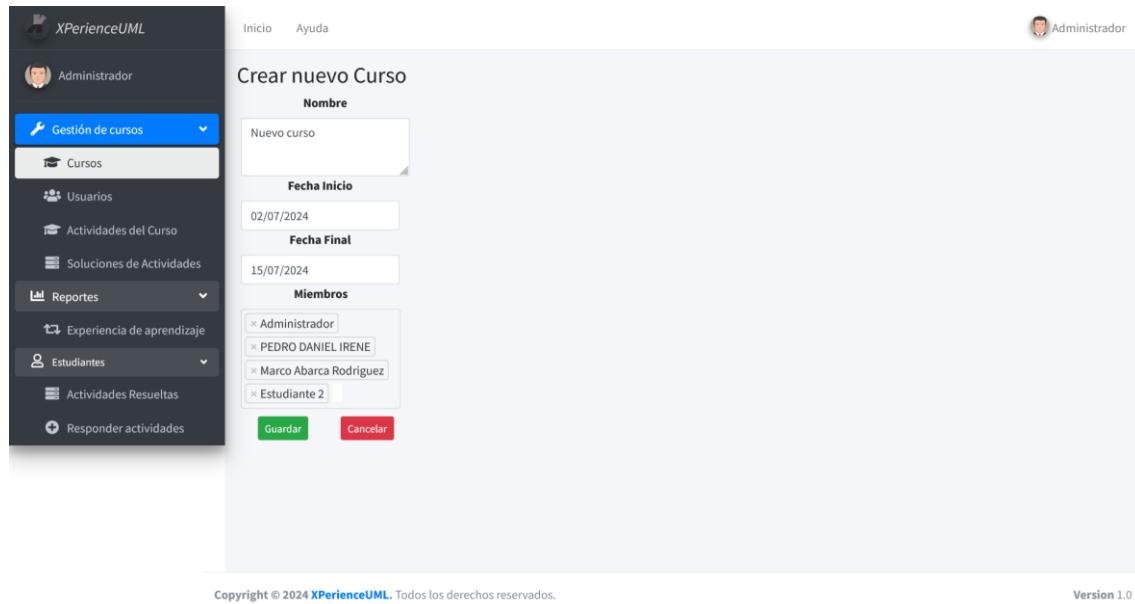
Luego se agregan los campos de tipo *DateTimeType* para la fecha de inicio y la fecha final del curso. Ambos campos están deshabilitados si la opción *isEdit* es true. Se establece la etiqueta del campo como 'Fecha Inicio' y 'Fecha Final' respectivamente. Igual para ambos se utiliza el formato *dd/MM/yyyy* para la visualización y se aplica la clase CSS *form\_control* a cada campo.

Finalmente se agrega un campo de tipo *EntityType* para seleccionar múltiples miembros del curso. Se establece la etiqueta del campo como 'Miembros'. La clase de entidad asociada es *User*. El campo se muestra como una lista desplegable, donde se permite la selección múltiple de miembros. Se aplica un estilo CSS para el campo. Este campo no es requerido.

En la página del listado de cursos además de los datos de los cursos se tienen dos acciones para cada uno de los cursos: *Ver* y *Editar*.

Con la opción *Editar* puede Guardarse los cambios o Cancelarse la edición de los datos de un curso. (Ver Figura 36)

Figura 36 Datos de un curso nuevo



Fuente: (Macas, 2023)

Figura 37 Código del backend de editar la información de un curso.

```
96. /**
97. * @Route("/{id}/edit", name="app_course_edit", methods={"GET", "POST"})
98. */
99. public function edit(Request $request, Course $course, CourseRepository
100. $courseRepository): Response
101. {
102.     $form = $this->createForm(CourseType::class, $course);
103.     $form->handleRequest($request);
104.     if ($form->isSubmitted() && $form->isValid()) {
105.         $courseRepository->add($course);
106.         $this->addFlash('success', 'Datos actualizados satisfactoriamente !!!');
107.         return $this->redirectToRoute('app_course_index', [], Response::HTTP_SEE_OTHER);
108.     }
109.     return $this->render('course/edit.html.twig', [
110.         'course' => $course,
111.         'form' => $form->createView(),
112.     ]);
113. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la gestión de la modificación o edición de un curso existente se emplea el fichero *CourseController.php*, como se mencionó anteriormente. Se lleva a cabo mediante la función *edit*, en la que se define una ruta */id/edit* para esta acción, donde *id* es un parámetro dinámico

que representa el ID del curso. La ruta acepta tanto métodos GET como POST. Recibe como parámetros:

- Request \$request: Objeto que representa la solicitud HTTP.
- Course \$course: Entidad Course correspondiente al ID de la ruta.
- CourseRepository \$courseRepository: Repositorio para gestionar cursos.

Mientras que devuelve una respuesta HTTP.

En la implementación, inicialmente se crea un formulario basado en la clase *CourseType* y se le vincula al curso existente. Luego se procesan los datos enviados en la solicitud HTTP. Se comprueba si el formulario ha sido enviado y si los datos son válidos. Entonces se guarda el curso, utilizando el repositorio *\$courseRepository* para persistir los cambios del curso en la base de datos. Se muestra un mensaje de éxito. Finalmente se redirige a la ruta *app\_course\_index* después de actualizar el curso. Si el formulario no es válido o existe algún error se renderiza la plantilla *course/edit.html.twig* pasando el objeto *\$course* y el formulario como variables.

Si se elige en el listado de cursos la acción **Ver** para un curso, se detallan los datos del curso en cuestión. (Ver Figura 38)

Figura 38 Vista de los datos de un curso

The screenshot shows the XperienceUML application interface. On the left is a sidebar with a dark header 'XperienceUML' and a user icon. It contains several menu items under 'Gestión de cursos': 'Cursos' (selected), 'Usuarios', 'Actividades del Curso', 'Soluciones de Actividades', 'Reportes' (selected), and 'Experiencia de aprendizaje'. Under 'Estudiantes': 'Actividades Resueltas' and 'Responder actividades'. At the bottom of the sidebar are links for 'Matricular estudiantes', 'Actividades creadas', 'Actividades realizadas', and 'Crear actividad'. The main content area has a header 'Curso'. Below it, there are four data entries: 'Nombre: Herencia y Polimorfismo en Programación Orientada a Objetos', 'Fecha de Inicio: 2024-06-01', 'Fecha Fin: 2024-06-30', and 'Miembros: [button] Ver miembros [button]'. At the bottom of the main content area are buttons for 'Matricular estudiantes', 'Actividades creadas', 'Actividades realizadas', and 'Crear actividad'. The top right corner shows a user icon and the text 'Administrador'. The footer contains the copyright notice 'Copyright © 2024 XperienceUML. Todos los derechos reservados.' and 'Version 1.0'.

Fuente: (Macas, 2023)

Figura 39 Código del frontend de para ver datos de un curso.

```
10.      <tbody>
11.          <tr>
12.              <th>Nombre:</th>
13.              <td>{{ course.name }}</td>
14.          </tr>
15.          <tr>
16.              <th>Fecha de Inicio:</th>
17.              <td>{{ course.initialDate ? course.initialDate|date('Y-m-d ') : '' }}</td>
18.          </td>
19.          </tr>
20.          <tr>
21.              <th>Fecha Fin:</th>
22.              <td>{{ course.fdate ? course.fdate|date('Y-m-d ') : '' }}</td>
23.          </tr>
24.          <tr>
25.              <th>Miembros:</th>
26.              <td>
27.                  <a class="btn btn-success btn-sm" href="{{ path('app_course_students', {'id': course.id}) }}>Ver miembros</a>
28.              </td>
29.          </tr>
30.          </tr>
31.          <tr><td></td><td></td></td></tr>
32.      </tbody>
33.  </table>
34. <table>
35. <tr>
36. <td>
37. {% if is_granted('ROLE_ADMIN') %}
38. <a class="btn btn-info btn-sm" href="{{ path('app_course_members', {'id': course.id}) }}>Matricular estudiantes</a>
39. <a class="btn btn-info btn-sm" href="{{ path('app_n_activity_show_course', {'id': course.id}) }}>Actividades creadas</a>
40. <a class="btn btn-info btn-sm" href="{{ path('app_model_diagram_test_index1', {'id': course.id}) }}>Actividades realizadas</a>
41. <a class="btn btn-info btn-sm" href="{{ path('app_n_activity_new', {'id': course.id}) }}>Crear actividad</a>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para ver los estudiante matriculados puede presionar el botón *Ver miembros*. Esto nos llevará a una lista de miembros del curso seleccionado, visualizándose específicamente el nombre y el correo. (Ver Figura 40)

Figura 40 Listado de estudiantes matriculados en un curso

Nombre	Email
Cristian Estudiante	crisestudiante@utpl.com
Daniel Est	pdirene@gmail.com
Estudiante 10	estudiante10@utpl.com
Estudiante 11	estudiante11@utpl.com
Estudiante 12	estudiante12@utpl.com
Estudiante 13	estudiante13@utpl.com
Estudiante 14	estudiante14@utpl.com
Estudiante 15	estudiante15@utpl.com
Estudiante 16	estudiante16@utpl.com
Estudiante 17	estudiante17@utpl.com

Fuente: (Macas, 2023)

Figura 41 Código del frontend. Listados de miembros de un curso.

```

13. <h1>Lista de Miembros del curso {{ course_name }}</h1>
14. <div class="card-body table-responsive p-0" style="height: auto;">
15.     <table id="example1" class="table table-bordered table-striped dataTable"
16.         role="grid" aria-describedby="example1_info">
17.         <thead>
18.             <tr>
19.                 <th>Nombre</th>
20.                 <th>Email</th>
21.             </tr>
22.         </thead>
23.         <tbody>
24.             {% for user in users %}
25.                 <tr>
26.                     <td>{{ user.name }}</td>
27.                     <td>{{ user.email }}</td>
28.                 </tr>
29.             {% else %}
30.                 <tr>
31.                     <td colspan="5">no hay datos</td>
32.                 </tr>
33.             {% endfor %}
34.         </tbody>
35.     </table>
36. </div>
37. {% if is_granted('ROLE_ADMIN')%}
38.     <a class="btn btn-info btn-sm" href="{{ path('app_course_members', {'id':
39. course_id}) }}">Matricular estudiantes</a>
40.     <a class="btn btn-info btn-sm" href="{{ path('app_course_show', {'id':
41. course_id}) }}">
42.         Volver al curso
43.     </a>
44. {% endif %}
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Desde la página de visualización de los datos de un curso, así como desde la página donde se visualizan los miembros de un curso, puede llevarse a cabo la opción *Matricular estudiantes*. Esto muestra la página que se ve en la figura siguiente.

Figura 42 Edición de miembros de un curso

The screenshot shows the XperienceUML application interface. On the left is a sidebar with the following menu items:

- Gestión de cursos
  - Cursos (highlighted)
  - Usuarios
  - Actividades del Curso
  - Soluciones de Actividades
- Reportes
  - Experiencia de aprendizaje
- Estudiantes
  - Actividades Resueltas
  - Responder actividades

The main content area is titled "Editar Miembros del Curso". It has two sections: "Nombre" (Name) and "Miembros". The "Nombre" section contains a dropdown menu with the following options:

- Introducción a la Programación Orientada a Objetos
- PEDRO DANIEL IRENE (highlighted)
- Daniel Est
- Marco Abarca Rodriguez
- Cristian Estudiante

The "Miembros" section contains a list of student names:

- Administrador
- PEDRO DANIEL IRENE
- Daniel Est
- Marco Abarca Rodriguez
- Estudiante 2
- Estudiante 3
- Estudiante 4

At the bottom of the page, there is a copyright notice: "Copyright © 2024 XperienceUML. Todos los derechos reservados." and a version number: "Version 1.0".

Fuente: (Macas, 2023)

Cuando va a editarse los miembros de un curso, se carga el formulario de datos del curso, pero solo con la posibilidad de modificar los miembros del curso. Los demás campos quedarán inhabilitados sin posibilidades de modificación.

Figura 43 Código del frontend de para editar los estudiantes de un curso.

```
2. {{ form_start(form) }}
3. <div class="row">
4. <div class="col-14">
5.   {{ form_row(form.name, { 'attr': { 'disabled': 'disabled' , 'class': 'form-
6. control' } }) }}
7. </div>
8. </div>
9. <div class="row">
10. <div class="col-14">
11.   {{ form_row(form.members, { 'attr': { 'class': 'form-control' }}) }}
12. </div>
13. </div>
14. <div class="row">
15. <div class="col-14">
16.   {{ form_row(form.initialDate, { 'attr': { 'disabled': 'disabled' , 'class': 'form-
17. control' } }) }}
18. </div>
19. </div>
20. <div class="row">
21. <div class="col-14">
22.   {{ form_row(form.fdate, { 'attr': { 'disabled': 'disabled' , 'class': 'form-
23. control' } }) }}
24. </div>
25. </div>
26. <div class="row" style="margin-top:10px">
27.   <div class="col-6">
28.     <button class="btn btn-success btn-sm">{{ button_label|default('Guardar') }}</button>
29.   </div>
30.   <div class="col-6">
31.     <a class="btn btn-danger btn-sm" href="{{ path('app_course_index') }}>Cancelar</a>
32.   </div>
33. </div>
34. </div>
35. </div>
36. {{ form_end(form) }}
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 44 Código del backend. Función que llama a listar de miembros de un curso.

```
114. /**
115. * @Route("/{id}/members", name="app_course_members", methods={"GET", "POST"})
116. */
117. public function members(Request $request, Course $course, CourseRepository
118. $courseRepository): Response
119. {
120.   $form = $this->createForm(CourseType::class, $course, ['isEdit'=> true,]);
121.   $form->handleRequest($request);
122.   if ($form->isSubmitted() && $form->isValid()) {
123.     $courseRepository->add($course);
124.     $this->addFlash('success', 'Datos actualizados satisfactoriamente !!!');
125.     return $this->redirectToRoute('app_course_students', ['id' => $course-
126. >getId(),], Response::HTTP_SEE_OTHER);
127.   }
128.   return $this->render('course/members.html.twig', [
129.     'course' => $course,
130.     'form' => $form->createView(),
131.   ]);
132. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código presentado en la Figura 44 es una función de controlador Symfony que se encarga de mostrar y potencialmente actualizar los miembros de un curso, que define una ruta */id/members* para esta acción, donde *id* es un parámetro dinámico que representa el ID del curso. La ruta acepta tanto métodos GET como POST. Los parámetros que se emplean son:

- Request \$request: Objeto que representa la solicitud HTTP.
- Course \$course: Entidad Course correspondiente al ID de la ruta.
- CourseRepository \$courseRepository: Repositorio para gestionar cursos.

Mientras que devuelve una respuesta HTTP.

Inicialmente en la implementación se crea un formulario basado en la clase *CourseType* y se le vincula al curso existente. Se pasa una opción *isEdit* con valor true al formulario. Procesa los datos enviados en la solicitud HTTP. Se comprueba si el formulario ha sido enviado y si los datos son válidos. Se guarda el curso entonces, utilizando el repositorio *\$courseRepository* para persistir los cambios del curso en la base de datos. Se muestra un mensaje de éxito. Finalmente se redirige a la ruta *app\_course\_students* después de actualizar el curso, pasando el ID del curso como parámetro. Si el formulario contiene datos no válidos o erróneos se renderiza la plantilla *course/members.html.twig* pasando el objeto *\$course* y el formulario como variables.

Figura 45 Código del backend. Listados de miembros de un curso.

```
56. /**
57. * @Route("/{id}/users", name="app_course_students", methods={"GET"})
58. */
59. public function students(CourseRepository $courseRepository, int $id): Response
60. {
61.     return $this->render('course/showMembers.html.twig', [
62.         'users' => $courseRepository->findUserByCourseId($id),
63.         'course_name' => $courseRepository->find($id),
64.         'course_id' => $id,
65.     ]);
66. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Los miembros de un curso, es decir, los usuarios que han sido matriculados para un curso, se visualizan mediante la función *students* del fichero *CourseController.php*, la cual hace una consulta a la base de datos de todos los usuarios que han sido vinculados con el curso. Esta

función es invocada por la función *member*, que la que le dice a la función *students*, cuál es el curso del cual va a visualizarse la matrícula. Una vez seleccionado los miembros, en los que puede adicionarse o eliminarse estudiantes, se retorna a la página que tiene el listado de estudiantes para el curso.

Para cada curso se tiene además otras acciones como *Actividades Creadas*, *Actividades realizadas*, *Crear actividad*, acciones que aunque parten del grupo de funciones del curso, pertenecen a la gestión de actividades. De ahí que serán analizadas en el siguiente epígrafe.

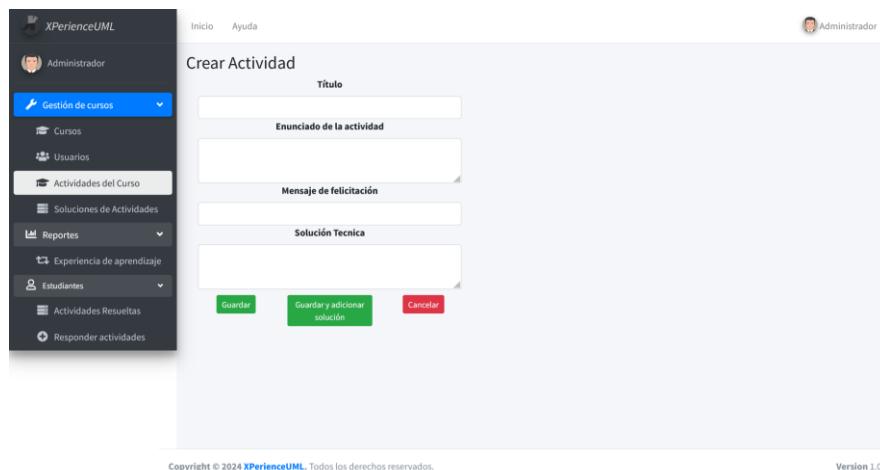
## Gestión de actividades

La gestión de actividades se basa en la creación, modificación, visualización y eliminación de actividades. Solo está permitido para usuarios con rol Docente.

