Universidad de Granada

PRÁCTICA 2

Adaptación de la web de la práctica Evaluable I

Antonio Jiménez Rodríguez

1. Introducción

En esta práctica final se ha procedido a implementar toda la funcionalidad back-end a la primera práctica de maquetación HTML y CSS. Para ello se ha hecho uso del framework de PHP, Symfony, desplegado en el servidor bahía por medio de contenedores. En los puntos de está documentación se desarrollarán las tecnologías usadas además de la explicación de la realización de los objetivos de la práctica haciendo uso de estas.

Además en la parte del cliente se ha hecho uso de tecnologías como JQuery y Ajax para validación y modificación del DOM.

2. Contenedores

Primero de todo para poder hacer uso de Symfony en el servidor bahía es necesario la configuración para el despliegue de los contenedores necesarios. Han sido necesarios definir cuatro servicios en nuestro **docker-compose.yml**:

- nginx-service: este servicio será el encargado de manejar las peticiones **HTTP**. Para ello deberemos añadir un archivo de configuración con el que indicar donde estará la raíz desde la que servir los archivos solicitados. En este archivo, entre otras cosas, indicaremos el puerto de escucha del contenedor (no el del servidor), el archivo que hará de index y el directorio donde se encuentra este. Por otra parte, deberemos enlazar el puerto del servidor proporcionado con el puerto de escucha configurado en el servicio, en este caso será **30300:80**.
- php-service: en esté servicio será donde instalemos PHP. Gracias a esto, podemos elegir cualquier versión de PHP y no necesariamente la que se encuentre en el servidor bahia. Instalaremos por tanto la versión **8.0.5** la cual definiremos en nuestro Dockerfile junto a otras librerías necesarias en el contenedor para la ejecución de Symfony (**Symfony Cli**, **Composer**, etc.).
- mysql-service: como se indica en el nombre, este servicio será el encargado de levantar el entorno para la base de datos con el motor de mysql. En este servicio definiremos con variables de entorno el nombre de la base de datos, el usuario y la contraseña que posteriormente definiremos en nuestro **dotenv** de la aplicación.
- node-service: por último, este servicio será utilizado para la instalación de paquetes con yarn como pueden ser **Bootstrap** ó **JQuery**. Además por un módulo que usaremos para la compilación de los archivos estáticos JS y CSS será necesario este servicio.

3. Symfony

He elegido Symfony como framework para esta práctica final por la experiencia en él y las facilidades que aporta a la hora de construir una web e implementar el CRUD y manejo del modelo de datos. Junto a este se hará uso del motor de plantillas **Twig** con lo que se podrá modularizar y encapsular todos los trozos de código HTML que se repitan en las mismas páginas como puede ser el header o el footer entre otros.

Symfony se compone de **Bundles**. Un Bundle es un paquete de funcionalidades (una librería) que proporciona facilidades a la hora de trabajar distintos aspectos de una aplicación. Para el manejo del modelo de datos haremos uso de Doctrine. Este bundle nos permitirá añadir anotaciones a las entidades definidas con las que se construirá nuestra base de datos. Es algo muy útil ya que por ejemplo, en relaciones de tablas muchos a muchos no necesitaremos implementar la tabla pivote ya que symfony será el encargado de manejarla. Para sacar partido a estas automatizaciones usaremos los repositorios de las entidades en los cuales definiremos funciones (sin sentencias SQL) para acceder a los diferentes datos almacenados. También destacar la facilidad para crear, actualizar o eliminar una entidad ya que únicamente debemos trabajar con la instancia de la clase que hayamos creada y con el llamado EntityManager persistir o eliminar la entidad además de llamar al método flush() que es el encargado de hacer el **commit SQL** en base de datos de los cambios indicados.

Otra facilidad que nos brinda este framework de **MVC** es el tema del enrutado. Para definir funciones en diferentes path del servidor lo único que debemos indicar en la documentación de la función es el path desde el cual se accede. Además los parámetros de las funciones de ruta serán auto inyectados por lo que no tendremos que preocuparnos de instanciar estos previamente. Las respuestas de estas rutas serán recogidas por las plantillas de Twig que indiquemos en la respuesta, pudiendo pasar objetos con los que construir la vista de una manera relativamente sencilla.

