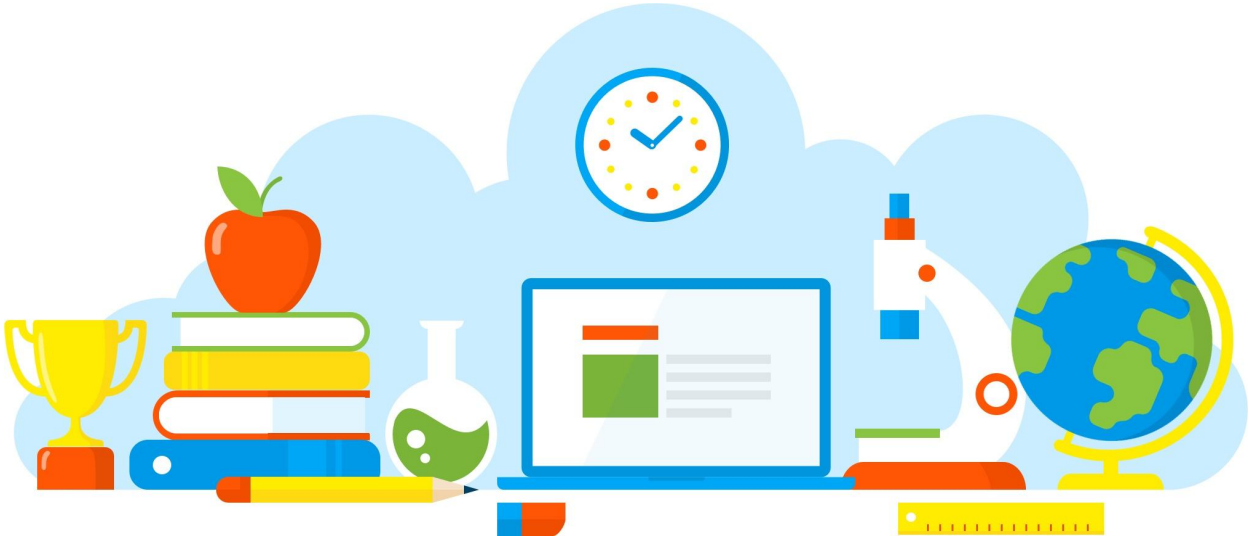

Trabajo de Symfony

Simulación de base de datos de una plataforma de streaming de música

Para comprobar esta práctica necesitarás XAMPP instalado en tu ordenador, un editor de código, laravel y symfony en su última versión (a poder ser) y Apache y MySql activados en el XAMPP Control Panel.

Pablo Lestau Martín
17/02/2023

Componentes más relevantes

Los componentes más importantes de mi proyecto serían el Home, Canción, ya que hereda tanto de Cantante como de Disco, y por último Disco, ya que hereda de Cantante, tanto los controllers como las vistas, y claro, el Login y el Registro, en ese orden.

HomeController

En el HomeController se procesan la mayoría de operaciones importantes de este proyecto, muestra las canciones más nuevas en orden de novedad, y los cantantes y las canciones más escuchadas por orden de reproducciones. El cantante guarda las reproducciones totales de todas sus canciones. También se encuentra aquí la barra de buscador que muestra los resultados de la búsqueda debajo, tanto de canciones, como de discos o cantantes. En la función Index se encuentran las operaciones SQL.

Canción, Disco y Cantante

En estos, al loguearte (ya que si no, no se te mostrarán en la barra de navegación) podrás crear cantantes, discos y canciones con sus datos y relacionados entre sí, que se mostrarán en la vista Home y poder interactuar con ellos.

Login y registro

Componentes creados por comandos (guía a continuación) en el que permite al usuario crear un perfil y loguearse en él, recibir un correo confirmando que funciona, y por último, complementarse muy bien con

La barra de navegación

Ya que al estar logueado se muestra el contenido de { % if app.user % } que es Canciones, Cantante, y Disco, sin loguearte no deberías poder acceder al resto de la página más allá de la página de inicio.

El footer y el header están añadidos estáticamente en la vista base, no requieren mucha explicación.

Guia de realización

Pasos a seguir para reproducir mi trabajo. **Todos los comandos son sin comillas " ".**

Creación de proyecto y de base de datos

1.- Configurar el entorno de desarrollo, asegurarse de tener symfony y crear un proyecto con 'composer create-project symfony/skeleton "spotyFake"en la carpeta que queramos crear el proyecto. Después, accedemos a la carpeta que acabamos de crear desde la consola de comandos (cd …) e instalamos webapp con "composer require webapp". Editar el archivo "env" para que podamos conectar nuestro proyecto con la base de datos MySql. Antes de seguir, usar el comando "composer require symfony/orm-pack". (necesario para la creación de controladores y vistas)