La creación de actividades solo puede realizarse al visualizar los datos de un curso.

Si selecciona *Crear actividad* desde la visualización de los datos de un curso (Ver Figura 38), se creará una nueva actividad para el curso. Si selecciona *Actividades Creadas*, se listarán las actividades implementadas por los docentes como ejercicios y que luego realizarán los estudiantes matriculados en el curso en cuestión. Mientras que si se elige *Actividades realizadas*, se mostrarán las soluciones de los estudiantes a las actividades del curso, especificándose el nombre del estudiante, la fecha en que realizó la actividad, el curso al que pertenece la actividad, así como el título de la actividad. Cuando uno se refiere a las actividades realizadas, se está haciendo alusión a las actividades que han sido desarrolladas por los estudiantes. Con esta acción solo podrá listarse las actividades con su nombre, el curso al que pertenece, los nombres de los estudiantes que han dado una solución, así como la fecha en que fue dada la solución. La única acción permitida para esta vista es *Ver* que se refiere al diagrama UML desarrollado por los estudiantes.

Figura 46 Creación de una actividad



Fuente: (Macas, 2023)

Figura 47 Código del frontend. Formulario para crear una actividad de un curso.

```
4. <div class="container-fluid">
5.     <div class="row">
6.         <div class=" col-sm-12">
7.             {{form_row(form.title)}}
8.         </div>
9.     </div>
10.    <div class="row">
11.        <div class="col-sm-12">
12.            {{form_row(form.description)}}
13.        </div>
14.        <div class="col-sm-12">
15.            {{form_row(form.place)}}
16.        </div>
17.    </div>
18.    <div class="row">
19.        {{form_row(form.nta)}}
20.    </div>
21.    <div class="row">
22.        <div class="col-sm-12">
23.            {{form_row(form.tecsol)}}
24.        </div>
25.    </div>
26. </div>
27.
28. <div class="row" style="margin-top:10px">
29.     <div class="col">
30.         <button id="acceptButton" class="btn btn-success btn-sm">{{ button_label|default('Guardar') }}</button>
31.     </div>
32.     <div class="col">
33.         {% if not('/edit' in app.request.uri) %}
34.             <button id="confirmButton" name="confirmButton" class="btn btn-success btn-sm">{{ button_label|default('Guardar y adicionar solución') }}</button>
35.         </div>
36.     </div>
37. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Los datos que se registran para una actividad como puede verse en la Figura 46, son título, enunciado de la actividad, mensaje de felicitación y solución técnica. La solución técnica es

una ayuda que le brindará el Docente al estudiante para realizar de la actividad. Una vez entrado los datos estos se guardarán al presionar el botón **Guardar**. Al crearse una actividad, se da la posibilidad de crear también la solución al ejercicio que se está referenciando con la actividad. Para ellos se tiene el botón **Guardar y adicionar solución**, con el que se abrirá el editor de diagrama UML y que veremos en la próxima sección. Si se decide no registrar la actividad se elige la opción *Cancelar*.

Figura 48 Código del backend de guardar una actividad creada de un curso.

```

39. /**
40.  * @Route("/new/{id}", name="app_n_activity_new", methods={"GET", "POST"})
41. */
42. function new (Request $request, NActivityRepository $nActivityRepository, int
43. $id): Response
44. {
45.     $nActivity = new NActivity();
46.     $form = $this->createForm(NActivityType::class, $nActivity);
47.     $form->handleRequest($request);
48.     if ($form->isSubmitted() && $form->isValid()) {
49.         $em = $this->getDoctrine()->getManager();
50.         /** @var NTA $interactiva */
51.         $interactiva = $em->getRepository('App:NTA')->findByName(array('denomination' =>
52. 'Interactiva'));
53.         $nActivity->setNTA($interactiva[0]);
54.         $nActivityRepository->add($nActivi
55.         $course = $em->getRepository('App:Course')->findOneBy(array('id' => $id));
56.         $diagramSuccess = new ModelDiagramSuccess();
57.         $diagramSuccess->setCourse($course);
58.         $diagramSuccess->setNActivity($nActivity);
59.         $diagramSuccess->setData([]);
60.         $modeldiagramsuccessRepo = $em->getRepository('App:ModelDiagramSuccess');
61.         $modeldiagramsuccessRepo->add($diagramSuccess);
62.         $idDiagramSuccess= $diagramSuccess->getId();
63.         $diagram = new ModelDiagramTest();
64.         $diagram->setCourse($course);
65.         $diagram->setNActivity($nActivity);
66.         $user = $this->getUser();
67.         $diagram->setAction($user->getName());
68.         $diagram->setData([]);
69.         $diagram->setFecha();
70.         $modeldiagramtestRepo = $em->getRepository('App:ModelDiagramTest');
71.         $modeldiagramtestRepo->add($diagram);
72.         $nActivity->addModelDiagramTest($diagram);
73.         $nActivity->addModelDiagramSuccess($diagramSuccess);
74.         $course->addModelDiagramTest($diagram);
75.         $course->addModelDiagramSuccess($diagramSuccess);
76.         if ($request->getMethod() === 'POST' && $request->request-
77. >has('confirmButton'))
78.     {
79.         return $this->redirect('/public/GoJS-
80. master/projects/pdf/diagrama.php?mode=edit&table=model_diagram_success&id='.(stri
81. ng)$idDiagramSuccess.'&course='.(string)$id);
82.     }
83.     return $this->render('n_activity/index.html.twig', [
84.         'n_activities' => $nActivityRepository-
85. >findDataToShowIdSuccessByCourse($id),
86.         'title' => 'Actividades del curso '. $course->getName(),]);
87. }
88. return $this->render('n_activity/new.html.twig', [
89.     'n_activity' => $nActivity,
90.     'id' => $id,
91.     'form' => $form->createView(),
92. ]);
93. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la gestión de actividades se emplea el fichero *NActivityController.php*, que es clase controladora de Symfony, con la cual se puede crear, modificar, visualizar y borrar actividades.

En la creación de una actividad mediante el botón *Guardar*, se emplea la función *new*, que se encarga de crear una nueva actividad (*NActivity*) asociada a un curso específico. Además, crea registros relacionados en otras entidades para gestionar diagramas y datos asociados. La función define una ruta */new/{id}* para esta acción, donde id es un parámetro dinámico que representa el ID del curso. La ruta acepta tanto métodos GET como POST. Los parámetros que emplea son:

- Request \$request: Objeto que representa la solicitud HTTP.
- NActivityRepository \$nActivityRepository: Repositorio para gestionar actividades.
- int \$id: ID del curso asociado.

Mientras que devuelve una respuesta HTTP.

En la implementación, primeramente se crea una nueva instancia de *NActivity*. Se crea un formulario basado en la clase *NActivityType* y se le vincula al objeto *\$nActivity*. Se procesan los datos enviados en la solicitud HTTP. Luego se comprueba si el formulario ha sido enviado y si los datos son válidos. Entonces se obtiene el entity manager. Se busca una entidad *NTA* con la denominación 'Interactiva' y la asigna a la nueva actividad. Se persiste la nueva actividad en la base de datos. Se obtiene el curso correspondiente al ID proporcionado como parámetro. Se crea una nueva instancia de *ModelDiagramSuccess* y se le relaciona con el curso y la actividad.

Se persiste el *ModelDiagramSuccess* en la base de datos y obtiene su ID. Se crea una nueva instancia de *ModelDiagramTest* y se le relaciona con el curso, la actividad y el usuario. Se persiste el *ModelDiagramTest* en la base de datos. Se relacionan los diagramas (*ModelDiagramSuccess* y *ModelDiagramTest*) con la actividad y el curso.

Si se presiona el botón '*confirmButton*', se redirige a la URL específica del editor de diagramas UML. Ese botón corresponde con la acción *Guardar y Adicionar solución*.

En caso de que el formulario no sea válido o contenga datos erróneos, se renderiza la plantilla *n\_activity/new.html.twig* con los datos necesarios.

Cuando se acepta el registro de una actividad se muestra el listado de todas las actividades de todos los cursos. Este listado está compuesto también por el título, nombre del curso, descripción, retroalimentación y solución técnica. (Ver Figura 49). Esta vista es la misma cuando se selecciona *Actividades creadas* en la página de información de un curso.

*Figura 49 Listado de actividades creadas de un curso*

Título	Curso	Enunciado	Retroalimentación	Solución Técnica	Acciones
Casa y Habitaciones	El mundo de la programación orientada a objetos	Imagina que estás diseñando un sistema para gestionar la información de una casa. Cada casa tiene una dirección y está compuesta por varias habitaciones. Las habitaciones no pueden existir independientemente de la casa. Cada habitación tiene un tipo (como sala, cocina, dormitorio) y una dimensión metros cuadrados. Recuerda que en diseño de una casa puedes agregar o eliminar habitaciones según sea la necesidad.	<i>¡Excelente! Has modelado correctamente la relación entre la casa y las habitaciones.</i>	<ol style="list-style-type: none"> <li>1. Crea la clase Casa con los atributos dirección (Cadena) y habitaciones (Lista de Cadenas).</li> <li>2. En la clase Casa, añade los métodos agregarHabitacion (Void) y eliminarHabitacion (Void).</li> <li>3. Crea la clase Habitacion con los atributos tipo (Cadena) y dimension (Entero).</li> <li>4. Establece que Casa compone a Habitacion. --Relación: Casa &lt;--&gt; Habitacion (Composición).</li> </ol>	
Departamento y Empleados	El mundo de la programación orientada a objetos	En una empresa, un departamento está compuesto por varios empleados. Cada empleado tiene un nombre y un puesto específico, y pertenece a un único departamento. Sin embargo, un departamento puede existir independientemente de los empleados. Debes modelar esta situación creando una estructura adecuada para representar los departamentos y los empleados. Incluye métodos para agregar y eliminar empleados de un departamento.	<i>¡Excelente! Has modelado correctamente la relación entre el departamento y los empleados.</i>	<ol style="list-style-type: none"> <li>1. Crea la clase Departamento con los atributos nombre (Cadena) y empleados (Lista de Cadenas).</li> <li>2. En la clase Departamento, añade los métodos agregarEmpleado (Void) y eliminarEmpleado (Void).</li> <li>3. Crea la clase Empleado con los atributos nombre (Cadena) y puesto (Cadena).</li> <li>4. Establece que Departamento agrega a Empleado. ---Relación: Departamento &lt;-&gt; Empleado (Agregación).</li> </ol>	
Sistema de Notificaciones	El mundo de la programación orientada a objetos	En una aplicación de mensajería, el sistema de notificaciones necesita enviar notificaciones a los usuarios a través de un servicio de mensajería. Este servicio puede enviar mensajes por correo electrónico o SMS. Debes modelar esta situación creando una estructura adecuada para representar el sistema de notificaciones y el servicio de mensajería.	<i>¡Excelente! Has modelado correctamente la relación entre el sistema</i>	<ol style="list-style-type: none"> <li>1. Crea la clase SistemaNotificaciones con un método enviarNotificacion de tipo Void.</li> <li>2. Crea la clase ServicioMensajeria con métodos enviarCorreo (Void) y enviarSMS (Void).</li> <li>3. Establece que SistemaNotificaciones depende de ServicioMensajeria. ---Relación: SistemaNotificaciones --&gt; ServicioMensajeria (Dependencia).</li> </ol>	

Fuente: (Macas, 2023)

Figura 50 Código del frontend. Listado de actividades creadas de un curso.

```
29.  <tr>
30.    <th>T&iacute;tulo</th>
31.    <th>Curso</th>
32.    <th>Descripci&acute;n</th>
33.    <th>Retroalimentaci&acute;n</th>
34.    <th>Soluci&acute;n T&eacute;cnica</th>
35.    <th class="col-2">Acciones</th>
36.  </tr>
37. </thead>
38. <tbody>
39.  {% for n_activity in n_activities %}
40.  <tr>
41.    <td>{{ n_activity.title }}</td>
42.    <td>{{ n_activity.name }}</td>
43.    <td>{{ n_activity.description }}</td>
44.    <td>
45.      <strong>
46.        <i>{{ n_activity.place }}</i>
47.      </strong>
48.    </td>
49.    <td>{{ n_activity.tecsol }}</td>
50.    <td>
51.      {% if n_activity.data=="[]" %}
52.        <a href="/public/GoJS-
53. master/projects/pdf/diagrama.php?mode=edit&table=model_diagram_success&id={{ 
54. n_activity.idssuccess }}" title="Crear soluci&on"><i class="far fa-lightbulb"></i></a>
55.      </a>
56.      {% else %}
57.        <a href="/public/GoJS-
58. master/projects/pdf/diagrama.php?mode=view&table=model_diagram_success&id={{ 
60. n_activity.idssuccess }}" title="Ver soluci&on"><i class="fa fa-lightbulb"></i></a>
61.      </a>
62.      {% endif %}
63.      {% if is_granted('ROLE_ADMIN') %}
64.        <a href="{{ path('app_n_activity_edit', {'id': n_activity.id, 'idcurso': 
65. n_activity.curso}) }}" title="Editar">
66.          <i class="fa fa-edit"></i>
67.        </a>
68.        <a href="{{ path('app_xapi_activity', {'id': n_activity.id, 'course': 
69. n_activity.name}) }}" title="Descargar">
70.          <i class="fa fa-briefcase"></i>
71.        </a>
72.        <a href="{{ path('app_n_activity_delete', {'id': n_activity.id, 'idcurso': 
73. n_activity.curso}) }}" title="Borrar">
74.          <i class="fa fa-trash"></i>
75.        </a>
76.      {% endif %}
77.    </td>
78.  </tr>
79.  {% else %}
80.  <tr>
81.    <td colspan="3">no hay datos</td>
82.  </tr>
83.  {% endfor %}
84. </tbody>
```

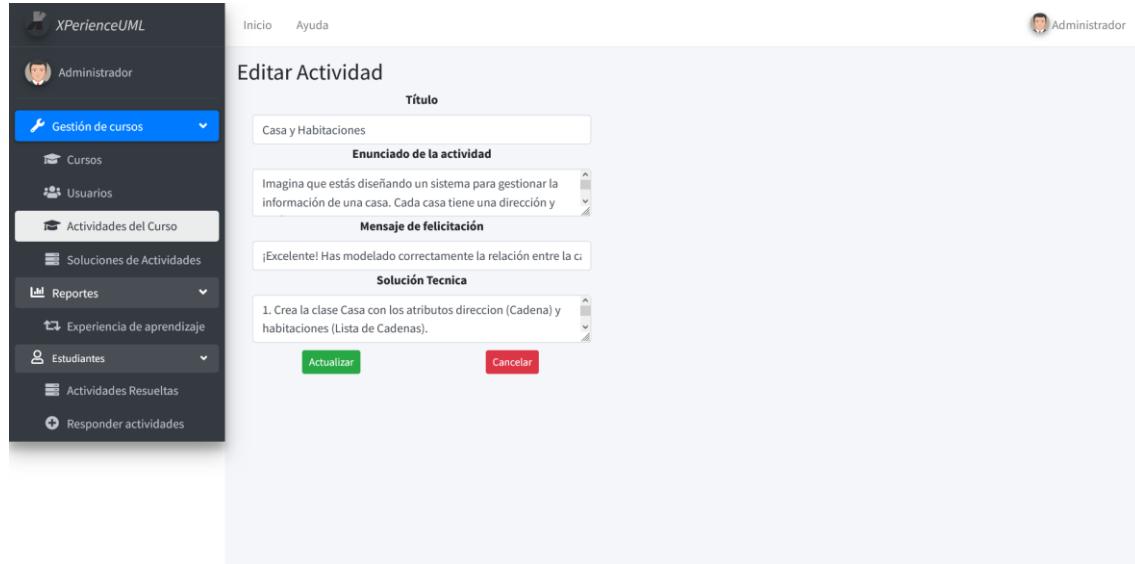
Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Cuando se listan las actividades de un curso, entre las acciones que pueden realizarse se encuentran, *Ver* o *Crear* solución, *Editar* la actividad, *Descargar* con empaquetado Cmi5 y *Eliminar* actividad. Las acciones de *Ver* o *Crear* solución se dejará para analizar en la sección

de Gestión de diagramas UML, debido a que están relacionadas precisamente con los diagramas UML.

Cuando se selecciona Editar una actividad se muestra la página de la Figura 51.

Figura 51 Editar actividad.



Fuente: (Macas, 2023)

En este paso puede modificarse el título de una actividad, el enunciado del ejercicio, el mensaje de felicitación que va a ser empleado como retroalimentación y la solución técnica que va a proveer el docente para la realización de la actividad. Las acciones para esta vista son *Actualizar* para llevar a cabo la modificación de los campos, y *Cancelar* para deshacer las modificaciones.