También cabe destacar la forma de trabajar con los archivos de nuestra carpeta assets. Con el bundle **Webpack** podremos definir un archivo de configuración el cual será el encargado de compilar los archivos **JavaScript** y hojas de estilo de **Sass** y moverlos a la carpeta pública donde podemos encontrar nuestro index.php. Además si lo deseamos también moverá nuestra imágenes que añadamos a esta carpeta pública haciéndolas accesibles desde cualquier punto.

4. Modelo de datos

Para nuestro modelo de datos hemos definido cinco entidades las cuales vamos a definir a continuación.

ImageFile

Debido a la necesidad de subir imágenes para añadirlas a una exposición o una pieza se ha creado la entidad ImageFile. Como bien sabemos cada imagen puede tener un tamaño diferente y para la web necesitamos saber el tamaño de cada imagen para no mostrarlas distorsionadas por adaptarse a una medida estándar. Por tanto en esta entidad guardaremos el SRC de la imagen (nombre de la imagen, ejemplo: imagen.png), y la anchura y altura en píxeles (valores enteros). Esta entidad no tendrá referencias a otras desde ella ya que una instancia de una imagen es independiente.

Piece

Esta entidad almacena toda la información relevante a una pieza de una exposición. En ella podremos ver las propiedades de nombre, autor, descripción y una relación con la entidad ImageFile. Estas relaciones entre entidades serán manejadas por el bundle de Doctrine que se encargará de añadir las foreign keys necesarias y recuperar los datos de la base de datos a la hora de hacer las consultas.

```
/**
 * @ManyToOne(targetEntity="App\Entity\ImageFile")
 * @JoinColumn(name="image_file_id", referencedColumnName="id", nullable=false)
 */
protected ImageFile $imageFile;
```

Comment

En la entidad Comment almacenaremos los comentarios que un usuario realice sobre una exposición. Por lo tanto deberemos almacenar la relación entre comentario y exposición, comentario y usuario, y el mensaje introducido en el comentario. Además almacenaremos la fecha de creación con la que podremos ordenar estos en la vista de la web.

Exhibition

En esta entidad almacenaremos la información relativa a las exposiciones. Es la entidad principal ya que hace de enlace entre las piezas y los comentarios creados para mostrar. Al igual que en la entidad Piece tendremos una referencia a un ImageFile con lo que se definirá la imagen de la exposición.

Además de sus campos de texto como el nombre, autor, descripción, etc. tendremos una nueva propiedad denominada Collections. Esto registra todos los comentarios y precias asociados a la exposición ya que se trata de relaciones OneToMany como podemos ver:

```php
/**
 * @ORM\OneToMany(targetEntity="App\Entity\Comment", mappedBy="exhibition", cascade={"remove"})
 */
private Collection $comments;

/**
 * @ORM\OneToMany(targetEntity="App\Entity\Piece", mappedBy="exhibition", cascade={"remove"})
 */
private Collection $pieces;
```

User

Por último la tabla User almacenará todos los datos respectivos a un usuario de la web. En esta se añadirá la contraseña hasheada y un array serializado con los permisos de cada usuario sobre la aplicación.

