**SYMFONY**

RESUMEN

27-Jul-22

**CREACIÓN DEL PRIMER PROYECTO**

**(Práctica 8)**

1 - Creación de un esqueleto para proyecto web tradicional (proyecto que crearemos):

composer create-project symfony/website-skeleton Proyecto1

2 - Desde el proyecto recién creado, lanzamos el servidor de pruebas para probar que está todo bien configurado.

php -S localhost:8000 -t public/

**CONTROLLER**

**(Práctica 9)**

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class PrimerController extends AbstractController

{

    #[Route('/ejer1', name: 'inicio')]

    public function inicio()

{

return $this->render('Deportes/Entrenador.html.twig', [

            'nombre' => $minombre,

            'apellido' => $miapellido

        ]);

    }

}

**ACCESO A VARIABLES DESDE TWIG**

**(Práctica 11)**

Los delimitadores en Twig:

Su sintaxis incluye 3 delimitadores:

Comentarios: {# ... #}

{# Esto es un comentario #}

Para mostrar contenido (echo): {{ ... }}

<td>{{ variable }}</td>

Para ejecutar lógica: {% ... %}

{% for fondo in fondos %}

...

{% endfor %}

**REALIZAR UNA COLECCIÓN EN UN DATALIST HTML5**

**(Práctica 13)**

Un control DataList tiene la siguiente sintaxis en HTML5:

<input type="text" name="txtAgenda" list="agenda" />

<datalist id="agenda">

<option value="Adrian"></option>

<option value="Ana"></option>

<option value="Lucia"></option>

</datalist>

**HERENCIA DE PLANTILLAS**

**(Prácticas 14 y 15)**

Plantilla base-padre (base.html.twig):

<!DOCTYPE html>

<html>

<head>

        {% block cabecera %}

            <link rel="stylesheet" href="css/styles.css"/>

            <title>{% block titulo %}{% endblock %}</title>

        {% endblock %}

</head>

<body>

{% block contenido %}

{% endblock %}

  {% block pie %}

  {% endblock %}

</body>

</html>

En la plantilla hija hay que incluir al principio:

{% extends "base.html.twig" %}

La función **INCLUDE** es útil para incluir una plantilla y devolver el contenido renderizado de esa plantilla a la actual.

{{ include('prueba.html.twig') }}

**PROCEDIMIENTOS ALMACENADOS**

**(Práctica 32)**

Hay que crear un procedimiento en PHPMyAdmin. Primero entraremos en la BD y luego buscamos Rutinas.

Sintaxis:

Create Procedure Procedimiento (IN/OUT nombreParametro Tipo de dato, IN/OUT...)

Instrucciones

Posteriormente lo ejecutamos introduciendo un valor/res correspondiente al parámetro/os introducido.

call nombreProcedure (valor1,valor2,valor3,…)

# VARIABLES DE SALIDA EN PROCEDIMIENTOS ALMACENADOS

Crear procedimiento almacenado con un parámetro de entrada y otro de salida. Pasaremos como parámetro de entrada un número de departamento y nos devolverá la localidad en la que está ubicado.

CREATE PROCEDURE DevolverLocalidad (OUT ploc VARCHAR(50), IN dp INT)

SELECT loc INTO ploc from

dept where dept\_no=dp;

CALL devolverLocalidad(@a,10);

SELECT @a as dato;

**CREAR BASE DE DATOS DESDE DOCTRINE ORM**

**(Práctica 35)**

1 - Configurar el acceso a la base de datos. Modificar el archivo .env para poder acceder a la base de datos MySQL instalada con nuestro paquete XAMPP:

DATABASE\_URL="mysql://root@127.0.0.1:3306/NOMBREBASEDATOS"