Figura 52 Código del frontend. Editar o actualizar una actividad.

```
3. {{ form_start(form)  }}
4. <div class="container-fluid">
5.     <div class="row">
6.         <div class=" col-sm-12">
7.             {{form_row(form.title)  }}
8.         </div>
9.     </div>
10.    <div class="row">
11.        <div class="col-sm-12">
12.            {{form_row(form.description )  }}
13.        </div>
14.        <div class="col-sm-12">
15.            {{form_row(form.place )  }}
16.        </div>
17.    </div>
18.    <div class="row">
19.        {{form_row(form.nta)  }}
20.    </div>
21.    <div class="row">
22.        <div class="col-sm-12">
23.            {{form_row(form.tecsol)  }}
24.        </div>
25.    </div>
26. </div>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 53 Código del backend. Actualización de la información de una actividad.

```
120. /**
121. * @Route("/{id}/edit/{idcurso}", name="app_n_activity_edit", methods={"GET",
122. "POST"})
123. */
124. public function edit(Request $request, NActivity $nActivity,
125. NActivityRepository $nActivityRepository, int $idcurso): Response
126. {
127.     $form = $this->createForm(NActivityType::class, $nActivity);
128.     $form->handleRequest($request);
129.     if ($form->isSubmitted() && $form->isValid()) {
130.         $em = $this->getDoctrine()->getManager();
131.         /** @var NTA $interactiva */
132.         $interactiva = $em->getRepository('App:NTA')-
133. >findBy(['denomination' => 'Interactiva']);
134.         $nActivity->setNTA($interactiva[0]);
135.         $nActivityRepository->add($nActivity);
136.         $this->addFlash('success', 'Datos actualizados satisfactoriamente
137. !!!');
138.
139.         $course = $em->getRepository('App:Course')->findOneBy(array('id' =>
140. $idcurso));
141.         return $this->render('n_activity/index.html.twig', [
142.             'n_activities' => $nActivityRepository-
143. >findDataToShowIdSuccessByCourse($idcurso),
144.             'title' => 'Actividades del curso '. $course->getName(),
145.         ]);
146.     }
147.
148.     return $this->render('n_activity/edit.html.twig', [
149.         'n_activity' => $nActivity,
150.         'form' => $form->createView(),
151.     ]);
152. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En la actualización de los datos de una actividad mediante el botón *Actualizar*, se emplea la función *edit* del fichero *NActivityController.php*, la que se encarga de editar una actividad existente (*NActivity*) asociada a un curso específico. En ella se define una ruta */{id}/edit/{idcurso}* para esta acción, donde *id* es el ID de la actividad y *\$idcurso* es el ID del curso. La ruta acepta tanto métodos GET como POST. Los parámetros que se tienen son:

- Request \$request: Objeto que representa la solicitud HTTP.
- NActivity \$nActivity: Entidad NActivity a editar.
- NActivityRepository \$nActivityRepository: Repositorio para gestionar actividades.
- int \$idcurso: ID del curso asociado.

Mientras que devuelve una respuesta HTTP.

En la implementación, primeramente se crea un formulario basado en la clase *NActivityType* y se le vincula a la actividad existente. Se procesa los datos enviados en la solicitud HTTP.

Se comprueba si el formulario ha sido enviado y si los datos son válidos. Entonces se obtiene el entity manager. Se busca una entidad *NTA* con la denominación 'Interactiva' y la asigna a la actividad. Se actualiza la actividad existente, guardando los cambios. Se muestra un mensaje flash de éxito. Finalmente se obtiene el curso correspondiente al ID proporcionado y se redirige a la página de lista de actividades del curso.

Si el formulario contiene datos no validos o erróneos se renderiza la plantilla de edición de actividad.

Otra de la acción que pueden ejecutarse desde la vista de los datos de un curso, es ver las actividades realizadas. Cuando uno se refiere a las actividades realizadas, se está haciendo alusión a las actividades que han sido desarrolladas por los estudiantes. Con esta acción solo podrá listarse las actividades con su nombre, el curso al que pertenece, los nombres de los estudiantes que han dado una solución, así como la fecha en que fue dada la solución. La única

acción permitida para esta vista es *Ver* que se refiere al diagrama UML desarrollado por los estudiantes.

Figura 54 Actividades realizadas de un curso

The screenshot shows the XPerienceUML application interface. The left sidebar has a dark theme with white icons and text. It includes sections for 'Gestión de cursos' (Courses), 'Reportes' (Reports), and 'Estudiantes' (Students). Under 'Estudiantes', there are 'Actividades Resultadas' (Resulting Activities) and 'Responder actividades' (Answer activities). The main content area is titled 'Diagramas Realizados por Estudiantes' (Diagrams made by Students). It features a table with columns: 'Creado por' (Created by), 'Fecha' (Date), 'Curso' (Course), 'Actividad' (Activity), and 'Acciones' (Actions). The table contains 10 rows, each representing an activity created by 'Estudiante 18' on June 18, 2024, at various times between 04:38:41 and 04:44:33. The activities are all related to the course 'El mundo de la programación orientada a objetos' (The world of object-oriented programming) and are categorized under 'Sistema de Notificaciones' (Notification System) and 'Departamento y Empleados' (Department and Employees). At the bottom, it says 'Mostrando página 1 de 20' (Showing page 1 of 20) and has a navigation bar with pages 1 through 20 and 'Próximo' (Next).

Creado por	Fecha	Curso	Actividad	Acciones
Estudiante 18	2024-06-18 04:38:41	El mundo de la programación orientada a objetos	Sistema de Notificaciones	
Estudiante 18	2024-06-18 04:38:54	El mundo de la programación orientada a objetos	Sistema de Notificaciones	
Estudiante 18	2024-06-18 04:41:30	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:41:45	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:42:05	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:42:13	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:42:25	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:42:38	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:42:47	El mundo de la programación orientada a objetos	Departamento y Empleados	
Estudiante 18	2024-06-18 04:44:33	El mundo de la programación orientada a objetos	Departamento y Empleados	

Fuente: (Macas, 2023)

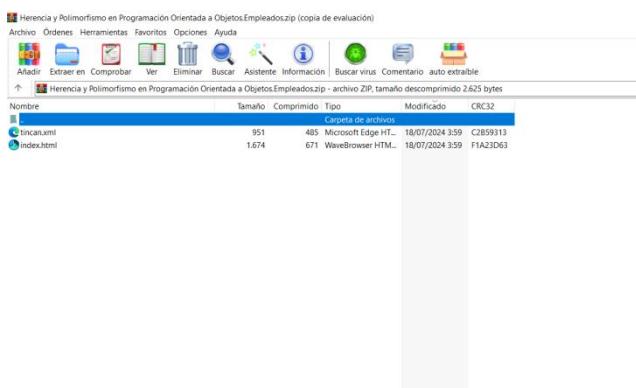
Figura 55 Código del frontend de para ver listado de actividades realizadas de un curso.

```
24. <thead><thead>
25.   <tr>
26.     <th>Creado por</th>
27.     <th>Fecha</th>
28.     <th>Curso</th>
29.     <th>Actividad</th>
30.     <th>Acciones</th>
31.   </tr>
32. </thead>
33. <tbody>
34.   {% for model_diagram_test in model_diagram_tests %}<br/>
35.   <tr>
36.     <td><strong><i>{{ model_diagram_test.action }}</i></strong></td>
37.     <td><strong><i>{{ model_diagram_test.Fecha }}</i></strong></td>
38.     <td><strong><i>{{ model_diagram_test.course }}</i></strong></td>
39.     <td><strong><i>{{ model_diagram_test.nactivity }}</i></strong></td>
40.     <td>
41.       <a href="/public/GoJS-
42. master/projects/pdf/diagrama.php?mode=view&table=model_diagram_test&id={{
43. model_diagram_test.id }}" title="Ver">
44.         <i class="fa fa-eye"></i>
45.       </a>
46.     </td>
47.   </tr>
48.   {% else %}
49.   <tr>
50.     <td colspan="6">no hay datos</td>
51.   </tr>
52. {% endfor %}
53. </tbody>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En el listado de actividades, la opción *Guardar*, permite descargar lo datos de una actividad en un fichero **ZIP**, mediante empaquetado Cmi5.

Figura 56 Fichero descargado con la información de una actividad



Fuente: (Macas, 2023)

Figura 57 Código del backend para crear el fichero .zip de una actividad de un curso.

```
141. /**
142. * Método que crea el archivo ZIP
143. * @param array $files
144. * @param string $destination
145. * @param boolean $overwrite
146. */
147. public function createZipArchive($files = array(), $destination = '',
148. $overwrite = false)
149. {
150.     if (file_exists($destination) && !$overwrite) {
151.         return false;
152.     }
153.     $validFiles = array();
154.     if (is_array($files)) {
155.         foreach ($files as $file) {
156.             if (file_exists($file)) {
157.                 $validFiles[] = $file;
158.             }
159.         }
160.     }
161.     if (count($validFiles)) {
162.         $zip = new ZipArchive();
163.         if ($zip->open($destination, $overwrite ? ZIPARCHIVE::OVERWRITE :
164. ZIPARCHIVE::CREATE) == true) {
165.             foreach ($validFiles as $file) {
166.                 $zip->addFile($file, $file);
167.             }
168.             $zip->close();
169.             return file_exists($destination);
170.         } else {
171.             return false;
172.         }
173.     } else {
174.         return false;
175.     }
176. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la creación y después descarga del fichero con empaquetado CMI5 se emplea el fichero *XApiController.php*, que es una clase controladora de Symfony. En ella se encuentran varias funciones que permiten crear ficheros ZIP, HTML; convertir una declaración en un objeto Json y/o en un objeto pdf; entre otras. Para el correcto funcionamiento de estas funciones se hace uso de los módulos *ZipArchive* y *DOMDocument* de PHP.

El nombre del fichero ZIP que va a crearse para ser descargado coincidirá con el ‘nombre del curso’.’nombre de la actividad’. El fichero ZIP estará compuesto por dos ficheros que también serán creados en la clase controladora: *index.html* y *tincan.xml*. Ambos contendrán el nombre de la actividad y su descripción. (Ver Figura 56)

El código presentado en la define una función llamada *createZipArchive* que crea un archivo ZIP a partir de un conjunto de archivos especificados. La función verifica la existencia del archivo de destino, valida los archivos de entrada y utiliza la clase *ZipArchive* para crear y llenar el archivo ZIP.

Esta función acepta tres parámetros que son:

- *\$files*: Un array de rutas de archivos a incluir en el ZIP.
- *\$destination*: La ruta del archivo ZIP de salida.
- *\$overwrite*: Un booleano que indica si se debe sobrescribir el archivo ZIP si ya existe.

En la implementación se verifica la existencia del archivo de destino. Si el archivo de destino existe y *\$overwrite* es falso, devuelve false para indicar que no se puede crear el archivo.

Luego se valida los archivos de entrada. Para ellos se crea un array *\$validFiles* para almacenar los archivos válidos. Itera sobre el array *\$files* y se verifica la existencia de cada archivo, agregándolo al array *\$validFiles* si existe. Si la cantidad de ficheros añadidos al array *\$validFiles* es diferente de cero entonces se crea un fichero ZIP y se abre en modo de sobreescritura o creación, en dependencia de los especificado en el parámetro *\$overwrite*, cada uno de los ficheros, retornándose el mismo. En caso de que el fichero se cree pero no pueda abrirse o la cantidad de archivos válidos sea cero la función retorna falso.

Figura 58 Código del backend para editar los datos una actividad de un curso.

```
120.      /**
121.       * @Route("/{id}/edit", name="app_n_activity_edit", methods={"GET",
122.       "POST"})
123.       */
124.      public function edit(Request $request, NActivity $nActivity,
125.      NActivityRepository $nActivityRepository): Response
126.      {
127.          $form = $this->createForm(NActivityType::class, $nActivity);
128.          $form->handleRequest($request);
129.
130.          if ($form->isSubmitted() && $form->isValid()) {
131.              $em = $this->getDoctrine()->getManager();
132.              /** @var NTA $interactiva */
133.              $interactiva = $em->getRepository('App:NTA')-
134.              >findBy(['denomination' => 'Interactiva']);
135.              $nActivity->setNTA($interactiva[0]);
136.              $nActivityRepository->add($nActivity);
137.              $this->addFlash('success', 'Datos actualizados
138. satisfactoriamente !!!');
139.              return $this->render('n_activity/index.html.twig', [
140.                  'n_activities' => $nActivityRepository->findDataToShow(),
141.                  'title' => 'Actividades del curso',
142.              ]);
143.          }
144.
145.          return $this->render('n_activity/edit.html.twig', [
146.              'n_activity' => $nActivity,
147.              'form' => $form->createView(),
148.          ]);
149.      }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código presentado en la Figura 58 es una función de controlador Symfony que se encarga de editar una actividad existente (*NActivity*). En ella se define una ruta */id/edit* para esta acción, donde *id* es un parámetro dinámico que representa el ID de la actividad. La ruta acepta tanto métodos GET como POST. Los parámetros que presenta son:

- Request *\$request*: Objeto que representa la solicitud HTTP.
- NActivity *\$nActivity*: Entidad NActivity a editar.
- NActivityRepository *\$nActivityRepository*: Repositorio para gestionar actividades.

Mientras que devuelve una respuesta HTTP.

En la implementación, primeramente se crea un formulario basado en la clase *NActivityType* y se le vincula a la actividad existente. Se procesa los datos enviados en la solicitud HTTP.

Luego se comprueba si el formulario ha sido enviado y si los datos son válidos. Entonces se obtiene el entity manager. Se busca una entidad NTA con la denominación 'Interactiva' y la

asigna a la actividad. Se actualiza la actividad guardando los cambios. Se muestra un mensaje flash de éxito. Finalmente se redirige a la página de lista de actividades de todos los cursos.

Si el formulario no es válido o contiene datos erróneos se renderiza la plantilla de edición de actividad.

Mediante la opción *Borrar*, se borra el registro de la actividad de la base de datos y solo puede ser llevada a cabo por un Docente.

Figura 59 Código del backend de borrar la solución de una actividad de un curso.

```
148.      /**
149.      * @Route("/{id}/delete/{idcurso}", name="app_n_activity_delete",
150.      methods={"POST", "GET"})
151.      */
152.      public function delete(Request $request, NActivity $nActivity,
153.      NActivityRepository $nActivityRepository, int $idcurso): Response
154.      {
155.          if (!$this->asocActividad($nActivity) ){
156.              $nActivityRepository->remove($nActivity);
157.              $this->addFlash('success', 'Datos eliminados satisfactoriamente
158.              !!!');
159.          } else
160.          {
161.              $nActivityRepository->deleteDiagramAssocTest($nActivity->getId());
162.              $nActivityRepository->deleteDiagramAssocSucce($nActivity-
163. >getId());
164.              $nActivityRepository->remove($nActivity);
165.              $this->addFlash('success', 'Datos eliminados
166. satisfactoriamente!!!!');
167.
168.          }
169.          $em = $this->getDoctrine()->getManager();
170.          $course = $em->getRepository('App:Course')->findOneBy(array('id' =>
171. $idcurso));
172.          return $this->render('n_activity/index.html.twig', [
173.              'n_activities' => $nActivityRepository-
174. >findDataToShowIdSuccessByCourse($idcurso),
175.              'title' => 'Actividades del curso '. $course->getName(),
176.          ]);
177.      }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para borrar una actividad se emplea la función *delete* de la clase controladora *NActivityController.php*, que se encarga de eliminar una actividad existente (*NActivity*), y que define una ruta */id/delete/{idcurso}* para esta acción, donde *id* es el ID de la actividad y *idcurso* es el ID del curso. La ruta acepta tanto métodos GET como POST. Los parámetros que recibe son:

- Request \$request: Objeto que representa la solicitud HTTP.
- NActivity \$nActivity: Entidad NActivity a eliminar.
- NActivityRepository \$nActivityRepository: Repositorio para gestionar actividades.
- int \$idcurso: ID del curso asociado.

Mientras que devuelve una respuesta HTTP.

En la implementación, inicialmente se verifica asociaciones de la actividad. Para ellos se llama al método *asocActividad* para determinar si la actividad tiene asociaciones. Si no tiene asociaciones, se elimina la actividad directamente usando *\$nActivityRepository->remove()*. Si tiene asociaciones, llama a métodos para eliminar las asociaciones relacionadas (*deleteDiagramAssocTest* y *deleteDiagramAssocSuccess*) y luego elimina la actividad. Luego se muestra un mensaje flash de éxito.

Finalmente se obtiene el curso correspondiente al ID proporcionado y se redirige a la página de la lista de actividades del curso.

Solo los docentes podrán ejecutar esta acción.

## Gestión de diagramas UML

Una vez creadas las actividades, estas pueden clasificarse como actividades creadas y actividades a realizar. Cuando se hable de los diagramas de las actividades creadas, se refiere a las soluciones propuestas por los docentes y se almacenan en la tabla *model\_diagram\_success*.

Mientras que cuando se hable de los diagramas de las actividades realizadas o a realizar, se refiere a los diagramas realizados por los estudiantes que se evaluarán midiéndose con las soluciones de los docentes. Los diagramas de los estudiantes son almacenados en la tabla *model\_diagram\_test*. Para cada actividad, el docente solo tendrá una propuesta de solución, mientras que los estudiantes si podrán tener más de una solución a cada actividad.

La gestión de diagramas está íntimamente relacionada con el editor de diagramas UML. Para el manejo de los diagramas se emplea la biblioteca GoJS, la cual es implementada en JavaScript y

permite la creación de diagramas interactivos del lado del cliente. Con esta biblioteca los diagramas solo necesitan ser guardados y/o restaurados.