Estructura de tablas

En la siguiente imagen podemos ver la estructura de cada una de las tablas descritas, con sus tipos de datos y sus configuraciones. Los cambios realizados en base de datos se harán a través de migraciones. Esta funcionalidad permite añadir nuevas propiedades a una entidad o modificar ya existentes, lanzar un comando desde la shell que compare la base de datos actual con las nuevas anotaciones de código y generar un archivo ejecutable con sentencias SQL las cuales realizarán las modificaciones pertinentes en base de datos. Esto facilita a la hora de crear grandes tablas y de crear relaciones o índices ya que Symfony con esta funcionalidad lo hace de manera automática.

```
mysql> describe comment;
+---------------+----------+------+-----+---------+----------------+
| Field         | Type     | Null | Key | Default | Extra          |
+---------------+----------+------+-----+---------+----------------+
| id            | int      | NO   | PRI | NULL    | auto_increment |
| exhibition_id | int      | NO   | MUL | NULL    |                |
| user_id       | int      | NO   | MUL | NULL    |                |
| text          | longtext | NO   |     | NULL    |                |
| comment_at    | datetime | NO   |     | NULL    |                |
+---------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> describe exhibition;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| id            | int          | NO   | PRI | NULL    | auto_increment |
| image_file_id | int          | NO   | MUL | NULL    |                |
| author        | varchar(512) | NO   |     | NULL    |                |
| start_at      | datetime     | NO   |     | NULL    |                |
| room          | varchar(10)  | NO   |     | NULL    |                |
| description   | longtext     | NO   |     | NULL    |                |
| name          | varchar(512) | NO   | UNI | NULL    |                |
| category      | int          | NO   |     | 0       |                |
+---------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)

mysql> describe image_file;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int          | NO   | PRI | NULL    | auto_increment |
| src    | varchar(512) | NO   |     | NULL    |                |
| width  | int          | NO   |     | 0       |                |
| height | int          | NO   |     | 0       |                |
+--------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> describe piece;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| id            | int          | NO   | PRI | NULL    | auto_increment |
| name          | varchar(512) | NO   |     | NULL    |                |
| exhibition_id | int          | NO   | MUL | NULL    |                |
| image_file_id | int          | NO   | MUL | NULL    |                |
| author        | varchar(512) | NO   |     | NULL    |                |
| description   | longtext     | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql> describe user;
+-----------+---------------+------+-----+---------+----------------+
| Field     | Type          | Null | Key | Default | Extra          |
+-----------+---------------+------+-----+---------+----------------+
| id        | int           | NO   | PRI | NULL    | auto_increment |
| roles     | json          | NO   |     | NULL    |                |
| email     | varchar(512)  | NO   | UNI | NULL    |                |
| password  | varchar(512)  | NO   |     | NULL    |                |
| surname   | varchar(1024) | NO   |     | NULL    |                |
| name      | varchar(1024) | NO   |     | NULL    |                |
| birthdate | datetime      | YES  |     | NULL    |                |
+-----------+---------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

5. Objetivos de la práctica

CRUD de entidades

La estructura en el CRUD del modelo de datos definido será la misma. Deberemos definir un **FormType**, el cual será el encargado de generar el formulario para cada una de las entidades con sus respectivos campos. Este formulario se enlazará directamente con los atributos de cada entidad por lo que la información introducida en el formulario se mapeará automáticamente. Estos FormTypes su utilizarán en cada una de las rutas definidas para crear o editar una entidad (para la eliminación y la lectura no será necesaria ya que el formulario es para añadir o modificar datos). En cada ruta se tratarán los datos del **request** de la manera que corresponda. Esto solo será necesario en campos no mapeados como por ejemplo las imágenes en las exposiciones o en las piezas, ya que los campos definidos en el formulario no se corresponden directamente con la entidad. Por último será necesario crear los templates para la renderización de estos formularios. En algunos casos algunos de los datos como labels o requerimientos de inputs no serán los mismos en la creación o edición, por lo tanto, estos se modificarán desde los templates de cada vista.

```php
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add( child: 'name',  type: TextType::class, array(
            'label' => 'Nombre:'
        ))
        ->add( child: 'author',  type: TextType::class, array(
            'label' => 'Autor:'
        ))
        ->add( child: 'description',  type: TextareaType::class, array(
            'label' => 'Descripción:'
        ))
        ->add( child: 'imageSrc',  type: FileType::class, array(
            'label' => 'Imagen:',
            'mapped' => FALSE,
            'required' => TRUE,
        ))
        ->add( child: 'width',  type: IntegerType::class, array(
            'label' => 'Anchura de la imagen:',
            'mapped' => FALSE,
            'required' => TRUE,
        ))
        ->add( child: 'height',  type: IntegerType::class, array(
            'label' => 'Altura de la imagen:',
            'mapped' => FALSE,
            'required' => TRUE,
        ))
        ->add( child: 'submit',  type: SubmitType::class, array(
            'label' => 'Crear'
        ))
    ;
}
```

Construcción de formulario con ExhibitionFormType