2 - Crear la base de datos. Podemos crear la base de datos a mano, o utilizar la consola de symfony:

php bin/console doctrine:database:create

3 - Crear las entidades. Desde la consola de symfony):

php bin/console make:entity

4 - Crear las tablas en la base de datos

php bin/console doctrine:schema:update --force

**GENERAR ENTIDADES CON LAS TABLAS**

**DESDE LA BASE DE DATOS**

**(Práctica 41)**

1 - Ejecutar el comando para que genere las entidades de las tablas en nuestra base de datos en el formato de anotación:

php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity

2 - Generar el repositorio con anotaciones

@ORM\Entity(repositoryClass="App\Repository\NOMBRERepository")

3 - Ejecutar el comando para que regenere las entidades:

En general, para todas las entidades:

php bin/console make:entity --regenerate App

Para una entidad específica:

php bin/console make:entity --regenerate "App\Entity\NOMBRE"

**GET / POST**

Si method=”POST” hay que usar:

$request->request->get(‘NOMBRE’);

Si method=”GET” hay que usar:

$request->query->get(‘NOMBRE’);

**RECUPERAR REGISTROS DE LA BASE DE DATOS**

**(Práctica 39)**

La clase Repository nos ofrece automáticamente varios métodos muy útiles para obtener registros de la base de datos:

find($valor) - Obtiene un registro a partir de la clave primaria de la tabla.

findOneByXXX ($valor) - Obtiene un registro pudiendo pasar como argumentos los valores que irían dentro del WHERE.

findByXXX($valor) - Obtiene los registros encontrados pudiendo pasar como argumentos los valores que irían dentro del WHERE. Retorna un array.

findAll() - Obtiene todos los registros de la tabla. Retorna un array.

findOneBy(array)

findBy(array)

Ejemplo Práctica 42: findBy(array('oficio'=>$oficio)).

Crea un array con todos los registros que tienen el mismo **oficio**.

**INSERTAR BOOTSTRAP**

**(Práctica 36)**

Para incluir Bootstrap hay que poner esta línea en el header:

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor" crossorigin="anonymous">

Para retocar los CSS con Bootstrap:

<label **class="form-label"**>

      <input type="text" **class="form-control"** id="txtLoc">

<button type="submit" **class="btn btn-primary"**>

**DAR DE ALTA UN REGISTRO EN BD (DOCTRINE)**

**(Práctica 36)**

  // 1) recibir datos del formulario

      $nombre = $request->request->get('txtNombre');

      $loc = $request->request->get('txtLoc');

  // 2) dar de alta en bbdd

      $departamento = new Dept();

      $departamento->setDnombre($nombre);

      $departamento->setLoc($loc );

      // Informamos a Doctrine de que queremos guardar el Grado (todavía no se ejecuta ninguna query)

$em->persist($departamento);

     // Para ejecutar las queries pendientes, se utiliza flush().

$em->flush();

// 3) redirigir al formulario. Coincide con el nombre de la ruta del método anterior: name: 'nuevoDepart’

return $this->redirectToRoute("nuevoDepart");

**BORRAR UN REGISTRO EN BD (DOCTRINE)**

**(Práctica 37)**

$identificador = $request->request->get('txtId');

$datos = $em->getRepository(Dept::class)->find($identificador);

if(!$datos) {

          return $this->render('noEncontrado.html.twig', [

              'mensaje' => 'Departamento no existe'

          ]);

      }

$em->remove($datos);

$em->flush();

return $this->render('ok.html.twig', [

          'mensaje' => 'Dato eliminado correctamente:'. $identificador

      ]);

**MODIFICAR UN REGISTRO EN BD (DOCTRINE)**

**(Práctica 38)**

$identificador = $request->request->get('txtId');

$nombre = $request->request->get('txtNombre');

$loc = $request->request->get('txtLoc');

$departamento = $em->getRepository(Dept::class)->find($identificador);

$departamento->setDnombre($nombre);

$departamento->setLoc($loc );

$em->persist($departamento);

// Para ejecutar las queries pendientes, se utiliza flush().