Para la gestión de diagramas se emplean varios ficheros, siendo el fichero principal y conductor de toda la gestión *diagrama.php*. En este fichero se tiene la mayoría del frontend del editor y algunas funcionalidades del backend. Otros de los ficheros que se emplean es *uml\_diagram.js*, donde se hace uso de la biblioteca GoJS, personalizada para crear los diagramas UML, con todas las funcionalidades que se necesitan para ello como son crear clase, adicionar y eliminar métodos y propiedades a las clases, establecer las relaciones entre las clases, así como salvar los diagramas, reportar los verbos Cmi5, realizar modificaciones de nombre, posicionamiento y demás en los diagramas. Otros ficheros como *uml\_json\_comparar.php* complementan las funcionalidades necesarias para el editor. En estos ficheros puede verse la combinación de código PHP y JavaScript para manejar tanto la lógica del servidor como la interacción del cliente. Utiliza condicionales en PHP para creación dinámica del contenido, ajustando la habilitación del mismo.

Cuando se invoca al editor en el URL se le pasan parámetros en dependencia de lo que vaya a hacerse, es decir, si se va a crear, visualizar o modificar una solución. Entre los parámetros se pasa en modo (*mode*), la tabla de la base de datos en la que se va a trabajar (*table*) y el identificador del diagrama (*id*).

Los modos que se utilizan son ‘view’ y ‘edit’. Las tablas con las que trabajan son ‘*model\_diagram\_success*’ para el trabajo de las soluciones de los docentes y ‘*model\_diagram\_test*’ para el trabajo de las soluciones de los estudiantes, como se dijo anteriormente. Veamos unos ejemplos.

Figura 60 Código del frontend para ver o editar una solución docente de actividad.

```
51. <td>
52.   {% if n_activity.data=="[]" %}
53.     <a href="/public/GoJS-
54. master/projects/pdf/diagrama.php?mode=edit&table=model_diagram_success&id={{ 
55. n_activity.idsucces }}" title="Crear solución"><i class="far fa-
56. lightbulb"></i></a>
57.   </a>
58.   {% else %}
59.     <a href="/public/GoJS-
60. master/projects/pdf/diagrama.php?mode=view&table=model_diagram_success&id={{ 
61. n_activity.idsucces }}" title="Ver solución"><i class="fa fa-lightbulb"></i></a>
62.   </a>
63.   {% endif %}
64.   {% if is_granted('ROLE_ADMIN') %}
65.     <a href="{{ path('app_n_activity_edit', {'id': n_activity.id, 'idcurso': 
66. n_activity.curso}) }}" title="Editar">
67.       <i class="fa fa-edit"></i>
68.     </a>
69.     <a href="{{ path('app_xapi_activity', {'id': n_activity.id, 'course': 
70. n_activity.name}) }}" title="Descargar">
71.       <i class="fa fa-briefcase"></i>
72.     </a>
73.     <a href="{{ path('app_n_activity_delete', {'id': n_activity.id, 'idcurso': 
74. n_activity.curso}) }}" title="Borrar">
75.       <i class="fa fa-trash"></i>
76.     </a>
77.   {% endif %}
78.
79. </td>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En la Figura 60 se está estableciendo un enlace al editor en dependencia de si existe la solución del ejercicio o no (`n_activity.data=="[]"`). Recuérdese que siempre que se crea una actividad, se crea una solución vacía para los docentes que después puede ser editada. Si no existe la solución (`n_activity.data!="[]"`), se va invocar el editor en modo de edición (`mode=edit`), de lo contrario se invoca en modo de visualización (`mode=view`). Como se está hablando de una solución docente se carga la tabla correspondiente (`table=model_diagram_success`). Por último se establece el identificador del diagrama, con el cual se podrá obtener luego el nombre de la actividad y el nombre del curso.

Figura 61 Código del frontend de para ver listado de actividades realizadas de un curso. Acción ver solución.

```
41. <a href="/public/GoJS-
42. master/projects/pdf/diagrama.php?mode=view&table=model_diagram_test&id={{ 
43. model_diagram_test.id }}" title="Ver">
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Este segundo ejemplo, es un fragmento de la visualización del listado de actividades realizadas de un curso. Cuando se habla de actividades realizadas se hace referencia a las actividades que los estudiantes respondieron y que no tendrán posibilidades de editarse. De ahí que solo pueda establecerse el modo de visualización. Por lo que se han establecidos como parámetros `mode=view&table=model_diagram_test`. Por último se establece el identificador del diagrama.

*Figura 62 Código del backend del editor de diagramas.*

```

47. <?php
48.     //Aqui se selecciona si se va a agregar o a evaluar el diagrama.
49.     $buttonText = 'Evaluar Soluci&on';
50.     $modalType = "buttonCheck";
51.     if (isset($_GET["mode"])) && ($_GET["mode"] != 'undefined')){
52.         if (($_GET["mode"] == 'view') || ($_GET["mode"] == 'edit'))
53.     {
54.         $buttonText = ($_GET["mode"] == 'edit') ? 'Guardar cambios' : '';
55.         if ($_GET["mode"] == 'edit')
56.         {
57.             $modalType = "buttonInsert";
58.             echo '<pre hidden="true" id="modeEdit" value="true"></pre>
59.         ';
60.     }
61.     else
62.     {
63.         echo '<pre hidden="true" id="modeEdit" value="false"></pre>';
64.     }
65.     if (isset($_GET["table"])) && ($_GET["table"] != 'undefined'))
66.     {
67.         $table = $_GET["table"];
68.         if (isset($_GET["id"])) && ($_GET["id"] != 'undefined'))
69.         {
70.             $id = $_GET["id"];
71.             $conexion = mysqli_connect('localhost', 'xperienc',
72. 'Zb(nY;w1A62r1R', 'xperienc_uml');
73.             $result = mysqli_query($conexion, "select data from
74. ".$table." where id='".$id."'");
75.             $diagram = mysqli_fetch_array($result);
76.             if (isset($diagram))
77.             {
78.                 if (sizeof($diagram) > 0){
79.                     $data = (string)$diagram[0];
80.                     $conexion->close();
81.                     echo '<pre hidden="true"
82. id="dataJsonDiagram">' . $data . '</pre>';
83.                 }
84.             }
85.         }
86.     }
87. }
88. }
89. ?>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En el fragmento anterior, se establece si el editor va a ser usado para editar, ver o modificar una solución.

El código PHP presentado en la Figura 62, presente en fichero *diagram.php*, es parte de un sistema que maneja diagramas, permitiendo tanto la creación como la edición de los mismos. El código se encarga de determinar el modo de operación (crear, ver o editar) basado en los parámetros de la URL, recuperar datos del diagrama si es necesario y preparar el entorno para el siguiente paso.

Primeramente, se inicializan las variables *\$buttonText* y *\$modalType* con valores predeterminados para el modo de creación. Luego se verifica si existe el parámetro *mode* en la URL y si es diferente de 'undefined'. Si el modo es 'view' o 'edit', se modifica el texto del botón y el tipo de modal según el modo. Si el modo es 'edit', se crea un elemento oculto *modeEdit* con valor 'true'. Si el modo es 'view', crea un elemento oculto *modeEdit* con valor 'false'. Si existen los parámetros *table* e *id*, se recuperan los datos del diagrama desde la base de datos y se almacenan en la variable *\$data*.

Este fragmento de código está construyendo parte del frontend de la aplicación. Luego veremos en otra parte del código de *diagrama.php*, reflejado en la Figura 63, como esos valores van a formar parte de uno de los botones presentes en la aplicación.

Figura 63 Código del frontend del editor de diagramas. Botones del editor.

```
212.     echo '<button type="button" class="btn btn-success btn-sm"';
213.     name="'. $modalType.'" id="'. $modalType.'" style="margin-left:4px">';
214.     echo '<i class="bi bi-arrow-bar-up"></i>';
215.     echo '<strong>'. $buttonText.'</strong>';
216.     echo '</button>';
217.
218.     echo '<button name = "AregarClases" id = "AregarClases" class="btn';
219.     btn-info btn-sm" onclick="addNewClass(0, 0)" '. $a.' style="margin-
220.     left:4px">';
221.     echo '<i class="bi bi-window"></i>';
222.     echo '<strong>Aregar clase</strong>';
223.     echo '</button>';
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para editar una solución de una actividad escogida, uno puede crear clases, editar los nombres de las clases, de las propiedades y los métodos de las clases, así como establecer como privados, públicos o protegido los métodos y propiedades; y darle un tipo específico a esos métodos y atributos. También se pueden establecer relaciones entre las clases, las cuales además pueden ser de diferentes tipos.

## Gestión de diagramas para Docentes

La gestión de diagramas para los docentes se basa en la visualización, creación, modificación y eliminación de diagramas. Solo está permitido para usuarios con rol Docente la modificación y eliminación.

La gestión de los diagramas para los Docentes puede iniciarse desde que se crea la actividad eligiendo la opción de *Guardar y Adicionar solución*. Si solo se escoge *Guardar*, luego en el listado de actividades se puede ejecutar la acción *Crear solución*. Esto es, opción de **Gestión de curso**, apartado *Actividades del curso*. La acción de *Crear solución* solo aparecerá si no se ha creado con anterioridad la solución, de lo contrario en su lugar aparecerá acción *Ver solución*.

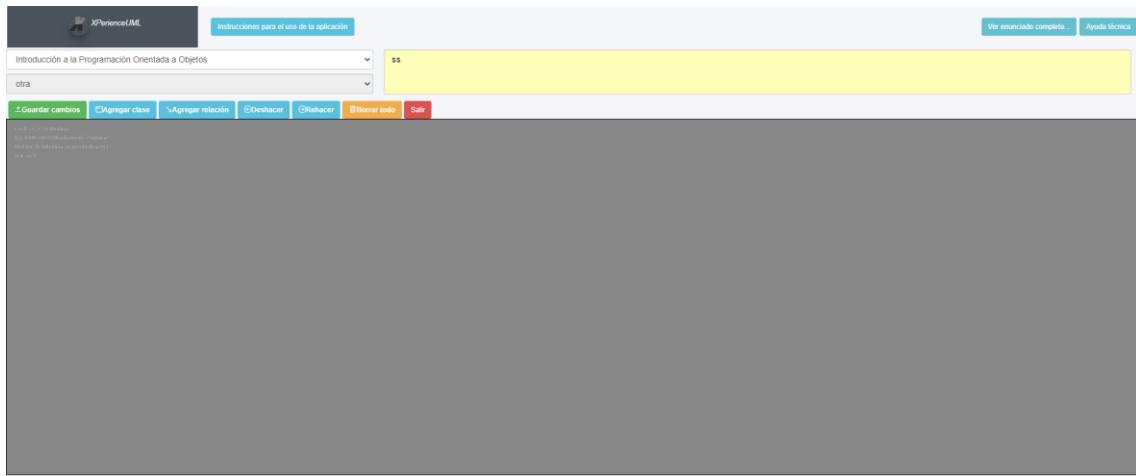
Figura 64 Listado de actividades de cursos con las opciones de Crear o Ver la solución de una actividad.

Título	Curso	Enunciado	Retroalimentación	Solución Técnica	Acciones
Nueva Actividad	Introducción a la Programación Orientada a Objetos	Enunciado de la actividad. Se describe el ejercicio a realizar	Mensaje de felicitación	Pistas para resolver el ejercicio. Detalles a tener en cuenta.	 Crear
Clases y Objetos	Introducción a la Programación Orientada a Objetos	Los automóviles tienen marcas y modelos, y tienen una funcionalidad arrancar. Representa esto en una clase	¡Buen trabajo! Has creado tu primera clase	Crea una clase llamada Automovil. Agrega un atributo marca de tipo cadena. Agrega un atributo modelo de tipo cadena. Agrega un método arrancar de tipo vacío	 Ver
Departamento y Empleados	El mundo de la programación orientada a objetos	En una empresa, un departamento está compuesto por varios empleados. Cada empleado tiene un nombre y un puesto específico, y pertenece a un único departamento. Sin embargo, un departamento puede existir independientemente de los	¡Excelente! Has modelado correctamente la relación entre el departamento y los empleados.	1. Crea la clase Departamento con los atributos nombre (Cadena) y empleados (Lista de Cadena). 2. En la clase Departamento, añade los métodos agregarEmpleado (Void) y eliminarEmpleado (Void). 3. Crea la clase Empleado con los atributos nombre (Cadena) y puesto (Cadena). 4. Establece que Departamento	

Fuente: (Macas, 2023)

Como se explicó en el apartado de *Gestión de actividades*, cuando se crea una actividad, se crea para la actividad un diagrama con solución vacía. De ahí que la cuando hablemos de crear un diagrama lo que estamos haciendo realmente es editar un diagrama con solución vacía. Por tanto no van a diferenciarse en nada las opciones crear diagrama y modificar diagrama. Luego nos referiremos a esta operación como editar solución.

Figura 65 Vista del editor para la edición de la solución por parte de los Docentes



Fuente: (Macas, 2023)

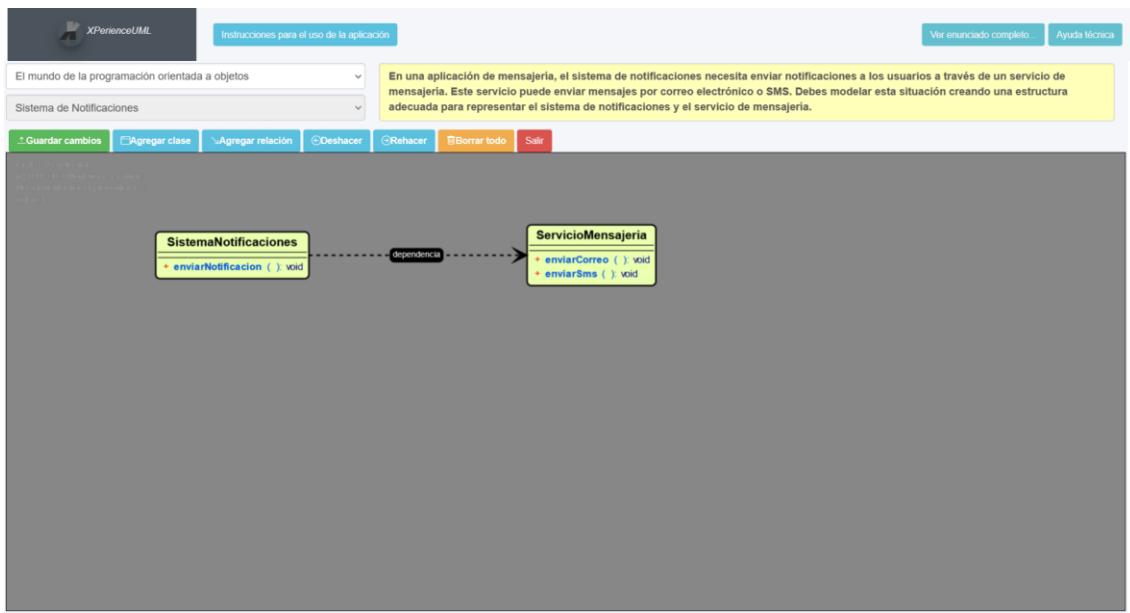
Figura 66 Código del frontend del editor de diagramas.

```
18. <body>
19.     <div>
20.         <section >
21.             <div class="row" style="background-color:#f4f6f9; margin-left:
22. 5px;">
23.                 <div class="col-md-2" style="color:white;background-
24. color:#4b545c;width:100%; text-align: center;display: grid;place-items: center;">
25.                     <a href="php echo $ruta;?&gt;" class="brand-link"&gt;
26.                         &lt;img
27. src="/public/dist/img/XperienceUML.png" alt="AdminLTE Logo" class="brand-image
28. img-circle elevation-3" style="opacity: .8"&gt;
29.                         &lt;span class="brand-text font-weight-
30. light" style="color:white"&gt;
31.                             &lt;i&gt;XPerienceUML&lt;/i&gt;
32.                         &lt;/span&gt;
33.                     &lt;/a&gt;
34.                 &lt;/div&gt;
35.             &lt;div class="col-md-10" style="width:100%"&gt;</pre
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Cuando se elige *Editar* solución se muestra también el curso, la actividad y la descripción de la actividad sin posibilidad de modificación, ya que estas vienen establecidas desde la creación de la actividad. Mientras que el diagrama si podrá modificarse y/o actualizarse. (Ver Figura 67)

Figura 67 Gestión de Diagrama. Editar Diagrama



Fuente: (Macas, 2023)

Para *Editar* una solución se cuenta con los botones **Agregar clase**, **Agregar relación**, **Deshacer**, **Rehacer**, **Borrar todo** y **Guardar cambios**.

Figura 68 Código del frontend del editor de diagramas. Botones del editor.