2.- Crear la base de datos con el comando"php bin/console doctrine:database:create" para inicializar la base con el nombre del env, y luego creamos las tablas de las bases de datos y establecer una relación entre ellas, en este caso con el comando "php bin/console make:entity", ingresamos el nombre de la tabla y luego sus atributos, junto a su tipo, longitud, etc. Para relacionar las tablas, usar el tipo de atributo "relation" e indicar a qué entidad se refiere. Una vez ya hayamos creado las tablas y sus relaciones, estamos listos para iniciar la migración e insertar las tablas en la base de datos. Usamos el comando "php bin/console make:migration" para crear una versión de la migración y "php bin/console doctrine:migrations:migrate" para confirmar los cambios. Comprobamos si se han realizado los cambios en algún programa o en localhost/phpmyadmin, y si se han realizado, estamos listos para crear el CRUD a partir de nuestras tablas con el comando "php bin/console make:crud". Si has seguido correctamente todos los pasos, ya tendrás las tablas creadas y podrás acceder a cantante (a canción y disco no porque debemos editarlas para coger las claves foráneas)

3.- Editamos “CancionType” y “DiscoType” para que, en el formulario de creación de estos, salgan los datos foráneos pertenecientes a otras tablas, ¿cómo? pues cuando en la entidad pone “add->(claveforanea)” le añadimos “, null, array(‘class’ => “Dirección” “choice_label” => “nombre”, ‘multiple’ => false, “expanded”, “require” y “placeholder”...)

```php
class CancionType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('titulo')
            ->add('genero')
            ->add('duracion')
            ->add('cantante', null, array(
                'class' => 'App\Entity\Cantante',
                'choice_label' => 'nombre',
                'multiple' => false,
                'expanded' => false,
                'required' => true,
                'placeholder' => 'Seleccione un cantante'
            ))
            ->add('disco', null, array(
                'class' => 'App\Entity\Disco',
                'choice_label' => 'titulo',
                'multiple' => false,
                'expanded' => false,
                'required' => true,
                'placeholder' => 'Seleccione un disco'
            ))
        ;
    }
```

4.- Ya hemos creado y relacionado la base de datos, ahora a crear la página de inicio que se muestre al cargar la página principal. Y hacemos que se muestran las últimas novedades, además de las canciones más escuchadas, y un buscador para cada tabla de la base de datos. Creamos el controlador HomeController con “php bin/console make:controller”, y editamos el método index() para agregar la lógica que nos muestra las novedades y tendencias.

Para mostrar las canciones más novedosas, en index llamamos a findBy() del repositorio de la entidad Cancion para recuperar las 10 canciones más recientes.

Para mostrar las canciones más escuchadas, tenemos que reproducir el apartado anterior pero cambiando el id por reproducciones (atributos de la tabla Cancion) y añadiendo el resultado al render.

5.- Creamos un formulario de búsqueda de canciones, discos y cantantes y buscar en la base de datos resultados concluyentes para mostrarlos en la vista, en el index.html.twig. En cancionController/cantanteController/discoController añadimos la función buscarCanciones/cantante/disco etc y una ruta para acceder a ella, crear la plantilla de formulario en home/index.html.twig para mostrar el formulario de búsqueda. y añadir una acción de controlador para que se encargue de procesar todos los datos del formulario y realizar la búsqueda. En la entidad correspondiente añadir una función para procesar las búsquedas. Y por último, crear una plantilla para mostrar los resultados de la búsqueda.

6.- Añadir un icono que se muestra junto a cada canción, que al pulsar nos manda a reproducir/idcancion que nos muestra el título y el autor de la canción que has pulsado. Añadiendo el icono en el index y añadiendo la función reproducir en CancionController y actualizar su contador de visitas. Crear una plantilla “reproduciendo.html.twig” e introducir los datos que le pasamos al pulsar el icono (el id de la canción y del cantante)

Añadir barra de navegación (y footer)

En mi caso he creado una vista “navigation” que estará incluida en la vista base.html.twig con {% block nombreBloque %} con un {% include ‘nombreArchivoVista.html.twig” %}.

```
    <body>
        {% block navBar %}
            {% include 'navigation.html.twig' %}
        {% endblock %}
        {% block body %}{% endblock %}
        {% block footer %}
            {% include 'footer.html.twig' %}
        {% endblock %}
    </body>
</html>
```

ya una vez esté agregada en el archivo base, damos forma a la vista creada para que, en este caso, al pulsar en ciertos campos, nos lleve a las vistas correspondientes.

Para agregar el footer es repetir los mismos pasos y colocarlo en el archivo base en el orden que queramos (primero navbar, luego body y por último, footer).

Añadir archivo css

Si deseas añadir un archivo css para darle estilo a tu proyecto, debes usar “composer require symfony/asset” en la consola con la carpeta de tu proyecto abierta, una vez instalado, creamos el archivo css en la carpeta /public y por último, en el <% block stylesheets %> de base.html.twig añadir “<link rel="stylesheet" href="{{ asset(nombredelarchivo.css') }}"> ” para usar el archivo en nuestro proyecto.