```twig
{% block form %}
    {{ form_start(form) }}
    <section class="form-section form-section-info">
        <h4><i class="fas fa-solid fa-circle-info"></i> Información general</h4>
        {{ form_row(form.name) }}
        {{ form_row(form.author) }}
        {{ form_row(form.description) }}
    </section>
    <section class="form-section form-section-detail">
        <h4><i class="fas fa-solid fa-asterisk"></i> Detalles</h4>
        {{ form_row(form.startAt) }}
        {{ form_row(form.room) }}
        {{ form_row(form.category) }}
    </section>
    <section class="form-section form-section-image">
        <h4><i class="fas fa-solid fa-image"></i> Imagen</h4>
        {{ form_row(form.imageSrc, {
            required: false,
            help: '* Si no se añade ninguna imagen nueva se mantendrá la actual.'
        }) }}
        <p id="validate-image-file" class="help-text font-bold"></p>
        {{ form_row(form.width, {
            required: false
        }) }}
        {{ form_row(form.height, {
            required: false
        }) }}
    </section>
    <section class="form-section form-section-submit">
        {{ form_row(form.submit, {
            label: 'Editar'
        }) }}
    </section>
    {{ form_end(form) }}
{% endblock %}
```

Modificación de ExhibitionFormType en template

Usuarios

Para los usuarios del sistema algo diferencial en el CRUD es la encriptación de la contraseña en base de datos. Esto se puede manejar de una manera sencilla con la interfaz proporcionada por el SecurityBundle, **UserPasswordHasherInterface**. Está coge el algoritmo de codificación indicado en el archivo de configuración del Bundle y convierte el texto plano introducido por el usuario en un hash indescifrable. Además debido a los tipos de roles de usuario que puede soportar el sistema se ha añadido una comprobación en la ruta en la que sí el usuario con la sesión es de rol administrador o comisario, podrá registrar otros comisarios en el sistema.

Exposiciones y Piezas

Para las exposiciones y piezas tenemos lo comentado anteriormente, existen campos en el formulario que no están directamente mapeados con la entidad de cada una. Estos son la imagen que se suba al servidor junto con su anchura y su altura. Para tratar estos datos será necesario implementar funcionalidad que recoja los valores introducidos de la instancia **Request** inyectada en la ruta. Para la imagen se obtendrá el nombre de ella y se moverá a la carpeta del servidor que corresponda. Con estos datos se construirá la entidad ImageFile la cual se establecerá en la exposición o pieza correspondiente y se persistirán ambas. Así conseguiremos los enlaces entre entidades gracias a las **FOREIGN KEYs** definidas en la base de datos.

Comentarios

Para añadir comentarios no será necesario de un FormType debido a su simplicidad. Añadiremos un simple formulario en el template de exposición en el que añadiremos como campo oculto el ID de la exposición a la que corresponderá el comentario. Este se recuperará en la ruta desde el Request y se creará la instancia del comentario, persistiendose y añadiendose a la base de datos.

Inicio de sesión

Gracias al bundle usado para la autenticación y registro de usuario el inicio de sesión será relativamente sencillo. Deberemos indicar cuál es la ruta de inicio de sesión a la que el formulario llamará e indicar el nombre de los inputs del usuario y contraseña en las que se enviarán los datos. Internamente se autenticará y se creará la sesión para el usuario. Únicamente debemos añadir a nuestra ruta en el controlador **SecurityController** la acción que se realizará una vez autenticado o no el usuario (en nuestro caso será una redirección con un mensaje de validación).

Validaciones JavaScript

Los formularios creados a partir de FormTypes llevan una serie de restricciones o validaciones según las clases que usemos para ello. Sin embargo no está de más añadir validaciones con JavaScript para que el cliente no tenga que realizar la petición y esperar a que esta le confirme el formulario. Se ha usado JQUERY con eventos como **keyup** y **onchange** para estas validaciones.

Formulario de Usuario

Para el formulario de registro o edición de usuarios se usa una validación con Ajax para comprobar que el usuario introducido en la base de datos no existe previamente. En este caso para una práctica no es algo muy útil, sin embargo en una gran aplicación con muchos usuarios facilitará a la persona a encontrar un nombre más rápidamente sin necesidad de mandar el formulario completo una cantidad ingente de veces. Para esto hemos creado un **endpoint** en nuestra aplicación, que recibirá el nombre de usuario, y en caso de estar editando su perfil el ID, y comprobará si existe algún usuario con esos valores. Se devolverá una respuesta en formato JSON y en caso de existir se indicará al usuario que debe de elegir otro nombre y se deshabilitará el botón de submit para evitar envíos.