```
212.     <?php
213.         if (!isset($_GET["table"])) {
214.             echo '<button name="botonIniciar" id="botonIniciar" class="btn
215. btn-primary btn-sm" disabled
216. onclick="lanzar(this.name,\'AregarRel,AregarClases,Deshacer,Rehacer,
217. Borrar,Abandonar,verDescription,' . $modalType . ',Ayuda\')>';
218.             echo '<strong>Iniciar</strong>';
219.             echo '</button>';
220.         }
221.         if ($buttonText!='')
222.         {
223.             $a='';
224.             if (!isset($_GET["id"])) $a="disabled";
225.             echo '<button type="button" class="btn btn-success btn-sm"
226. name="'.$modalType.'" id="'.$modalType.'" style="margin-left:4px"
227. '.$a.'>';
228.             echo '<i class="bi bi-arrow-bar-up"></i>';
229.             echo '<strong>'.$buttonText.'</strong>';
230.             echo '</button>';
231.
232.             echo '<button name = "AregarClases" id = "AregarClases"
233. class="btn btn-info btn-sm" onclick="addNewClass(0, 0)" '.$a.'
234. style="margin-left:4px">';
235.             echo '<i class="bi bi-window"></i>';
236.             echo '<strong>Aregar clase</strong>';
237.             echo '</button>';
238.
239.             echo '<button name = "AregarRel" id = "AregarRel" class="btn
240. btn-info btn-sm" onclick="addNewLink(0, 0)" '.$a. ' style="margin-
241. left:4px">';
242.             echo '<i class="bi bi-arrow-down-right"></i>';
243.             echo '<strong>Aregar relaci&on</strong>';
244.             echo '</button>';
245.
246.             echo '<button name = "Deshacer" id = "Deshacer" class="btn
247. btn-info btn-sm" onclick="undoDiagramChanges()" '.$a.' style="margin-
248. left:4px">';
249.             echo '<i class="bi bi-arrow-left-circle"></i>';
250.             echo '<strong>Deshacer</strong>';
251.             echo '</button>';
252.
253.             echo '<button name = "Rehacer" id = "Rehacer" class="btn btn-
254. info btn-sm" onclick="redoDiagramChanges()" '.$a.' style="margin-
255. left:4px">';
256.             echo '<i class="bi bi-arrow-right-circle"></i>';
257.             echo '<strong>Rehacer</strong>';
258.             echo '</button>';
259.
260.             echo '<button name = "Borrar" id = "Borrar" class="btn btn-
261. warning btn-sm" onclick="clearDiagram()" '.$a.' style="margin-
262. left:4px">';
263.             echo '<i class="bi bi-trash"></i>';
264.             echo '<strong>Borrar todo</strong>';
265.             echo '</button>';
266.
267.         }
268.         if ( ! isset($_GET["table"])){
269.             if (!isset($_GET["id"])) { $a= "disabled"; } else { $a=''; };
270.             echo '<button name="Abandonar" id="Abandonar" class="btn btn-
271. danger btn-sm" onclick="abandonar()" '.$a.' style="margin-left:4px">';
272.             echo '<strong>Abandonar ejercicio</strong>';
273.             echo '</button>';

```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código presentado en la Figura 68, muestra la lógica para generar botones de control para el editor de diagramas. Los botones se crean dinámicamente dependiendo de la existencia de parámetros en la URL (`$_GET`). El código utiliza variables como `$buttonText` y `$a` que se

definen en previo a su utilización, como se vio, por ejemplo, en la Figura 62. La mayoría de los botones se deshabilitan si no hay un ID de diagrama (`$_GET["id"]`) definido. Por otra parte, se invocan las funciones Javascript como `lanzar`, `addNewClass`, `addNewLink`, `undoDiagramChanges`, `redoDiagramChanges` y `abandonar` que manejan la funcionalidad de cada botón.

Figura 69 Código del backend del editor de diagramas. Guardar solución.

```
450.     $("#buttonInsert").click(function() {
451.         $("#modalInsert").modal("show");
452.     });
453.     });
454.     $("#modalInsert").on('show.bs.modal', function() {
455.         diagram_insert(
456.             document.getElementById('course').value,
457.             document.getElementById('activity').value,
458.             document.getElementById('modeEdit').value
459.         );
460.     });
461. });


```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código que se muestra en la Figura 69 utiliza Javascript y jQuery para manejar el modal de inserción de un diagrama. Está compuesto de dos partes.

En una primera sección se abre un modal (líneas 450-452). Para ello se selecciona el elemento con el identificador `"buttonInsert"`. Luego se define una función anónima que se ejecutará cuando se haga clic en el botón. Finalmente se selecciona el elemento con el identificador `"modalInsert"` (un modal de Bootstrap). Se utiliza el método `modal("show")` de jQuery para mostrar el modal.

En la segunda sección se cargan datos en el modal al abrirse. Para ello se selecciona el elemento con el identificador `"modalInsert"` (el mismo del modal). Se utiliza el evento `on('show.bs.modal')` para ejecutar una función anónima justo antes de que el modal se muestre. Se llama a la función externa llamada `diagram_insert`. Esta función se encarga de cargar los datos necesarios para la inserción del diagrama. Los argumentos pasados a la función son:

- `document.getElementById('course').value`: El valor del elemento con identificador `"course"` (el campo de selección de curso).

- `document.getElementById('activity').value`: El valor del elemento con identificador "activity" (el campo de selección de actividad).
- `document.getElementById('modeEdit').value`: El valor del elemento con identificador "modeEdit" (campo oculto que indica si se trata de una edición, que fue establecido en código de la Figura 62).

En resumen, se hace clic en el botón "`butonInsert`". Se muestra el modal "`modalInsert`". Justo antes de mostrarse el modal, se llama a la función `diagram_insert` con los valores de los campos "`course`", "`activity`" y "`modeEdit`" para configurar el modal para la inserción.

*Figura 70 Código del frontend del editor de diagramas. Modal que se muestra cuando se está guardando un diagrama.*

```

400.      <!-- Modal insert-->
401.      <div class="modal fade" id="modalInsert" tabindex="-1" role="dialog"
402.          aria-labelledby="exampleModalLabel" aria-hidden="true">
403.          <div class="modal-dialog" role="document">
404.              <div class="modal-content">
405.                  <div class="modal-header" id="modalInsertHeader">
406.                      <h5 class="modal-title"
407.                          id="modalInsertTitle">Insertando soluci&oacute;n...</h5>
408.                  </div>
409.                  <div class="modal-body" id="modalInsertBody">
410.                      <div id="modalInsertResult" ></div>
411.                  </div>
412.                  <div class="modal-footer" style="padding:3px;">
413.                      id="modalInsertFooter">
414.                          <button type="button" class="btn btn-secondary btn-sm"
415.                              data-dismiss="modal" style="margin:0px;">Close</button>
416.                      </div>
417.                  </div>
418.              </div>
419.          </div>

```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código HTML presentado en la Figura 70 define un modal Bootstrap con los siguientes elementos:

- Un encabezado que muestra un mensaje de "*Insertando solución...*".
- Un cuerpo que contiene un div con el id "`modalInsertResult`" donde se espera mostrar los resultados del proceso de inserción.
- Un pie de página con un botón para cerrar el modal.

Figura 71 Código del backend del editor de diagramas. Guardar Solución: función invocada.

```
978.     function diagram_insert(course_id, nactivity_id, modeEdit) {
979.         saveDiagramProperties(); // do this first, before writing to JSON
980.         let dataToSend = {
981.             course_id: course_id,
982.             nactivity_id: nactivity_id,
983.             edit: modeEdit
984.         }
985.         dataToSend.diagramModel = JSON.parse(myDiagram.model.toJson());
986.         dataToSendAsJson = JSON.stringify(dataToSend);
987.         var home = window.location.hostname;
988.         var port = window.location.port;
989.         $('#modalInsertTitle').text('Insertando Diagrama...');
990.         $('#modalInsertResult').text("");
991.         $('#modalInsertHeader').css("background-color", "white");
992.         $('#modalInsertBody').css("background-color", "white");
993.         $('#modalInsertFooter').css("background-color", "white");
994.         $.ajax({
995.             url: 'handler_diagram_insert.php',
996.             type: "POST",
997.             data: dataToSendAsJson,
998.             contentType: "application/json",
999.             success: function(dataJson){
1000.                 //Convierte el string JSON en un objeto JAVA.
1001.                 data = JSON.parse(dataJson).data;
1002.                 console.log("Se ha insertado el diagrama, OK.");
1003.                 $('#modalInsertTitle').html('<b>Diagrama Agregado</b>');
1004.                 $('#modalInsertResult').text('El Diagrama UML se agregó
1005.                 correctamente a la actividad.');
1006.                 $('#modalInsertHeader').css("background-color",
1007.                 "#98FB98");
1008.                 $('#modalInsertBody').css("background-color", "white");
1009.                 $('#modalInsertFooter').css("background-color",
1010.                 "#98FB98");
1011.             },
1012.             error: function(request,error){
1013.                 const msg = "Error insertado el diagrama.";
1014.                 console.log(msg);
1015.                 $('#modalInsertTitle').html('<b>LO SENTIMOS</b>');
1016.                 $('#modalInsertResult').html('No se pudo agregar su
1017.                 diagrama.<br>Revise la selección de curso y actividad.');
1018.                 $('#modalInsertHeader').css("background-color",
1019.                 "#FFDAB9");
1020.                 $('#modalInsertBody').css("background-color", "white");
1021.                 $('#modalInsertFooter').css("background-color",
1022.                 "#FFDAB9");
1023.             }
1024.         });
1025.     }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Cuando se hace click en el botón Guardar solución, se invoca la función *diagram\_insert*. Esta función, que pertenece al fichero *uml\_diagram.js*, se encarga de enviar los datos de un diagrama al servidor para su inserción. Utiliza AJAX para realizar la comunicación y actualiza la interfaz de usuario basada en los resultados de la operación. Toma tres parámetros: *course\_id*, *nactivity\_id* y *modeEdit*.

Lo primero que se hace es una salva temporal del diagrama que se ha construido llamando a la función `saveDiagramProperties`. Luego se construye una estructura `dataToSend` con los parámetros pasados (identificador del curso, identificador de la actividad y el booleano `modeEdit`), y se le adiciona el modelo del diagrama como un objeto JavaScript, convertido a partir del formato JSON del modelo de diagrama implementado con GoJS. El diagrama implementado con GoJS está representado en la variable `myDiagram`. Luego se convierte el objeto `dataToSend` en una cadena de texto con formato JSON, es decir se serializa la estructura. Se actualiza el contenido del modal para mostrar un mensaje de "Insertando Diagrama...". Se realiza una petición AJAX al archivo `handler_diagram_insert.php` utilizando el método POST. A este archivo se le envían los datos del diagrama serializado en formato JSON. En caso de que sea satisfactoria la ejecución del código del fichero PHP, se cambian los valores del modal para indicar que el diagrama se almacenó correctamente. Se maneja entonces la respuesta en las funciones `success` y `error`. En caso de éxito, se parsea la respuesta JSON, se muestra un mensaje de éxito y se actualiza el estilo del modal. En caso de no sea satisfactoria la ejecución del código PHP entonces se cambian los valores del modal indicando el fallo en el proceso.

Figura 72 Código del backend del editor de diagramas. Consulta para guardar una solución.

```
15. $result = mysqli_query($conexion, "update model_diagram_success set  
16. data='".$diagramModel."' where course_id='".$data->course_id."' and  
17. nactivity_id='".$data->nactivity_id."');
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En el fichero `handler_diagram_insert.php`, se ejecuta la consulta de actualización del diagrama en la base de datos (Figura 72). Recuerde que cuando se creó la actividad, también se creó una entrada en la tabla `model_diagram_success`, donde la solución se estableció vacía. Por otra parte, también debe recordarse que para cada actividad solo existe un diagrama en la tabla `model_diagram_success`. De ahí que si existe una entrada para el diagrama, siempre la operación que se realice sea de actualización del modelo del diagrama.

Una de las partes principales en los diagramas UML son las clases. Para agregar clases al diagrama puede hacerse a través del botón *Agregar clase* o haciendo click derecho opción *Agregar clase*.

Figura 73 Agregar clase



Fuente: (Macas, 2023)

Figura 74 Código del backend del editor de diagramas. Adicionar clase.

```
1083.      /**
1084.      * Agrega una nueva clase UML en el diagrama.
1085.      * @param x - Coordenada X a partir de donde se crea el objeto clase.
1086.      * @param y - Coordenada Y a partir de donde se crea el objeto clase.
1087.      */
1088.     function addNewClass(x, y){
1089.       myDiagram.model.addNodeData({
1090.         name: "Nombre de clase",
1091.         properties: [
1092.           {
1093.             name: "propiedad",
1094.             type: "tipo",
1095.             visibility: "public"
1096.           }
1097.         ],
1098.         methods: [
1099.           {
1100.             name: "método",
1101.             type: "tipo",
1102.             visibility: "public",
1103.             param1Name: "parámetro",
1104.             param1Type: "tipo"
1105.           }
1106.         ],
1107.         loc: x+" "+y
1108.       });
1109.     }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

La función `addNewClass(x, y)` está diseñada para agregar una nueva clase al editor de diagramas UML. Este código forma parte de la personalización de la biblioteca GoJS, con la cual se crean y manipulan diagramas de manera programática.

Primero se añade un nuevo nodo (en este caso, una clase) al modelo `model`, donde se almacenan todas las clases, relaciones y otros elementos, de la referencia al diagrama UML completo representado por `myDiagram`.

Después se establecen las propiedades de la nueva clase representadas por:

- `name`: Establece el nombre de la clase.
- `properties`: Define un array de propiedades de la clase, cada una con un nombre, tipo y visibilidad.
- `methods`: Define un array de métodos de la clase, cada uno con un nombre, tipo, visibilidad y parámetros.
- `loc`: Especifica la ubicación (x, y) de la clase en el diagrama.

El funcionamiento general es el siguiente. Se llama a la función `addNewClass` con las coordenadas `x` e `y` donde se desea ubicar la nueva clase. Se crea un nuevo objeto de clase y se establecen sus propiedades: nombre, atributos, métodos y posición. El nuevo objeto de clase se agrega al modelo del diagrama.

En resumen, esta función proporciona una forma sencilla de agregar nuevas clases a un diagrama UML de manera programática. Al especificar las coordenadas y las propiedades de la clase, se puede construir un diagrama UML de forma dinámica.

Las propiedades de las clases pueden ser modificadas haciendo doble clic sobre los nombre de cada uno de estos elementos o haciendo click derecho sobre el tipo y el alcance o visibilidad. Si desea agregar un nuevo método o propiedad, o eliminar la clase, podrá hacerlo haciendo click derecho sobre el área del nombre de la clase. Para esto se emplean las funciones `addPropertyUML` y `addMethodUML`.

Figura 75 Código del backend del editor de diagrama. Funciones para adicionar una propiedad y un método haciendo uso de GoJS.

```
183.         function addPropertyUML(nodeObject) {
184.             myDiagram.model.commit(m => {
185.                 m.addArrayItem(
186.                     nodeObject.part.data.properties,
187.                     {
188.                         name:"propiedad",
189.                         type:"tipo",
190.                         visibility:"public"
191.                     }
192.                 );
193.             });
194.         }
195.
196.
197.         function addMethodUML(nodeObject) {
198.             myDiagram.model.commit(m => {
199.                 m.addArrayItem(
200.                     nodeObject.part.data.methods,
201.                     {
202.                         name:"método",
203.                         type:"tipo",
204.                         visibility:"public"
205.                     }
206.                 );
207.             });
208.         }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Estas funciones están relacionadas con la manipulación de diagramas. Veamos su funcionamiento.

La función *addPropertyUML* agrega una nueva propiedad a un nodo ya existente en el diagrama y que se recibe como parámetro (*nodeObject*). Se utiliza una función de callback para modificar el modelo del diagrama de manera segura en la que se emplea una referencia mutable al modelo del diagrama. En esta función de callback se accede a un array dentro del nodo que almacena las propiedades existentes, donde se define un nuevo objeto que representa la propiedad a agregar compuesta por:

- **name:** Nombre de la propiedad.
- **type:** Tipo de dato de la propiedad.
- **visibility:** Visibilidad de la propiedad (pública en este caso).

La función *m.addItem* es lo que agrega el nuevo objeto de propiedad al array de propiedades del nodo.

Por su parte la función *addMethodUML* agrega un nuevo método a un nodo ya existente en el diagrama. Su estructura es prácticamente idéntica a *addPropertyUML*. Recibe como parámetro un *nodeObject*, que representa el nodo al que se quiere añadir el método. El resto del código funciona de manera similar a *addPropertyUML*, pero modifica el array de métodos del nodo en lugar del array de propiedades.

En resumen, estas funciones permiten agregar propiedades y métodos a clases existentes en un diagrama UML de forma programática. La mutación del modelo se realiza utilizando un callback para garantizar la consistencia de los datos.

Figura 76 Código del backend del editor de diagramas. Adicionar un enlace entre clases haciendo uso de GoJS.

```
1109.      /**
1110.      * Agrega una nueva Relación en el diagrama, en la posición indicada.
1111.      * @param x - Coordenada X a partir de donde se crea el objeto
1112.      Relación.
1113.      * @param y - Coordenada Y a partir de donde se crea el objeto
1114.      Relación.
1115.      */
1116.      function addNewLink(x, y) {
1117.          myDiagram.model.addLinkData({
1118.              fromPort: "",
1119.              toPort: "",
1120.              "points": [x, y, x+100,y+100],
1121.              relationType: UML_RELATION_TYPES[0]
1122.          });
1123.      }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Otro de los pasos importantes en la elaboración de los diagramas es el establecimiento de las relaciones entre las clases. Esto se lleva a cabo mediante la función *addNewLink*. Esta función, dentro del contexto del editor de diagramas UML, se encarga de agregar una nueva relación entre dos elementos (nodos) del diagrama. La posición inicial de esta relación se define por las coordenadas *x* e *y* proporcionadas.

