Symfony

Especialidad PHP

Felipe Izquierdo Romero

Índice

1 Descripción del problema

2 Desarrollo

1. Descripción del problema

Crear una app usando Symfony que permita gestionar eventos. La app podrá ofrecer al usuario la capacidad de crear eventos y visualizarlos en una lista además de visualizar de manera individual cada evento con todos sus datos.

2. Desarrollo

En primer lugar se mostrará las diferentes vistas que heredan todas del documento "base.html.twig". Cada vista cumple con las funciones de mostrar los eventos, mostrar los detalles de un evento y crear un evento.

La plantilla base contiene la estructura del html y el link a la hoja de estilo css que usarán las plantillas descendientes.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Blog Symfony{% endblock %}</title>
    {# link para cargar css #}
    <link rel="stylesheet" href="{{ asset('css/styles.css') }}">
</head>
<body>
    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

Ilustración 1- Vista: plantilla base

Para mostrar los eventos se mostrará el título del evento con un bucle for each en una lista desordenada.

```
{% extends 'base.html.twig' %}

{% block title %}Lista de Eventos{% endblock %}

{% block content %}
    <h1>Lista de Eventos</h1>
    <ul>
        {% for evento in eventos %}
            <li>
                <h2>
                    <a href="{{ path('eventos_show', {'id': evento.id}) }}">{{ evento.nombre }}</a>
                </h2>
            </li>
        {% else %}
            <li>No hay eventos disponibles.</li>
        {% endfor %}
        <button type="button" class="btn btn-primary" onclick="window.location.href='{{ path('evento_create') }}'">Añadir Evento</button>
    </ul>
{% endblock %}
```

Ilustración 2- Vista: lista de eventos

La creación de un evento será recogida con este formulario en una lista desordenada con el input de los datos debajo de su correspondiente etiqueta.

```twig
{% extends 'base.html.twig' %}

{% block title %}Crear Nuevo Evento{% endblock %}

{% block content %}
    <h1>Crear Nuevo Evento</h1>
    <ul>

        {{ form_start(form) }}
            <li>
                {{ form_label(form.nombre) }}<br>
                {{ form_widget(form.nombre) }}
            </li>
            <li>
                {{ form_label(form.fecha) }}<br>
                {{ form_widget(form.fecha) }}
            </li>
            <li>
                {{ form_label(form.hora)}}<br>
                {{ form_widget(form.hora)}}
            </li>
            <li>
                {{ form_label(form.ubicacion) }}<br>
                {{ form_widget(form.ubicacion) }}
            </li>
            <li>
                {{ form_label(form.detalles) }}<br>
                {{ form_widget(form.detalles) }}
            </li>

            <button type="submit" class="btn btn-primary">Guardar</button>
            <button type="button" class="btn btn-primary" onclick="window.location.href='{{ path('index') }}'">Volver a la lista</button>
        {{ form_end(form) }}
    </ul>
```

Ilustración 3- Vista: formulario de creación de evento

Pasamos a ver como se mostrarán los datos de manera sencilla con un título y un párrafo con una línea por cada etiqueta y sus valores.

```twig
{% extends 'base.html.twig' %}

{% block title %}Detalles del Evento{% endblock %}

{% block content %}
    <h1>{{ evento.nombre }}</h1>
    <p>Fecha: {{ evento.fecha|date('d-m-Y') }}</p>
    <p>Hora: {{ evento.fecha|date('H:i') }}</p>
    <p>Ubicación: {{ evento.ubicacion }}</p>
    <p>Detalles: {{ evento.detalles }}</p>
    <button type="button" class="btn btn-primary" onclick="window.location.href='{{ path('index') }}'">Volver a la lista</button>
{% endblock %}
```

Ilustración 4- Vista: mostrar datos de un evento específico

A continuación pasaremos a ver el controlador y las rutas definidas de dos maneras con un archivo “routes.yaml” o usando el componente Route que te permite añadir las rutas encima de las funciones como si fueran comentarios.

```yaml
index:
    path: /
    controller: App\Controller\EventosController::index
eventos_show:
    path: /eventos/{id}
    controller: App\Controller\EventosController::show
evento_create:
    path: /eventos/create
    controller: App\Controller\EventosController::create
```

Ilustración 5- Rutas definidas en routes.yaml

Este sería un ejemplo de la definición con Route en el propio controlador en forma de comentario.

```
22      /**
23       * @Route("/", name="index")
24       */
```

Ilustración 6- Ruta definidas en el controlador

En controlador consta de tres funciones y el atributo entityManager de tipo "EntityManagerInterface", para comunicar con el repositorio que extrae e inserta registros en la base de datos.