```
$.ajax( url: {
    type: 'GET',
    url: '/check/user',
    data: {
        id: $('#user-id').val(),
        email: $('#registration_form_email').val()
    }
}).done(function (result) {
    if (result.data) {
        $validateEmailUser.html('* Email disponible');
        $validateEmailUser.addClass('color-success');
        $validateEmailUser.removeClass('color-danger');
        emailError = false;
        checkingUserName = false;
        $('.form-container button').prop(
            'disabled', emailError || nameError || surnameError || checkingUserName
        );
    } else {
        $validateEmailUser.html('* Email ya registrado');
        $validateEmailUser.removeClass('color-success');
        $validateEmailUser.addClass('color-danger');
        emailError = true;
        checkingUserName = false;
        $('.form-container button').prop(
            'disabled', emailError || nameError || surnameError || checkingUserName
        );
    }
}).fail(function (jqXHR, textStatus, errorThrown) {
    $('#validate-user').html();
});
```

Campos de texto en formulario

Se han añadido también validaciones para campos que sean únicamente de texto como pueden ser los nombres o apellidos de un usuario. Se recoge el valor del campo y con una expresión regular se comprobará si son o no únicamente caracteres alfabéticos. En el caso de no serlo, se añadirá un mensaje de error bajo el campo y se bloqueará el botón de submit al igual que en el caso anterior.

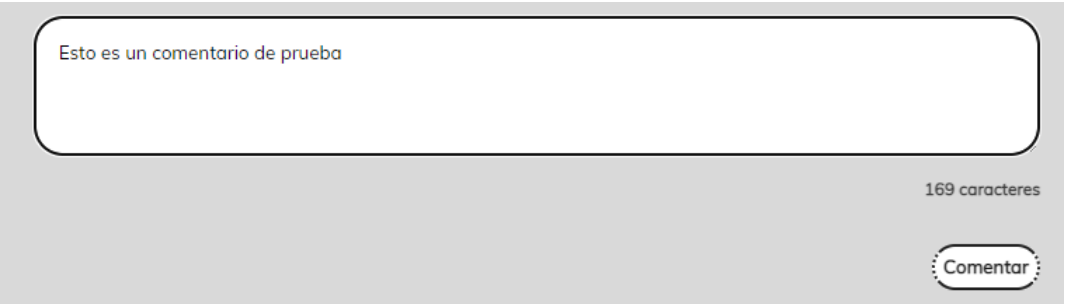
Campo de subida de archivo

Debido a que solo deseamos subir imágenes al servidor para las piezas y las exposiciones, se ha añadido validación para impedir subir archivos con extensiones diferentes a las de imágenes. Al igual que en los anteriores, si se sube una imagen incorrecta se mostrará el correspondiente mensaje de error y se bloqueará el botón de submit.

Comentarios

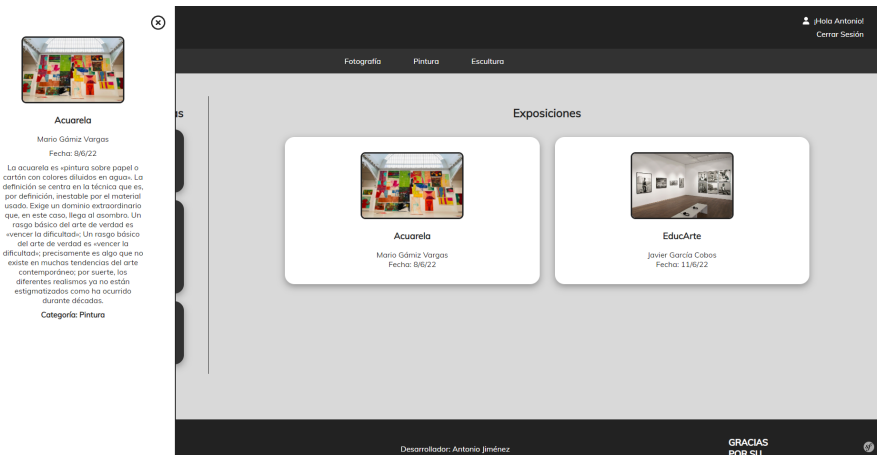
Por las restricciones indicadas en la práctica se ha añadido un límite de caracteres en la caja de comentarios de cada exposición. Gracias al input del mensaje y a los inputs ocultos añadidos se hará un cálculo de los caracteres restantes en la pantalla y se impedirá añadir un mensaje de mayor longitud que la indicada truncando el mensaje.