Primeramente se invoca a la función *addLinkData* del modelo de datos del diagrama, donde se almacenan todas las clases, relaciones y demás elementos. Esta función se utiliza para agregar

un nuevo enlace (relación) al modelo. El objeto que se pasa como argumento define las características de esta relación.

Entre las propiedades de la nueva relación se encuentran *fromPort* y *toPort*. Estos atributos (que están vacíos inicialmente indican los puertos (puntos de conexión) en los nodos de origen y destino de la relación. Otra de las propiedades es *points*, el que define una lista de coordenadas que representan los puntos de control de la línea que conecta los dos nodos. También se tiene la propiedad *relationType*, la que indica el tipo de relación UML que se está creando. En este caso, se toma el primer elemento del array *UML\_RELATION\_TYPES*, el que es un array predefinido con los diferentes tipos de relaciones posibles (generalización, asociación, etc.).

Los Docentes además tiene la posibilidad de visualizar diagramas. Cuando se habla de visualizar los diagramas tanto de estudiantes como de docentes, la sección de botones estará reducida a solo el botón *Salir* y no podrá realizarse ninguna modificación. Los docentes tienen posibilidad también, y con las mismas limitaciones, de visualizar los ejercicios realizados por los estudiantes, sin ninguna posibilidad de modificación o de eliminación. Esto es, opción de **Estudiantes**, apartado *Actividades resueltas*. (Ver Figura 78)

Figura 77 Listado de actividades de cursos con las opciones de ver las actividades resueltas por los estudiantes.

The screenshot shows the XperienceUML application interface. The left sidebar has a dark theme with white icons and text. It includes sections for 'Administrador', 'Gestión de cursos' (Courses), 'Reportes' (Reports), and 'Estudiantes' (Students). Under 'Estudiantes', there are two options: 'Actividades Resueltas' (Solved Activities) which is highlighted in blue, and 'Responder actividades' (Answer activities). The main content area has a light gray background. At the top, it says 'Diagramas Realizados por Estudiantes' (Diagrams made by Students). Below that is a search bar labeled 'Search:'. A table lists seven entries, each with columns for 'Creado por' (Created by), 'Fecha' (Date), 'Curso' (Course), 'Actividad' (Activity), and 'Acciones' (Actions). All entries are created by 'Cristian Estudiante' on June 7, 2024, at various times between 02:54:43 and 03:14:22. The activities listed are 'Introducción a la Programación Orientada a Objetos', 'Clases y Objetos', 'Metodos', 'Vehículos', 'Usuarios y Autenticación', and 'Herencia y Polimorfismo en Programación Orientada a Objetos'. Each row has a small circular icon with a question mark in the 'Acciones' column.

Creado por	Fecha	Curso	Actividad	Acciones
Cristian Estudiante	2024-06-07 02:54:43	Introducción a la Programación Orientada a Objetos	Clases y Objetos	?
Cristian Estudiante	2024-06-07 02:58:59	Introducción a la Programación Orientada a Objetos	Metodos	?
Cristian Estudiante	2024-06-07 02:59:08	Introducción a la Programación Orientada a Objetos	Metodos	?
Cristian Estudiante	2024-06-07 03:06:46	Herencia y Polimorfismo en Programación Orientada a Objetos	Vehículos	?
Cristian Estudiante	2024-06-07 03:13:03	Herencia y Polimorfismo en Programación Orientada a Objetos	Usuarios y Autenticación	?
Cristian Estudiante	2024-06-07 03:13:50	Herencia y Polimorfismo en Programación Orientada a Objetos	Usuarios y Autenticación	?
Cristian Estudiante	2024-06-07 03:14:22	Herencia y Polimorfismo en Programación Orientada a Objetos	Usuarios y Autenticación	?

Fuente: (Macas, 2023)

Para visualizar los diagramas de las actividades creadas se listan en la opción de **Gestión de curso**, apartado *Soluciones de actividades*. En este listado los datos que se muestran son el

Curso, la Actividad, el Enunciado, la Descripción técnica. Además se pueden tomar acciones de Ver, Editar y Eliminar la solución. (Ver Figura 78)

Figura 78 Listado de las soluciones de las actividades.

Curso	Actividad	Enunciado	Solución Técnica	Acciones
<i>El mundo de la programación orientada a objetos</i>	<i>Sistema de Notificaciones</i>	<i>En una aplicación de mensajería, el sistema de notificaciones necesita enviar notificaciones a los usuarios a través de un servicio de mensajería. Este servicio puede enviar mensajes por correo electrónico o SMS. Debes modelar esta situación creando una estructura adecuada para representar el sistema de notificaciones y el servicio de mensajería.</i>	<i>1. Crea la clase SistemaNotificaciones con un método enviarNotificación de tipo Void. 2. Crea la clase ServicioMensajería con métodos enviarCorreo (Void) y enviarSMS (Void). 3. Establece que SistemaNotificaciones depende de ServicioMensajería. ---Relación: SistemaNotificaciones --&gt; ServicioMensajería (Dependencia).</i>	
<i>El mundo de la programación orientada a objetos</i>	<i>Departamento y Empleados</i>	<i>En una empresa, un departamento está compuesto por varios empleados. Cada empleado tiene un nombre y un puesto específico, y pertenece a un único departamento. Sin embargo, un departamento puede existir independientemente de los empleados. Debes modelar esta situación creando una estructura adecuada para representar los departamentos y los empleados. Incluye métodos para agregar y eliminar empleados de un departamento.</i>	<i>1. Crea la clase Departamento con los atributos nombre (Cadena) y empleados (Lista de Cadenas). 2. En la clase Departamento, añade los métodos agregarEmpleado (Void) y eliminarEmpleado (Void). 3. Crea la clase Empleado con los atributos nombre (Cadena) y puesto (Cadena). 4. Establece que Departamento agrega a Empleado. ---Relación: Departamento &lt;-&gt; Empleado (Agregación).</i>	

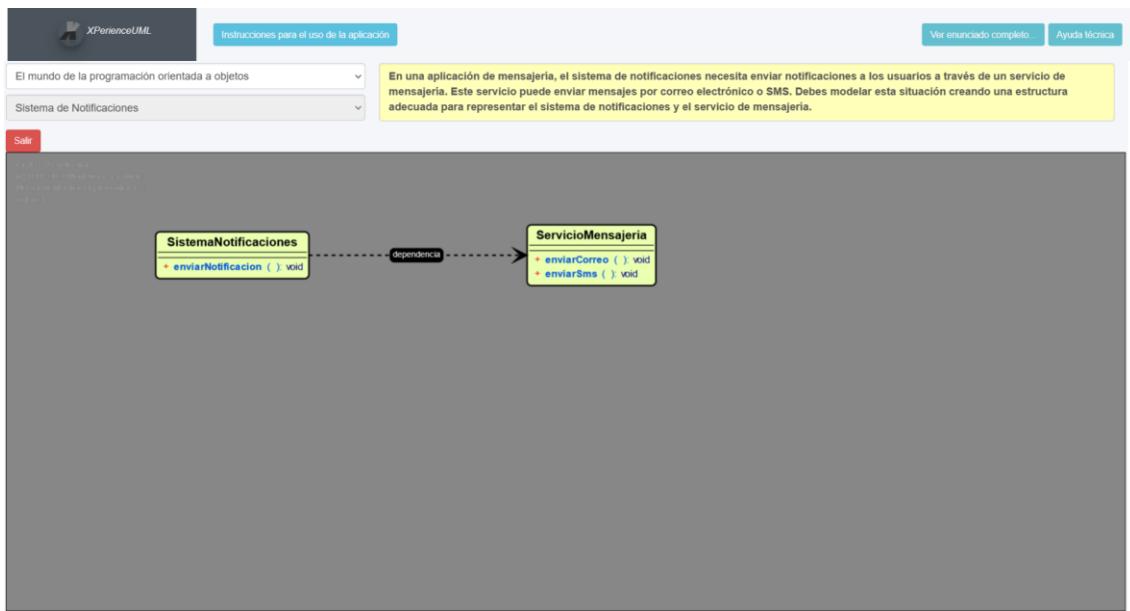
Fuente: (Macas, 2023)

Cuando se toman alguna de las acciones de *Ver* o *Editar* solución, el editor de diagrama se muestra, cargando la solución almacenada en la base de datos.

La opción de *Editar* ya fue analizada anteriormente.

Cuando se elige *Ver solución*, se visualiza el curso, la actividad, la descripción de la actividad, y la solución propuesta por el Docente como puede verse en la siguiente figura. Aquí no se podrá modificar el diagrama, solo se visualiza. (Ver Figura 79)

Figura 79 Gestión de Diagramas. Ver Diagrama



Fuente: (Macas, 2023)

En la visualización de un diagrama se emplea el fichero *diagrama.php*. Cuando se llama al editor para visualizar entre los parámetros se le pasa en modo, la tabla que será empleada y el identificador de la actividad. Tanto para docentes como para estudiantes esta visualización será exactamente igual, con la salvedad de que las soluciones para los docentes se encuentran en la tabla *model\_diagram\_success*, mientras que para los estudiantes estas soluciones se encuentran en la tabla *model\_diagram\_test*. De ahí que en código de visualización se pregunte por el nombre de la tabla para hacer la consulta. Una vez realizada la consulta los datos se pasan a un elemento oculto que luego el editor tomará su valor y lo visualizará correctamente. Los diagramas almacenados en la base de datos están en formato JSON, el cual es el mismo formato con el que trabaja GoJS para la gestión de diagramas.

Figura 80 Código del backend. Cargar diagrama UML desde la base de datos.

```
49. if (isset($_GET["table"]) && ($_GET["table"] != 'undefined'))
50. {
51.     $table = $_GET["table"];
52.     if (isset($_GET["id"]) && ($_GET["id"] != 'undefined'))
53.     {
54.         $id = $_GET["id"];
55.         $conexion = mysqli_connect('localhost', 'xperienc', 'Zb(nY;w1A62r1R',
56. 'xperienc_uml');
57.         $result = mysqli_query($conexion, "select data from ".$table." where
58. id='".$id."'");
59.         $diagram = mysqli_fetch_array($result);
60.         if (isset($diagram))
61.         {
62.             if (sizeof($diagram) > 0){
63.                 $data = (string)$diagram[0];
64.                 $conexion->close();
65.                 echo '<pre hidden="true"
66. id="dataJsonDiagram">' . $data . '</pre>';
67.             }
68.         }
69.     }
70. }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código proporcionado en la Figura 80 es un fragmento PHP que es parte de un script que recupera datos de la base de datos y los muestra en la página web. Pertenece al fichero *diagrama.php*.

Inicialmente en este fragmento se verifica si existe un parámetro llamado "*table*" en la URL de la página y si el valor dicho parámetro es diferente a "*undefined*". Si ambas condiciones son verdaderas, significa que se ha proporcionado un valor válido para el parámetro "*table*" en la URL. Luego se verifica la existencia del parámetro "*id*": Similar a la verificación anterior, pero para el parámetro "*id*". Si el parámetro "*id*" existe, se asigna su valor a la variable *\$id*. Entonces se establece una conexión a la base de datos utilizando los parámetros proporcionados (localhost, usuario, contraseña y base de datos).

A continuación se ejecuta la consulta SQL. Es decir se realiza una consulta a la base de datos para obtener el campo "*data*" de la tabla especificada en el parámetro "*table*" donde el campo "*id*" coincida con el valor del parámetro "*id*". Se obtiene el primer registro del resultado de la consulta y se le asigna a la variable *\$diagram*. Si el valor de esta variable no es nulo, significa que se encontró un registro. Se verifica también la cantidad de datos obtenidos en la consulta: Si el tamaño del array *\$diagram* es mayor que cero, significa que hay datos. Entonces se

convierte el primer elemento del *array* `$diagram` a una cadena y lo asigna a la variable `$data`. Cierra la conexión a la base de datos y establece el dato en un elemento `<pre>` oculto con el id `"dataJsonDiagram"`.

En resumen, el código verifica la presencia de los parámetros `"table"` e `"id"` en la URL. Si ambos están presentes, establece una conexión a la base de datos, realiza una consulta para obtener datos específicos y establece el resultado en un elemento oculto en la página.

### Gestión de diagramas para Estudiantes

Los usuarios con rol estudiantes tendrán sus limitaciones y solo podrán crear actividades del curso en el que han sido matriculados, mientras que visualizaran solo los diagramas que han creado, sin que puedan modificarlos. Para los usuarios con rol de Estudiantes, la gestión de diagramas hace referencia las soluciones de las actividades realizadas o por realizar.

Cuando un usuario Estudiante se registra lo único que tiene permitido es ver su perfil, sin acceso a modificarlo y gestionar diagramas UML. Esta gestión viene limitada a la visualización y creación de diagramas, sin la posibilidad de modificaciones. Estas acciones están dadas por las opciones *Mis actividades* y *Responder actividad*. (Ver Figura 81)

Si selecciona *Mis actividades*, se listarán todas las actividades realizada por el estudiante registrado, sin posibilidad de ver los diagramas de otros estudiantes, o la solución brindada por el Docente. De esos diagramas se mostrarán el autor o creador del mismo, que coincide con el usuario registrado, el nombre del curso y el nombre de la actividad a la que pertenece dicho diagrama. También se brinda la posibilidad de ver el diagrama como tal, mediante el ícono *Ver diagrama*. La vista de esos diagramas es exactamente igual a la vista de un diagrama creado por un Docente, con las mismas limitaciones.

Figura 81 Gestión de Diagramas. Diagramas Realizados por Estudiantes

The screenshot shows a web-based application interface for managing UML diagrams. At the top, there's a header with the logo 'XPerienceUML', a user icon for 'Estudiante 10', and navigation links for 'Inicio' and 'Ayuda'. On the right, it says 'Estudiante 10'. Below the header is a sidebar with a dropdown menu set to 'Estudiantes' and two options: 'Mis actividades resueltas' (selected) and 'Responder actividades'. The main content area is titled 'Mis actividades resueltas' and displays a table of solved activities. The table has columns: 'Creado por', 'Fecha', 'Curso', 'Actividad', and 'Acciones'. The data shows multiple entries for 'Estudiante 10' from June 12, 2024, at various times, with courses like 'Introducción a la Programación Orientada a Objetos' and 'Herencia y Polimorfismo en Programación Orientada a Objetos', and activities like 'Clases y Objetos' and 'Metodos'. At the bottom of the table, it says 'Mostrando página 1 de 4' and has a page navigation bar with buttons for 'Anterior', page numbers 1, 2, 3, 4, and 'Próximo'. At the very bottom of the page, there's a copyright notice 'Copyright © 2024 XPerienceUML. Todos los derechos reservados.' and a version number 'Version 1.0'.

Fuente: (Macas, 2023)

Figura 82 Código del frontend. Listado de actividades resueltas por un estudiante.

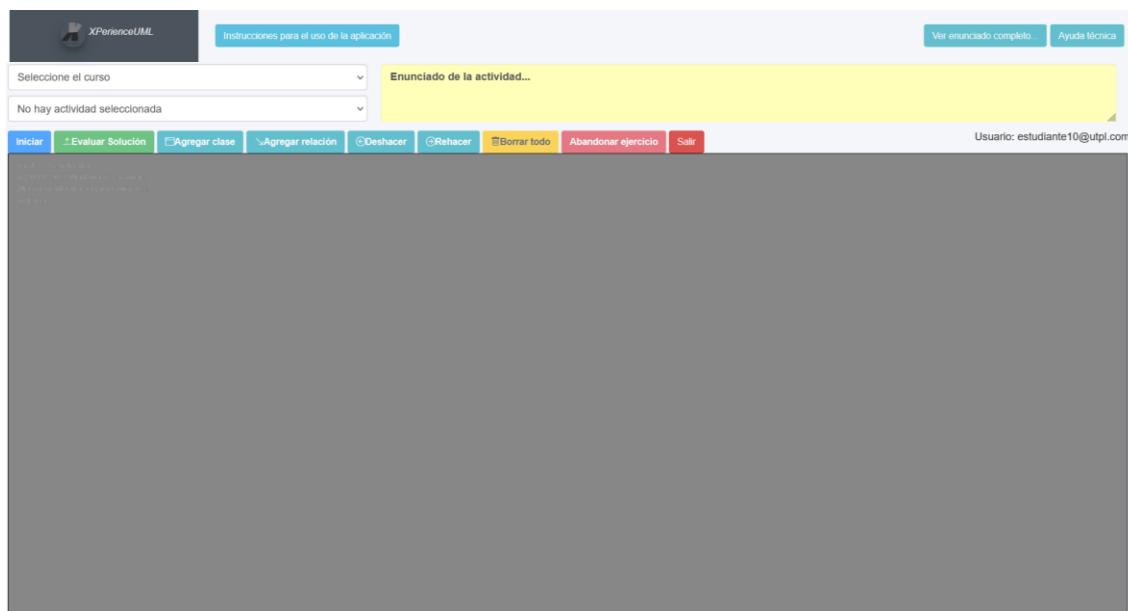
```

23. <table id="example1" class="table table-bordered table-striped dataTable"
24. role="grid" aria-describedby="example1_info">
25. <thead>
26.   <tr>
27.     <th>Creado por</th>
28.     <th>Fecha</th>
29.     <th>Curso</th>
30.     <th>Actividad</th>
31.     <th>Acciones</th>
32.   </tr>
33. </thead>
34. <tbody>
35.   {%
36.     for model_diagram_test in model_diagram_tests %}
37.   <tr>
38.     <td><strong><i>{{ model_diagram_test.action }}</i></strong></td>
39.     <td><strong><i>{{ model_diagram_test.Fecha }}</i></strong></td>
40.     <td><strong><i>{{ model_diagram_test.course }}</i></strong></td>
41.     <td><strong><i>{{ model_diagram_test.nactivity }}</i></strong></td>
42.       <a href="/public/GoJS-
43. master/projects/pdf/diagrama.php?mode=view&table=model_diagram_test&id={{ 
44. model_diagram_test.id }}" title="Ver">
45.         <i class="fa fa-eye"></i>
46.       </a>
47.     </td>
48.   </tr>
49.   {%
50.     else %}
51.   <tr>
52.     <td colspan="6">no hay datos</td>
53.   </tr>
54.   {%
55.     endfor %}
56. </tbody>
57. </table>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Si selecciona **Responder actividad**, se carga el editor de diagramas UML, tal y como se ve en la figura siguiente. (Ver Figura 83)

Figura 83 Editor de Diagramas UML



Fuente: (Macas, 2023)

La composición de editor será prácticamente igual al editor para los usuarios con rol Docente, con la salvedad de que:

- Los usuarios con rol Estudiantes deben escoger el curso y la actividad que van a responder. La actividad estará deshabilitada hasta que no se escoja el curso. Segundo la actividad se carga entonces el enunciado de dicha actividad.
- Los botones que se muestran son *Iniciar*, *Evaluar Solución*, *Agregar clase*, *Agregar solución*, *Deshacer*, *Rehacer*, *Borrar todo*, *Abandonar ejercicio* y *Salir*. Todos estarán inicialmente deshabilitados.