La función index que recoge los eventos de la base de datos y los carga en la plantilla "eventos.html.twig".

```
25      public function index(): Response
26      {
27          $eventos = $this->entityManager->getRepository(Evento::class)->FindAll();
28          return $this->render('eventos/eventos.html.twig',[
29              'eventos' => $eventos,
30          ]);
31      }
```

Ilustración 7- Función index

La función show busa por id el evento que se ha clicado y envía los datos a la plantilla "show.html.twig".

```
41      public function show($id): Response
42      {
43          $eventoId = (int)$id;
44          $evento = $this->entityManager->getRepository(Evento::class)->find($eventoId);
45
46          if (!$evento) {
47              throw $this->createNotFoundException('La publicación no se encontró');
48          }
49          return $this->render('eventos/show.html.twig', ['evento' => $evento]);
50      }
```

Ilustración 8- Función show

Por último la función create recoge y valida los datos del formulario de la plantilla “créate.html.twig”, unifica los valores de fecha y día para guardarlos en un solo campo en la tabla eventos y redirige hacia la plantilla “eventos.html.twig” definida como index.

```php
    public function create(Request $request): Response
    {
        $evento = new Evento();
        $form = $this->createForm(EventoType::class, $evento);

        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $evento->setFecha($evento->combinarFechaYHora());
            $this->entityManager->persist($evento);
            $this->entityManager->flush();

            $this->addFlash('success', 'El evento se ha creado correctamente.');

            return $this->redirectToRoute('index');
        }

        return $this->render('eventos/create.html.twig', [
            'form' => $form->createView(),
        ]);
    }
```

Ilustración 9- Funcón create

El repositorio que comunica con la base de datos a través de un ORM define arriba los metodos:

- find
- findOneBy
- findAll
- findBy

```php
/**
 * @extends ServiceEntityRepository<Post>
 *
 * @method Post|null find($id, $lockMode = null, $lockVersion = null)
 * @method Post|null findOneBy(array $criteria, array $orderBy = null)
 * @method Post[]    findAll()
 * @method Post[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
```

Ilustración 10- Metodos ORM

Y define la clase EventoRepository que hereda de ServiceEntityRepository con un constructor y dos funcions: add y remove.

```php
class EventoRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Evento::class);
    }

    public function add(Evento $entity, bool $flush = false): void
    {
        $this->getEntityManager()->persist($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }

    public function remove(Evento $entity, bool $flush = false): void
    {
        $this->getEntityManager()->remove($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }

}
```

Ilustración 11- Repositorio: EventoRepository

Para concluir con este laboratorio se mostrará el modelo que consta de dos clases "Evento.php" que mapea los valores de cada columna de un registro de la tabla Eventos y proporciona los getter y setter y la función "combinarFechaYHora".

```php
/**
 * @ORM\Entity(repositoryClass="App\Repository\EventoRepository")
 */
class Evento
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */

    private $nombre;
    /**
     * @ORM\Column(type="string", length=255)
     */

    private $ubicacion;
    /**
     * @ORM\Column(type="datetime")
     */

    private $fecha;

    private $hora;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $detalles;
```

Ilustración 12- Mapeo de atributos

```php
    public function combinarFechaYHora()
    {
        // Combinar fecha y hora en un solo objeto DateTime
        $fechaHora = new \DateTime();
        $fechaHora->setDate($this->fecha->format('Y'), $this->fecha->format('m'), $this->fecha->format('d'));
        $fechaHora->setTime($this->hora->format('H'), $this->hora->format('i'));

        return $fechaHora;
    }
}
```

Ilustración 13- Combinar fecha y hora

Definición de la tabla eventos en el archivo init.sql

```sql
SET NAMES utf8;
SET time_zone = '+00:00';
SET foreign_key_checks = 0;
SET sql_mode = 'NO_AUTO_VALUE_ON_ZERO';

SET NAMES utf8mb4;
-- DROP TABLE IF EXISTS `evento`;
CREATE TABLE IF NOT EXISTS evento (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) COLLATE utf8_unicode_ci NOT NULL,
    ubicacion VARCHAR(255) COLLATE utf8_unicode_ci NOT NULL,
    fecha DATETIME NOT NULL,
    detalles VARCHAR(255) COLLATE utf8_unicode_ci NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8_unicode_ci;
```

Ilustración 14- Tabla Evento

La clase “EventoType.php” genera el formulario definiendo los inputs para almacenar los valores de un Evento.

```php
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nombre', TextType::class, [
                'required' => true,
                'label' => 'Nombre del Evento'
            ])
            ->add('fecha', DateType::class, [
                'widget' => 'single_text',
                'required' => true,
                'label' => 'Fecha del Evento',

            ])
            ->add('hora', TimeType::class, [
                'widget' => 'single_text',
                'required' => true,
                'label' => 'Hora del Evento',
                'attr' => [
                    'min' => 0,
                    'max' => 23,
                    'class' => 'form-control',
                ],
                'attr' => [
                    'min' => 0,
                    'max' => 59,
                    'class' => 'form-control',
                ],
            ])
            ->add('ubicacion', TextType::class, [
                'required' => true,
                'label' => 'Ubicación'
            ])
            ->add('detalles', TextareaType::class, [
                'required' => true,
                'label' => 'Detalles'
            ]);
    }
```

Ilustración 13- Formulario de Evento