Tasks manager

[Capte la atención del lector con una descripción breve atractiva. Esta descripción es un resumen breve del documento. Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

SYMFONY 5.1

Contenido

[1. INSTALACION DE SYMFONY & CREACION DEL PROYECTO. 2](#_Toc58522331)

[2. conexión BASE DE DATOS. 2](#_Toc58522332)

[2.1 SCRIPT BASE DE DATOS. 2](#_Toc58522333)

[2.2 CONFIGURACION BD FICHERO .env . 3](#_Toc58522334)

[3.MODELO Y ENTIDADES. 3](#_Toc58522335)

[3.1 GENERAR ENTIDADES. 3](#_Toc58522336)

[3.2 GENERAR SETTERS & GETTERS. 3](#_Toc58522337)

[3.3 GENERAR CONTROLADORES. 3](#_Toc58522338)

[3.4 RELACIONES ORM. 4](#_Toc58522339)

[3.4 TESTEO ORM. 6](#_Toc58522340)

[4. REGISTRO DE USUARIOS. 7](#_Toc58522341)

[4.1 CREAR FORMULARIO. 7](#_Toc58522342)

[4.2 GUARDAR DATOS DEL FORMULARIO DE REGISTRO. 10](#_Toc58522343)

[4.2.1 ENCRIPTAR PASSWORD. ENCODERS. 10](#_Toc58522344)

[4.2.2 FECHAS. 12](#_Toc58522345)

[4.3 VALIDACION DEL FORMULARIO DE REGISTRO. 13](#_Toc58522346)

[4.3.1 VALIDACIONES EN FORMATO ANNOTATION. 14](#_Toc58522347)

[4.4 CARGAR ESTILOS PERSONALIZADOS. 15](#_Toc58522348)

[4.5 CARGAR ESTILOS BOOTSTRAP 4.(ACABAR) 15](#_Toc58522349)

[4.6 ESTILOS DEL FORMULARIO. 16](#_Toc58522350)

[4.7 GUARDAR DATOS. 16](#_Toc58522351)

[4.8 LOGIN DE USUARIO. 17](#_Toc58522352)

[4.8.1 CONFIGURACION LOGIN. 17](#_Toc58522353)

[4.9 OBTENER USUARIO IDENTIFICADO. 18](#_Toc58522354)

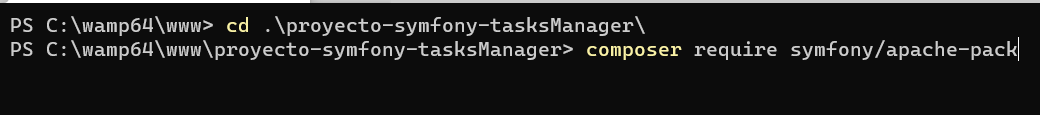
[4.10 MODIFICACION MENU EN FUNCION DE SI ESTAMOS LOGGED. 19](#_Toc58522355)

[4.11 MESSAGE FLASH. 19](#_Toc58522356)

# 1. INSTALACION DE SYMFONY & CREACION DEL PROYECTO.

Vamos a crear el proyecto y a instalar el apache pack de composer para poder visualizar correctamente la barra de degug.





# 2. conexión BASE DE DATOS.

## 2.1 SCRIPT BASE DE DATOS.

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

CREATE DATABASE IF NOT EXISTS taskmanager\_master;

USE taskmanager\_master;

DROP TABLE IF EXISTS users;

CREATE TABLE IF NOT EXISTS users

(

    id          *INT* *AUTO\_INCREMENT* NOT NULL,

    name        *VARCHAR*(100) NOT NULL,

    surname     *VARCHAR*(255),

    email       *VARCHAR*(255) NOT NULL,

    password    *VARCHAR*(255) NOT NULL,

    role        *VARCHAR*(20),

    imagen      *VARCHAR*(255),

    created\_at  DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    updated\_at  DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    CONSTRAINT pk\_usuarios PRIMARY KEY(id),