Los botones *Ayuda técnica* y *Ver enunciado completo*, inicialmente están inhabilitados, hasta tanto no se seleccione un curso y una actividad a resolver.

Para la gestión de los cursos y las actividades a la hora de darle solución a una actividad se emplean los ficheros *diagrama.php* y *actividad.php*, entre otros. Cuando se abre el editor se le pasa como parámetro en la URL el identificador del usuario, con lo que primero se buscan los cursos en los que fue matriculado. Una vez realizada la consulta se van llenando los campos del selector con el listado de cursos devueltos.

Los cursos que se muestran serán cursos donde el estudiante ha sido matriculado, es decir, es miembro.

Figura 84 Código del frontend para seleccionar los cursos donde el estudiante ha sido matriculado.

```
126. <select class="form-control" name="course" id="course" required>
127.   <?php
128.     echo '<option disabled selected> Seleccione el curso </option>';
129.     if (!isset($_GET["id"]))
130.     {
131.       $sql = 'SELECT course.* FROM course LEFT JOIN course_user ON
132. course_user.course_id = course.id
133.           WHERE course.id = course_user.course_id AND
134. course_user.user_id = '.$idUser;
135.       $result = mysqli_query($conn,$sql);
136.       while ($mostrar=mysqli_fetch_array($result)){
137.         echo '<option
138. value="'.$mostrar['id'].'">'.$mostrar['name'].'</option>';
139.       }
140.     }
141.   else
142.   {
143.     $id = $_GET["id"];
144.     $tabla = $_GET["table"];
145.     $sql = "SELECT course.id, course.name, $tabla.data, nactivity.title AS
146. title, nactivity.id AS idactivity, nactivity.description AS description,
147. nactivity.tecsol AS tecsol
148.             FROM course
149.                 LEFT JOIN $tabla ON $tabla.course_id = course.id
150.                 LEFT JOIN nactivity ON $tabla.nactivity_id =
151. nactivity.id
152.                     WHERE $tabla.course_id = course.id AND nactivity.id =
153. $tabla.nactivity_id AND $tabla.id = ".$id;
154.         $result = mysqli_query($conn,$sql);
155.         $mostrar=mysqli_fetch_array($result);
156.         echo '<option value="'.$mostrar['id'].'"'
157. selected'>'.$mostrar['name'].'</option>';
158.         $description = $mostrar['description'];
159.         $tecsol = $mostrar['tecsol'];
160.       }
161.     ?>
162.   </select>
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

El código de la Figura 84 genera un elemento *<select>* HTML para seleccionar un curso.

Veamos su funcionamiento paso a paso.

Inicialmente se crea una primera opción con el texto "*Seleccione el curso*". Se establecen los atributos *disabled* para deshabilitar la opción y *selected* para seleccionarla por defecto. Luego se verifica si existe un parámetro llamado "*id*" en la URL de la página utilizando *isset*. El valor del parámetro *id* en la URL hace referencia al identificador de una actividad.

Si no hay parámetro "*id*", se ejecuta una consulta SQL para obtener todos los cursos de la tabla *course*. La consulta utiliza un LEFT JOIN con la tabla *course\_user* (para filtrar cursos relacionados con el usuario). El resultado se recorre con un *while* y se crea una opción para cada curso con el valor (*value*) igual al id del curso y el texto (*name*) igual al nombre del curso.

Si existe el parámetro "*id*", se extrae su valor y se almacena en la variable *\$id*. Se lee también el valor del parámetro "*table*". Se construye una consulta SQL más compleja para obtener información detallada de un curso específico basado en el parámetro "*id*". La consulta utiliza LEFT JOIN para relacionar la tabla *course* con otra tabla (*\$tabla*) y con la tabla *nactivity* (para obtener información adicional de las actividades). Se ejecuta la consulta, se obtiene el primer resultado (*mysqli\_fetch\_array*) y se crea una opción para mostrar el curso seleccionado. Se extraen valores adicionales como *description* y *tecsol* del registro obtenido (para utilizarlos en otra parte del código).

En resumen, este código genera un elemento *<select>* que permite seleccionar un curso. Se muestra una opción por defecto para elegir un curso y luego, dependiendo del parámetro "*id*" en la URL, se muestran todos los cursos o se selecciona un curso específico con información adicional.

Una vez seleccionado el curso, se habilitan las actividades. Las actividades que se listarán están relacionadas con el curso que se escogió.

Figura 85 Código del frontend para seleccionar las actividades de un curso donde el estudiante ha sido matriculado.

```

126. <div class="row" style="margin-bottom:7px;margin-right:3px">
127.   <?php
128.     if (isset($_GET["id"]))
129.     {
130.       echo '<select class="form-control" name="activity" id="activity" disabled>';
131.       echo '<option value="'. $mostrar['idactivity'] .'" selected>' . $mostrar['title'] . '</option>';
132.       echo '</select>';
133.     }
134.   else {
135.     echo '<select class="form-control" name="activity" id="activity" onchange="selectDescription()" onclick="getCourse()" >';
136.     echo '<option disabled selected> No hay actividad seleccionada</option>';
137.     echo '</select>';
138.   }
139. ?>
140. </div>

```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En el código PHP de la Figura 85 se genera un elemento `<select>` HTML para seleccionar una actividad. El comportamiento del elemento depende del valor del parámetro `id` en la URL.

Si existe el parámetro `id` en la URL, se crea un elemento `<select>` con la clase `form-control`, nombre `activity` e `idactivity`. El elemento `<select>` tiene una única opción, cuyo valor y texto se toman de las variables `$mostrar['idactivity']` y `$mostrar['title']` respectivamente. El atributo `disabled` se establece para el elemento `<select>`, lo que impide al usuario seleccionar una opción diferente.

Si no existe el parámetro `id` en la URL, se crea un elemento `<select>` con la clase `form-control`, nombre `activity` e `idactivity`. Se agrega una opción por defecto con el texto "*No hay actividad seleccionada*". Se añaden los atributos `onchange="selectDescription()"` y `onclick="getCourse()"` al elemento `<select>`, lo que implica que al cambiar la selección o hacer clic en el elemento se ejecutarán las funciones `selectDescription()` y `getCourse()`.

En resumen, el código genera dinámicamente un elemento `<select>` para seleccionar una actividad. Si se proporciona un `id` en la URL, se muestra una única opción no editable. De lo contrario, se muestra un selector vacío con opciones para ser seleccionadas y se asocian funciones a los eventos `onchange` y `onclick`.

Figura 86 Código del backend para obtener los datos relacionados con una actividades.

```
145.     function selectDescription() {
146.         var descrip = document.getElementById('activity').value;
147.         $.ajax({
148.             type:'POST',
149.             url:"actividad.php",
150.             data:'idActividad='+descrip,
151.             success:function(data){
152.                 var arr = data.split("|");
153.                 document.getElementById('description').value=arr[0];
154.                 $('#modalInitResult').text(arr[0]);
155.                 $('#modalAyudaResult').text(arr[1]);
156.             }
157.         });
158.     }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

La función `selectDescription()` es una función JavaScript que se encuentra en el fichero `diagrama.php` y se ejecuta cuando cambia el valor del elemento `<select>` con el id "`activity`". Su objetivo principal es obtener información adicional sobre la actividad seleccionada y actualizar elementos en la página. Para ello, primero se obtiene el valor seleccionado del elemento `<select>` con el id "`activity`" y lo almacena en la variable `descrip`. Seguido realiza una petición AJAX. Para ello utiliza la librería jQuery para enviar una petición AJAX al servidor. En esta petición se tiene en cuenta los siguientes parámetros:

- `type: 'POST'`. Indica que la petición es de tipo POST.
- `url: "actividad.php"`. Especifica la URL del script PHP que procesará la solicitud.
- `data: 'idActividad='+descrip`. Envía el valor del elemento seleccionado (almacenado en `descrip`) como parámetro llamado `idActividad` al script PHP.

Luego se procesa la respuesta en caso que se tenga éxito. Se define una función de callback que se ejecutará cuando la petición AJAX tenga éxito. Entonces se divide la respuesta del servidor en un array utilizando el carácter "|" como separador. Se asume que la respuesta del servidor contiene dos valores separados por "|". Se asigna el primer elemento del array (`arr[0]`) al valor del elemento con el id "`description`". Por otra parte se establece el texto del elemento con el id "`modalInitResult`" al primer elemento del array. Mientras que se establece el texto del elemento con el id "`modalAyudaResult`" al segundo elemento del array.

En resumen, la función *selectDescription()* envía el valor de la actividad seleccionada a un script PHP mediante una petición AJAX. Luego, procesa la respuesta del servidor, extrae los datos necesarios y actualiza los elementos correspondientes en la página.

Figura 87 Código del backend. Consulta que devuelve el enunciado y la solución técnica de una actividad.

```
1. <?php
2.     require "conexion.php";
3.     if (isset($_POST['idActividad']))
4.     {
5.         $idActividad = $_POST['idActividad'];
6.         $sql= "SELECT description, tecsol
7.             FROM nactivity
8.             WHERE id = '$idActividad'";
9.         $result=mysqli_query($conn,$sql);
10.        $count=mysqli_num_rows($result);
11.        if (mysqli_num_rows($result)>0)
12.        {
13.            $mostrar=mysqli_fetch_array($result);
14.            echo $mostrar['description'].'|'.$mostrar['tecsol'];
15.        }
16.    }
17.
18. ?>}
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Una vez seleccionada una actividad se carga el enunciado y la solución técnica de la actividad seleccionada. Esto se realiza a través de la invocación y ejecución del fichero *actividad.php*. El código PHP proporcionado por este fichero recupera la descripción y el tecsol de una actividad a partir de su ID, enviado mediante una petición POST. Primero se incluye el archivo *conexion.php* que contiene la conexión a la base de datos. Seguido se comprueba si se ha enviado un valor para el parámetro *idActividad* mediante el método POST. Se construye una consulta SQL para obtener los campos *description* y *tecsol* de la tabla *nactivity* donde el id coincida con el valor recibido. A continuación se ejecuta la consulta SQL utilizando la conexión a la base de datos almacenada en *\$conn*. Se procede entonces a la verificación de resultados. Se cuenta el número de filas devueltas por la consulta y luego se comprueba que se hayan obtenidos registros. Se obtiene finalmente el primer registro del resultado de la consulta como un array asociativo y se imprime la *descripción* y el *tecsol* separados por un carácter "|".

Luego se habilitarán los botones *Ayuda técnica*, *Ver enunciado completo* e *Iniciar*, mediante la invocación de la función *getCourse*.

Figura 88 Código del backend para deshabilitar los botones Iniciar, Ayuda técnica y Descripción.

```
159.     function getCourse()
160.     {
161.         document.getElementById('botonIniciar').disabled=false;
162.         document.getElementById('Ayuda').disabled=false;
163.         document.getElementById('verDescription').disabled=false;
164.     }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Si hace clic en el botón **Iniciar**, se activarán todos los demás botones y el botón **Iniciar** nuevamente se deshabilitará, permitiéndose en este punto la creación de los diagramas.

Figura 89 Código del frontend del botón Iniciar.

```
203. echo '<button name="botonIniciar" id="botonIniciar" class="btn btn-primary
204. btn-sm" disabled
205. onclick="lanzar(this.name,\'AregarRel,AregarClases,Deshacer,Rehacer,Borrar,A
206. bandonar,verDescription,\'.\$modalType.\',Ayuda\')">';
207. echo '<strong>Iniciar</strong>';
208. echo '</button>';
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 90 Código del backend del script que se ejecuta la pulsar el botón Iniciar.

```
592.     function lanzar(name,nombreBotones){
593.         activarBotones(name,nombreBotones);
594.         var combo = document.getElementById("activity");
595.         var selected = combo.options[combo.selectedIndex].text;
596.         var txt = "Usted va a responder la actividad: "+selected;
597.         alert(txt);
598.         var c = document.getElementById('course').value;
599.         var a = document.getElementById('activity').value;
600.         var u = document.getElementById('user').value;
601.         report_verb(
602.             c,
603.             a,
604.             u,
605.             'initialized'
606.         );
607.         report_verb(
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

La función *lanzar*, que se encuentra en el fichero *diagrama.php*, recibe como parámetro el nombre del botón que esta desencadenado el evento, y una cadena con los nombres de los botones que van a activarse separados por coma (‘,’). Lo primero que se hace en la función es mandar a activar los botones. Aquí el nombre del botón que desencadena la acción pasa

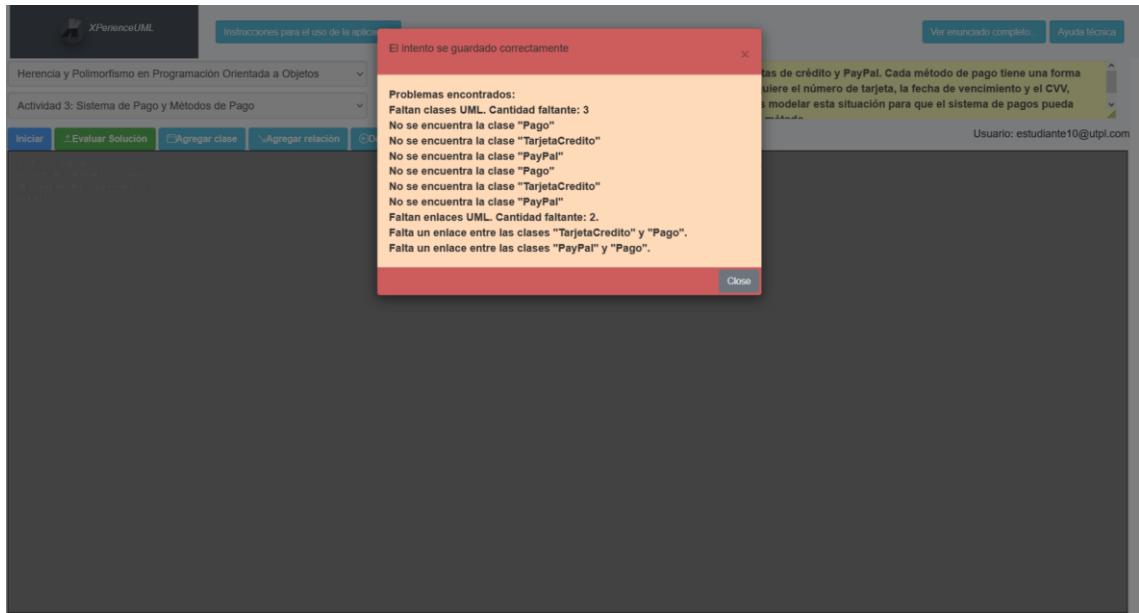
separado para diferenciarlo del resto y que pueda ser desactivado. Luego se toma el nombre del curso y la actividad que fue seleccionada para especificarle al estudiante cual solución va a responder. Por último, se toman los valores de identificación del curso, la actividad y el usuario, para reportar eventos lanzando los verbos Cmi5 “*initialized*” y “*launched*”. Antes de finalizar la función además, se deshabilitan el curso y la actividad para que no puedan ser modificados durante la realización del ejercicio.

Para la creación de una solución a la actividad escogida, se siguen exactamente los mismo pasos que describieron en la creación de una solución por parte de los docentes. Es decir, uno puede crear clases, editar los nombres de las clases, de los atributos y los métodos de las clases, así como establecer como privados, públicos o protegido los métodos y atributos; y darle un tipo específico a esos métodos y atributos. También se pueden establecer relaciones entre las clases.

La creación de una solución por parte de los estudiantes cuenta con los botones ***Evaluar Solución, Agregar clase, Agregar relación, Deshacer, Rehacer, Borrar todo, Abandonar ejercicio y Salir.***

Si se selecciona ***Evaluar***, la aplicación realizará una comparación entre la solución provista por el usuario en ese momento y la provista por el Docente. Se mostrará un modal con los errores detectados. (Ver Figura 91.)