```
// Validación comentario
$('#box-comment').keyup(function (e) {
    let $commentContainer = $('#box-comment'),
        numChars = $commentContainer.val().length,
        maxChars = $('#max_chars').val(),
        $countContainer = $('.comment-characters-count');

    $countContainer.html(maxChars - numChars);
    $commentContainer.val($commentContainer.val().substring(0, maxChars));
});

$('.commentsForm').submit(function (e) {
    if ($('#box-comment').val().length == 0) {
        e.preventDefault();
    }
});
```



PopUp en Home

Otro de los objetivos de esta práctica era añadir una ventana emergente cuando se pasara por encima de una de las exposiciones de la página de inicio. Existen varias opciones para realizar esto como puede ser hacer uso de bootstrap donde ya existen modales prediseñados donde solo tienes que añadir el código HTML y definir el evento que deseas que se lance. Sin embargo he decidido realizar una implementación desde 0 con un efecto de deslizar para darle más dinamismo.



Para realizar esto se ha hecho uso de la propiedad CSS transform la cual se puede añadir una función CSS que desplaza el el contenedor en el eje indicado. Si esto lo juntamos también con el uso del transition podemos obtener un efecto interesante con el que mostrar los PopUps de la página de inicio. Usando JQuery junto al evento **onMouseEnter** y **onMouseLeave** modificaremos estas propiedades para que el modal se oculte o se muestre según nuestras acciones.

```
// Evento de Home
$('.expo').mouseenter(function (e) {
    let id = $(this).find('#expo-id').val(), $container, $close, scrollPosition;

    $container = $('.expo-modal-' + id);
    $container.removeClass('d-none');
    setTimeout( handler: function () {
        $container.find('.expo-modal-info').css('transform', 'translateX(0)');
    },  timeout: 100);

    $close = $('.close-modal-' + id);
    $close.click(function (e) {
        $container.addClass('d-none');
        $container.find('.expo-modal-info').css('transform', 'translateX(-150%)');
    });

    $('.expo').mouseleave(function (e) {
        $container.addClass('d-none');
        $container.find('.expo-modal-info').css('transform', 'translateX(-150%)');
    });
});
```

Comandos para el uso de los contenedores

Entrar al bash del contenedor:

- podman exec -it php-container bash
- podman exec -it mysql-container bash
    - mysql -uroot -psecret

Compilar los archivos JS y CSS:

- podman-compose run --rm node-service yarn encore dev

Instalar un Bundle:

- podman-compose exec php-service composer require XXXXXX

Instalar un paquete de node:

- podman-compose exec app yarn add XXXXXX

Levantar el contenedor:

- podman-compose up -d --build

Apagar el contenedor:

- podman-compose down -v

Migraciones:

- docker-compose exec php-service bin/console doctrine:migrations:diff
- docker-compose exec php-service bin/console doctrine:migrations:execute --up DoctrineMigrations\Version20220611081059