Ahora somos capaces de usar nuestro propio archivo css, pero para poder usarlo en la vista, debemos añadirle el block stylesheets, escribiendo {{% block stylesheets %}} y por supuesto el {{% endblock %}}, nada más, ya que copiará el contenido de base.html.twig.

Añadir login y registro

Para agregar un componente para procesar el login y registro de usuarios, necesitaremos el bundle de seguridad de symfony instalado, para ello, ejecutamos el siguiente comando en la consola dentro de la carpeta de nuestro proyecto. “composer require symfony/security-bundle“.

Creamos y configuramos la entidad “User” para la autenticación de usuarios almacenados en la base de datos. Lo podemos generar automáticamente con “php bin/console make:user”. Nos preguntará por un nombre, el que queramos y en los siguientes pasos colocar la respuesta que nos da por defecto en amarillo y paréntesis.

Comprueba los cambios en el fichero /config/packages/security.yaml y que se han creado tanto el modelo como el repositorio para User. y para crear las tablas php bin/console make:migration php bin/console doctrine:migrations:migrate. Esto es para que surjan los cambios en nuestra base de datos sql.

Si te da error, prueba instalando estos paquetes (no los tendrías instalados, no pasa nada). Repetimos el proceso, en la consola dentro de la carpeta del proyecto:

“composer require symfonycasts/verify-email-bundle”

“php bin/console make:registration-form”

“composer require symfony/mailer”

“composer require Messenger”

Ya debería de haberte creado un controlador automáticamente, una tabla en tu base de datos “User” y un formulario de registro del que tendrás que cambiar a donde te lleva una vez hayas completado el registro

Para crear el login es mucho más simple, con “php bin/console make:controller Login” en la consola, y cambiando, al igual que en el registro, a donde te lleva y lo que pasa en la página una vez se haya completado el formulario de registro/login. Se te habrán creado las partes necesarias y la vista automáticamente .

Para más información: Security (Symfony Docs)

Recibir correo al registrarse

Symfony utiliza la librería Mailer para enviar correos. Está incluida en la instalación con la opción websiteskeleton. El servidor de correo se configura con la variable MAILER_URL que está en el fichero .env. Si no está instalada ejecuta “composer require symfony/mailer”

El componente Mailer incluye soporte para los siguientes servicios: Amazon SES, MailChimp, Mailgun, Gmail, Postmark y SendGrid. Cada uno se instala por separado. Por ejemplo, para usar el de Gmail: composer require symfony/google-mailer Al ejecutarlo cambia el fichero . env que deberás configurar con tu usuario y contraseña

Para enviar un correo habrá que conectar automáticamente con MaileInterface, especificar los componentes que vamos a usar y crear un objeto Email. Para que nos encuentre la configuración que hay en .env debemos instanciar el objeto mailer de la siguiente forma:

“$t = Transport::fromDsn($_ENV[‘MAILER_DSN’])” y “$mailer = new Mailer($t)”

Estas dos líneas las puedes poner justo antes de la llamada al método send() Por último, hay que usar la librería correspondiente:

“use Symfony\Component\Mailer\Transport”

Por último añadir los datos como, de dónde viene el correo, a donde va dirigido, el texto que contiene, etc…

En mi caso queda así dentro del formulario de registro:

```
$t = Transport::fromDsn($_ENV['MAILER_DSN']);
$mailer = new Mailer($t);

$email = (new TemplatedEmail())
->from(new Address('pepepruebas@gmail.com', 'Bot de correo de SpotyFake'))
    ->to($user->getEmail())
    ->subject('Por favor confirma tu correo')
    ->htmlTemplate('registration/confirmation_email.html.twig')
    ->text('¡Gracias por registrarte! Si te ha llegado este correo, es que tu registro en spotyFake ha sido correcto. Ya p
    ->context([
        'user' => $user,
    ]);

$mailer->send($email);
```

Reproducciones por cantante

Al finalizar el trabajo me he dado cuenta que es mejor guardar una variable con las reproducciones totales y guardarla al cantante para así poder usarlo en la base de datos. Mediante añadir el siguiente código a la entidad artista:

```php
public function getReproduccionesTotales(): int
{
    $reproduccionesTotales = 0;

    foreach ($this->canciones as $cancion) {
        $reproduccionesTotales += $cancion->getReproducciones();
    }

    return $reproduccionesTotales;
```

y en cantanteController añadir esto a la función show:

```php
#[Route('/{id}', name: 'app_cantante_show', methods: ['GET'])]
public function show(Cantante $cantante): Response
{
    $reproduccionesTotales = $cantante->getReproduccionesTotales();

    return $this->render('cantante/show.html.twig', [
        'cantante' => $cantante,
        'reproduccionesTotales' => $reproduccionesTotales
    ]);
}
```

Fin