Figura 91 Gestión De Diagrama. Evaluación con Error.



Fuente: (Macas, 2023)

Se puede ir construyendo el diagrama y evaluándose para corregir la solución.

Figura 92 Código del backend para comparar la solución del estudiante con la solución del docente.

```
227.     function check_uml_diagram($template, $test) {
228.         // $template = json_decode($template);
229.         // $test = json_decode($test);
230.         $result = [];
231.         try{
232.             if ((!isset($template)) || (!isset($test))){
233.                 $result[] = "Uno de los diagramas a comparar no ha sido
234. establecido.";
235.                 return($result);
236.             }elseif (!is_object($template)) || (!is_object($test))){
237.                 $result[] = "Uno de los diagramas a comparar no es un
238. objeto.";
239.                 return($result);
240.             }elseif (!property_exists($template, "nodedataArray")) ||
241. (!property_exists($test, "nodedataArray")){
242.                 $result[] = "El diagrama no tiene clases UML";
243.                 return($result);
244.             }elseif (!property_exists($template, "linkdataArray")) ||
245. (!property_exists($test, "linkdataArray")){
246.                 $result[] = "El diagrama no tiene enlaces UML";
247.                 return($result);
248.             }
249.             $result = check_uml_classes($result, $template->nodedataArray,
250. $test->nodedataArray);
251.             // $aux = getNamesArray($template->nodedataArray);
252.             $result = check_uml_links($result, $template->linkdataArray,
253. $test->linkdataArray, $template->nodedataArray);
254.         }
255.         catch(Exception $e){
256.             $result[] = "Ocurrió algún error de llave no existente o
257. desconocido.";
258.         }
259.         return $result;
260.     }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la comparación de los diagramas se emplea el fichero *uml\_json\_comparar.php*. Este fichero consta de varias funciones que van a ir permitiendo la comparación de las soluciones por parte. La función *check\_uml\_diagram* de este fichero debe recibir los dos modelos a comparar. Esta función es parte de un sistema que compara dos diagramas UML representados como objetos JSON. Su objetivo principal es verificar la estructura básica de los diagramas, asegurando que contengan los elementos necesarios (clases y enlaces). Se inicializa un array vacío para almacenar los resultados de la comparación.

A continuación se realiza una serie de verificaciones. Se verifica que ambos parámetros *\$template* y *\$test* estén definidos y que ambos parámetros sean objetos. También se verifica que ambos objetos tengan la propiedad *nodedataArray* (que contiene información sobre las clases) y *linkdataArray* (que contiene información sobre los enlaces). La función asume que los diagramas están representados como objetos JSON con propiedades *nodedataArray* y

*linkdataArray*. En caso de fallo en alguna de las verificaciones, se agrega un mensaje de error al array *\$result* y la función retorna.

Para la comparación de clases y enlaces se llama primero a la función *check.uml.classes* para comparar las clases de ambos diagramas y agrega los resultados al array *\$result*. Luego se llama a la función *check.uml.links* para comparar los enlaces de ambos diagramas y agrega los resultados al array *\$result*.

En este código también se hace manejo de excepciones. Esto se lleva a través de un bloque try-catch que captura posibles excepciones y agrega un mensaje de error genérico al array *\$result*.

Finalmente se retorna el array *\$result* que contiene los resultados de la comparación.

En caso de la comparación de las clases se verifica que la cantidad de ambos modelos coincidan, que los nombres coincidan independientemente del uso de mayúsculas o minúsculas, y que los métodos y propiedades también coincidan independientemente del uso de mayúsculas o minúsculas, señalándose también las clases que tienen métodos o propiedades repetidas en el modelo del estudiante.

En el caso de la comparación de los enlaces, además de la cantidad de enlaces y los tipos de enlaces, se verifica que los orígenes y destino de los mismos sean correctamente establecidos.

Si elige *Abandonar ejercicio* se vuelve a la vista principal de los estudiantes.

Figura 93 Código del backend del script que se ejecuta la pulsar el botón Abandonar.

```
618.     function abandonar()
619.     {
620.         report_verb(
621.             document.getElementById('course').value,
622.             document.getElementById('activity').value,
623.             document.getElementById('user').value,
624.             'abandoned'
625.         );
626.         report_verb(
627.             document.getElementById('course').value,
628.             document.getElementById('activity').value,
629.             document.getElementById('user').value,
630.             'waived'
631.         );
632.         var home = window.location.hostname;
633.         var port = window.location.port;
634.         window.location.href = 'http://'+home+ '/public/index.php';
635.     }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

En la acción de abandonar un ejercicio se emplea el fichero *diagrama.php*. Aquí inicialmente se lanzan los verbos *abandoned* y *waived*, antes de volver a la página inicial de la aplicación.

### Gestión de declaraciones.

Una de los pilares de la aplicación es la gestión de declaraciones CMI5.

Las reglas a seguir para el lanzamiento de verbos son:

- Con la ayuda del botón Iniciar en el editor de diagramas UML podemos lanzar los verbos de “Launched” y “Initialized”.
- Cuando evalué una solución y sea errónea debe lanzarse el verbo “Failed”.
- Cuando evalué una solución satisfactoriamente se lanzan los verbos “Passed” y “Completed”.
- Con el botón Abandonar ejercicio se lanzan los verbos “Waived” y “Abandoned”

Las declaraciones y los reportes son almacenados en la tabla **statement** de la base de datos.

Para la gestión de las declaraciones se emplean los ficheros *diagrama.php* y *uml\_diagram.js* desde donde se lanzan la mayoría de los verbos. También se emplea el fichero *handler\_cmi5\_as\_doctrine\_format.php*, que es donde se implementan los objetos como si fuesen realizados por Doctrine.

Figura 94 Código del backend del editor de diagramas.

```
1140.
1141.     function report_verb(course_id, nactivity_id, user_email, verb) {
1142.         saveDiagramProperties(); // do this first, before writing to JSON
1143.         let dataToSend = {
1144.             course_id: course_id,
1145.             nactivity_id: nactivity_id,
1146.             user_email: user_email,
1147.             verb: verb
1148.         }
1149.         dataToSend.diagramModel = JSON.parse(myDiagram.model.toJson());
1150.         dataToSendAsJson = JSON.stringify(dataToSend);
1151.         $.ajax({
1152.             url: "handle_cmi5_report_verb.php",
1153.             type: "POST",
1154.             data: dataToSendAsJson,
1155.             contentType: "application/json",
1156.             success: function(dataJson){
1157.                 console.log('Verbo cmi5: ', verb);
1158.             },
1159.             error: function(request,error){
1160.                 console.log('Error reportando verbo cmi5: ', verb);
1161.             }
1162.         });
1163.
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Como puede verse en la Figura 93 el lanzamiento de un verbo se realiza mediante la función *report\_verb*. Esta función, que se encuentra en el fichero *uml\_diagram.js*, es parte de una aplicación que interactúa con un sistema de aprendizaje electrónico basado en el estándar CMI5. Su objetivo principal es enviar información sobre la actividad del usuario al servidor para ser procesada. Recibe como parámetros, el identificador del curso, de la actividad, del usuario y el verbo.

Lo primero que se hace en la implementación de esta función es una salva temporal del diagrama que se ha construido, invocando la función *saveDiagramProperties*, la que llama a una función externa para guardar las propiedades del diagrama antes de serializar el modelo. Esto es necesario para preservar información adicional sobre el diagrama.

Luego se crea un objeto *dataToSend* con las siguientes propiedades:

- *course\_id*: Identificador del curso.
- *nactivity\_id*: Identificador de la actividad.
- *user\_email*: Correo electrónico del usuario.
- *verb*: Acción realizada por el usuario (verbo CMI5).

Se serializa entonces el modelo del diagrama usando `JSON.parse(myDiagram.model.toJson())` y lo agrega al objeto `dataToSend` con la clave `diagramModel`. Seguido se convierte el objeto `dataToSend` a una cadena JSON utilizando `JSON.stringify`. Se procede a enviar algunos datos al servidor, utilizando jQuery para realizar una petición AJAX al script `handle_cmi5_report_verb.php`. Se envían los datos serializados en el cuerpo de la petición con el tipo de contenido `application/json`. Finalmente se imprime un mensaje de éxito o error en la consola dependiendo de la respuesta del servidor.

En este código la variable global `myDiagram` representa el diagrama implementado con GoJS.

A continuación mostramos la implementación de los verbos definidos para esta aplicación.

Figura 95 Código del backend de la implementación de verbos.

```
19. function createVerb($verbType) {
20.     $verb = null;
21.     switch ($verbType) {
22.         case 'launched':
23.             $verb = array(
24.                 'id' => 'http://adlnet.gov/expapi/verbs/lanzado',
25.                 'display' => array('en-US' => 'launched', 'es' => 'lanzado')
26.             );
27.             break;
28.         case 'initialized':
29.             $verb = array(
30.                 'id' => 'http://adlnet.gov/expapi/verbs/inicializado',
31.                 'display' => array('en-US' => 'initialized', 'es' =>
32. 'inicializado')
33.             );
34.             break;
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para la gestión de las declaraciones se emplea del fichero `handler_cmi5_asDoctrineFormat.php`, la función `createVerb`. Esta función ayuda a crear objetos verbales basados en el tipo de verbo proporcionado siguiendo la especificación xAPI. Los objetos verbales contienen información sobre el identificador del verbo y muestran texto en diferentes idiomas. La forma en que se crean estos objetos sigue el fin de serializar los datos y que sean compatibles con el modo de almacenamiento de Doctrine de Sympony en PHP. Los datos de esta forma se corresponden al estándar Cmi5.

En la implementación inicialmente se declara una variable `$verb` e inicializa a null. Luego se hace la selección del verbo. Para esto se utiliza un *switch* basado en el parámetro `$verbType`.

Para cada tipo de verbo válido (*launched*, *initialized*, etc.) se crea un array con las propiedades *id* y *display*. El *id* contiene la URI del verbo según el estándar xAPI. Mientras que *display* contiene un objeto multilenguaje para la descripción del verbo (inglés y español en este caso). Luego se asigna el array creado a la variable `$verb`.

Si el valor de `$verb` no es null (es decir, se encontró un verbo válido) se serializa el array `$verb` utilizando *serialize*.

La función retorna la cadena serializada o null en caso de no encontrar un verbo válido.

### Gestión de reportes

La gestión de reportes es exclusiva de los Docentes y se limita a la visualización las experiencias de aprendizajes mediante declaraciones xAPI.

En esta visualización se muestran la fecha y hora en la que se lanza cada uno de los verbos, el curso y la actividad a la que pertenece la solución, el estudiante que estaba realizando la actividad, el verbo lanzado y la posibilidad de descargar en un fichero con extensión ‘*json*’, la solución provista por el estudiante.

A continuación mostramos la vista del listado de las declaraciones. (Ver Figura 96)

Figura 96 Experiencia de aprendizaje

Fecha/Hora	Curso	Actividad	Estudiante	Verbo	Reporte CMIS
02-07-2024 22:05:01	Introducción a la Programación Orientada a Objetos	Métodos	Estudiante 10	<span>initialized</span>	<a href="#">+</a>
02-07-2024 22:05:01	Introducción a la Programación Orientada a Objetos	Métodos	Estudiante 10	<span>launched</span>	<a href="#">+</a>
02-07-2024 22:05:09	Introducción a la Programación Orientada a Objetos	Métodos	Estudiante 10	<span>terminated</span>	<a href="#">+</a>
02-07-2024 22:05:09	Introducción a la Programación Orientada a Objetos	Métodos	Estudiante 10	<span>failed</span>	<a href="#">+</a>
02-07-2024 22:05:37	Herencia y Polimorfismo en Programación Orientada a Objetos	Sistema de Pago y Métodos de Pago	Estudiante 10	<span>launched</span>	<a href="#">+</a>
02-07-2024 22:05:37	Herencia y Polimorfismo en Programación Orientada a Objetos	Sistema de Pago y Métodos de Pago	Estudiante 10	<span>initialized</span>	<a href="#">+</a>
02-07-2024 22:05:40	Herencia y Polimorfismo en Programación Orientada a Objetos	Sistema de Pago y Métodos de Pago	Estudiante 10	<span>failed</span>	<a href="#">+</a>
02-07-2024 22:05:40	Herencia y Polimorfismo en Programación Orientada a Objetos	Sistema de Pago y Métodos de Pago	Estudiante 10	<span>terminated</span>	<a href="#">+</a>
12-06-2024 03:51:23	Herencia y Polimorfismo en Programación Orientada a Objetos	Vehículos	Cristian Estudiante	<span>launched</span>	<a href="#">+</a>
12-06-2024 03:51:23	Herencia y Polimorfismo en Programación Orientada a Objetos	Vehículos	Cristian Estudiante	<span>initialized</span>	<a href="#">+</a>

Fuente: (Macas, 2023)

Figura 97 Código del frontend del listado de declaraciones tomadas como experiencias de aprendizaje.

```

70. <td>
71.         <strong>
72.             <em>{{ getCampo(st.object, 'definition') }}</em>
73.         </strong>
74.     </td>
75.     <td><strong><em>{{ getCampo(st.actor, 'name') }}</em></strong>
76.     </td>
77.     <td>
78.         <span class="badge badge-pill badge-success">
79.             <strong>
80.                 <i>{{getCampo(st.verb, 'en-US') }}</i>
81.             </strong>
82.         </span>
83.     </td>
84.     <td>
85.         <a href="{{ path('app_xapi_statement_pdf', {'id': st.id}) }}">
86.             <i class="fa fa-file-pdf"></i>
87.         </a>
88.         <a href="{{ path('app_xapi_statement_json', {'id': st.id}) }}">
89.             <i class="fa fa-eye"></i>
90.         </a>-->
91.         <!--button class="btn btn-info btn-sm"
92. onclick="alert('{{st.id}}')">Ver</button-->
93.         <a href="{{ path('app_xapi_statement_json', {'id': st.id}) }}"
94. download title="Descargar">
95.             <i class="fa fa-download"></i>
96.         </a>
97.     </td>
98. </td>

```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Figura 98 Código del backend. Descarga de una declaración.

```
338.      /**
339.       * @Route("xapi/json/{id}", name="app_xapi_statement_json")
340.       */
341.      public function JsonActionStatement(Statement $st)
342.      {
343.
344.          $response = new Response();
345.          $response->setContent(
346.              json_encode(
347.                  [
348.                      'data' => array(
349.                          "actor" => $st->getActor(),
350.                          "verb" => $st->getVerb(),
351.                          "object" => $st->getObject(),
352.                          "result" => $st->getResult(),
353.                          "context" => $st->getContext()
354.                      )
355.                  ], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE |
356.                  JSON_UNESCAPED_SLASHES
357.              )
358.          );
359.          $response->headers->set('Content-Type', 'application/json',
360.              'Content-Disposition', 'attachment; charset=UTF-
361.              8;filename="statement.json"'
362.          );
363.
364.          return $response;
365.      }
```

Fuente: (Macas, 2023), Fragmento del código fuente subido al repositorio de [GitHub](#) desde la máquina local.

Para descargar un fichero con las declaraciones se emplea la función *JsonActionStatement* del archivo *XApiController.php*. Esta función se encarga de convertir un objeto de declaración en una respuesta con formato JSON y devolverla. Define una ruta *xapi/json/{id}* para acceder a la acción. El *{id}* es un parámetro de ruta que será inyectado en la función. Recibe un objeto *Statement* como parámetro. Mientras que crea una respuesta HTTP.

En la implantación se construye un array de datos que incluye los campos actor, verbo, objeto, resultado y contexto del enunciado xAPI. Luego se convierte el array a formato JSON con opciones de formateo (indentación, caracteres especiales) y se establece los encabezados de la respuesta:

- Content-Type: Indica que el contenido es JSON.
- Content-Disposition: Especifica que el contenido es un archivo adjunto llamado "statement.json". El objetivo es descargar el JSON como un archivo.

Retorna la respuesta.

En la implementación además se hace uso de los métodos `$st->getActor()`, `$st->getVerb()`. Estos métodos pertenecen a la clase `Statement` y retornan los respectivos componentes del enunciado xAPI.

Se emplea también las constantes `JSON_PRETTY_PRINT` | `JSON_UNESCAPED_UNICODE` | `JSON_UNESCAPED_SLASHES`, que son constantes de formato para `json_encode`, utilizadas para mejorar la legibilidad del JSON generado.

Recibe como parámetro una declaración. En la función, primero se crea un objeto de respuestas HTTP. A este objeto se le pone como contenido un arreglo, con una serie de datos como el verbo y el objeto en formato Cmi5, entre otros. Luego establece el encabezado de la respuesta y se retorna esta respuesta.

## Otros ajustes

Uno de los ajustes necesarios para el despliegue de la aplicación, está relacionado con el recolector de basura (*Garbage collector, gc*).

Cuando se abre una sesión, PHP llamará al controlador `gc` aleatoriamente de acuerdo con la probabilidad establecida por sesión y establecen en la directiva `session.gc_probability` dicha probabilidad. Algunos sistemas operativos (por ejemplo, Debian) manejan su propia sesión y configuran la directiva en 0 para evitar que PHP realice la recolección de basura. Sin embargo Symfony sobrescribe este valor a 1. Por lo que si desea utilizar el valor original establecido en su `php.ini`, es necesario agregar la siguiente configuración en el fichero `config/packages/framework.yaml`:

```
# config/packages/framework.yaml
framework:
    session:
        gc_probability: null
```

Ahora es el sistema está listo para ser utilizado.