$em->flush();

return $this->redirectToRoute("modDepart");

**WEB CON MÉTODO GET**

**(Práctica 44)**

Enviamos datos en un hipervínculo de la siguiente manera:

 <a href="{{ path('ver\_datos', **{id: 1}**) }}">SEVILLA</a>

Los recogeremos en otro archivo con la función GET:

 $datoget = $request->query->get('**id**');

**VOLVER ATRÁS CON UN BOTÓN**

**(Práctica 44)**

<a href="javascript:history.back()">VOLVER</a>

**INSERTAR CSS**

Abrir en **Public** una carpeta llamada **css** y dentro de esta abrir un archivo llamado **styles.css**.

Incluir en el archivo de inicio un enlace a los estilos CSS:

<link rel="stylesheet" type="text/css" href="/css/styles.css" />

**LENGUAJE DQL**

Se puede utilizar DQL para consultas de tipo SELECT, UPDATE y DELETE.

DQL no permite hacer INSERTS. Los inserts se deben realizar mediante el método persist() el EntityManager.

//Cuando trabajamos con DQL, no trabajamos con tablas y campos de las tablas sino con entidades y con propiedades de las entidades.

// App\Entity\Dept: Namespace del modelo.

// Incluimos la ruta completa de la clase Dept.php predeciendolo de su namespace "namespace App\Entity";

Prepara la búsqueda:

$query = $em->createQuery('SELECT u FROM App\Entity\Dept **u** WHERE u.id > 1'); (ese **u** es el alias de la entidad)

Ejecuta la búsqueda:

$datos = $query->getResult();

Devuelve el resultado de la búsqueda:

return $this->render('verDatos.html.twig', [

     'datosDept' => $datos,

   ]);

**DQL CON PARÁMETROS**

**(Práctica 50)**

PARÁMETROS

Se pueden indicar parámetros con el símbolo '?' o el símbolo ':'. El símbolo ‘:’ se usa más.

Ejemplo: $query = $em->createQuery('SELECT u FROM App\Entity\Dept u WHERE u.id = :dato');

Para poner el alias: u.oficio AS oficio

$datoget = $request->query->get('id');

$query = $em->createQuery('SELECT u FROM App\Entity\Dept u WHERE u.id = **:dato**');

Primero creamos el párametro dato y luego le asignaremos un valor:

$query->setParameter('**dato**',  $datoget);

$datos = $query->getResult();

 return $this->render('verDatos.html.twig', [

     'datosDept' => $datos,

]);

**PAGINAR EMPLEADOS POR NUMERO REGISTROS**

**(Práctica 53)**

Instalación biblioteca para poder paginar:

El primer paso será instalar la biblioteca con Composer:

**composer require knplabs/knp-paginator-bundle**

Incluir una función en el controlador para poder paginar registros.

**use Knp\Component\Pager\PaginatorInterface**

Poner esto dentro del controlador:

public function paginar(Request $request, PaginatorInterface $paginator,EntityManagerInterface $em)

{

   $query = $em->getRepository(Emp::class)->findAll();

// Paginar los resultados de la consulta

$datosPaginados = $paginator->paginate(

            // Consulta sin ejecutar

            $query,

            // Definir el parámetro de la página recogida por GET

            $request->query->getInt('page', 1),

            // Número de elementos por página

            5

        );

 return $this->render('Index.html.twig', [

            'datosP' => $datosPaginados

        ]);

}

Para colocar los índices y las flechas abajo del todo, poner esto en la página HTML, debajo de la tabla de datos:

**{{ knp\_pagination\_render(datosP) }}**

**EJECUTANDO PROCEDIMIENTOS ALMACENADOS**

**(Práctica 54)**

  public function eliminarDepart(Request $request, EntityManagerInterface $em)

  {

    $id = $request->request->get('txtId');

 $connection = $em->getConnection();

  $statement = $connection->prepare("CALL borrarDept(:dato)");

  $statement->bindValue('dato', $id);

  $resultado= $statement->executeStatement();

   return $this->render('borrado.html.twig', [

     'afectados' => $resultado,

   ]);

   }

}

**CARGAR VISTAS PARCIALES CON JQUERY AJAX**

**(Práctica 55 - Un poco complejo: ver la práctica)**

El fichero **jquery-3.2.1.min.js** tiene que estar incluido en la sección **public**. Podemos descargar la última versión de JQuery. Para ello, debemos entrar en la siguiente dirección:

<http://jquery.com/>

Y pulsamos sobre la opción de Download en la página:

Meter esta línea en el index.html:

<script src="/jquery-3.2.1.min.js" type="text/javascript"></script>

Lo que haremos será un código de JQuery que cargará en los <div> cada vista al pulsar cada botón con la siguiente sintaxis:

Texto

Descripción generada automáticamente

**GENERAR PDF**

**(Práctica 59)**

El primer paso será instalar la biblioteca Dompd (biblioteca para generar PDF) con Composer:

**composer require dompdf/dompdf**

Introducir estar línea en el controlador:

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

Meter esto en el controlador:

#[Route('/Index', name: 'inicio')]

    public function inicio()

    {

          // Existen muchas configuraciones para Dompdf. Incluimos una de las muchas que tiene

          // por ejemplo asignar el tipo de letra

          $pdfOptions = new Options();

          $pdfOptions->set('defaultFont', 'Arial');

          // Crea una instancia de Dompdf con nuestras opciones

          $dompdf = new Dompdf($pdfOptions);

          // Preparamos la página HTML para generar PDF

          $html = $this->renderView('figurasPDF.html.twig', [

              'autor' => "Benito Floro"

          ]);

          // Ahora se carga la página HTML en Dompdf

          $dompdf->loadHtml($html);

          // También podemos de forma opcional indicar el tamaño del papel y la orientación

          $dompdf->setPaper('A4', 'portrait');

          // Renderiza el HTML como PDF

          $dompdf->render();

          // Envíe el PDF generado al navegador. ¡DESCARGA FORZADA!

          $dompdf->stream("figuras.pdf", [

              "Attachment" => true

          ]);

}

"Attachment" => true manda un fichero adjunto

"Attachment" => false genera el pdf y permite visualizarlo

**VARIABLES SESSION**

**(Práctica 61)**

Crear dos funciones en el controlador

-Podemos guardar un dato en la sesión para utilizarlo en una petición posterior con la función set después de obtener el identificador de la sesión de la petición realizada.

-Recuperaremos la información de la sesión con la función get.

  #[Route('/inicio', name: 'verPagina1')]

  public function verPagina(Request $request)

    {

        $session = $request->getSession();

        $session->set('nombre', 'Benito');

        $session->set('apellido', 'Floro');

        $session->set('edad', 22);

        $session->set('altura', 1.85);

        return $this->render('Inicio.html.twig');

     }

     #[Route('/pagina2', name: 'verPagina2')]

     public function verPagina2(Request $request)

       {

        $minombre = $request->getSession()->get('nombre');

        $miapellido = $request->getSession()->get('apellido');

        return $this->render('verDatos.html.twig', [

            'nombrevariable' => $minombre,

            'apellidovariable' =>  $miapellido,

        ]);

        }

También podremos realizar otras acciones:

// Preguntar si existe un dato en la sesión

$existe = $session->has('cesta');

// Usar un valor por defecto si no existe

$filtro = $session->get('cesta', []);

//Destruye la sesión y crea otra nueva

$session->invalidate();

//Borra todos los atributos de la sesión actual

$session->clear();

**INTERNACIONALIZACION**

**(Práctica 64)**

**Ir a config > translation.yaml**

Dentro de la carpeta translations creamos ficheros llamados **messages** con las traducciones a los diferentes idiomas:

**messages.en.yaml**

Incluimos una ruta precedida del locale en una función del controlador:

**'/{\_locale}/inicio'**

Mostramos las traducciones en nuestro template:

Las traducciones de texto en twig se realizan a través de las etiquetas **trans**.

**{% trans %} mensaje {% endtrans %}**

**PASAR VARIABLES DENTRO DE UN CHECKBOX**

**(Práctica 46)**

En un formulario de html incluimos los datos de varios checkbox:

<input type="checkbox" name="colores[]" value="Azul" />

Los recogeremos en el controlador:

$arr = $request->request->all('colores');