    CONSTRAINT uq\_email UNIQUE(email)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

DROP TABLE IF EXISTS tasks;

CREATE TABLE IF NOT EXISTS tasks

(

    id          *INT* *AUTO\_INCREMENT* NOT NULL,

    user\_id     *INT* NOT NULL,

    title       *VARCHAR*(255) NOT NULL,

    content     *TEXT* NOT NULL,

    hours       *INT* NOT NULL,

    priority    *VARCHAR*(20),

    created\_at  DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    updated\_at  DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

     CONSTRAINT pk\_tasks PRIMARY KEY(id),

    CONSTRAINT fk\_tasks\_users FOREIGN KEY (user\_id)REFERENCES  users(id)

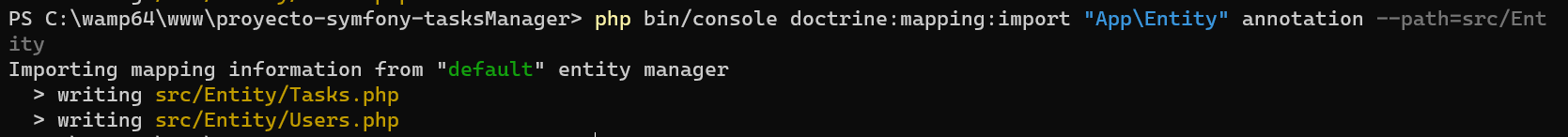
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_general\_ci;

## 2.2 CONFIGURACION BD FICHERO .env .



# 3.MODELO Y ENTIDADES.

## 3.1 GENERAR ENTIDADES.



Modificamos a singular los nombres de las entidades

## 3.2 GENERAR SETTERS & GETTERS.



Modificamos a singular los nombres de los métodos que reciban users en vez de user

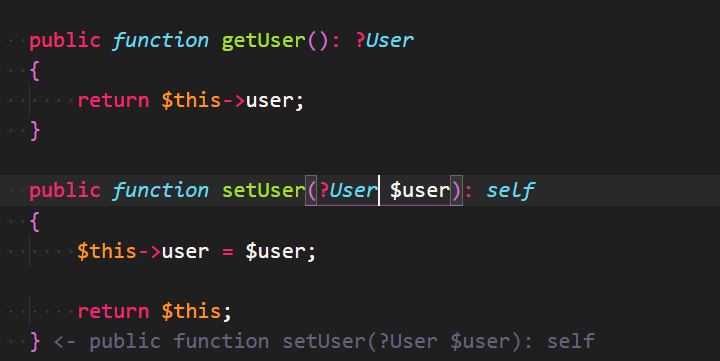
## 3.3 GENERAR CONTROLADORES.



## 3.4 RELACIONES ORM.

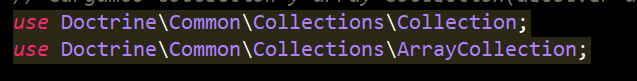
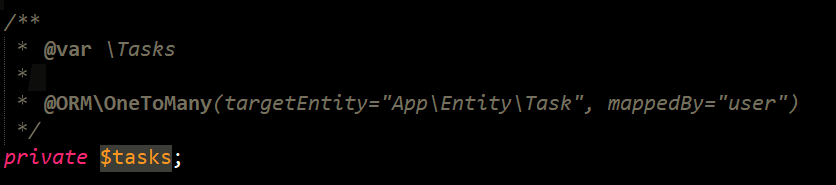
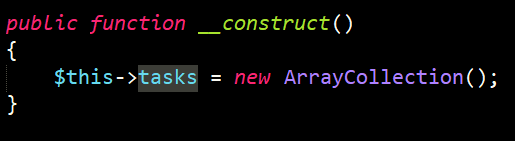
Automaticamente symfony nos ha generado en la clase Task , 2 metodos para obtener el usuario que creo la tarea :

**src/Entity/Task.php**



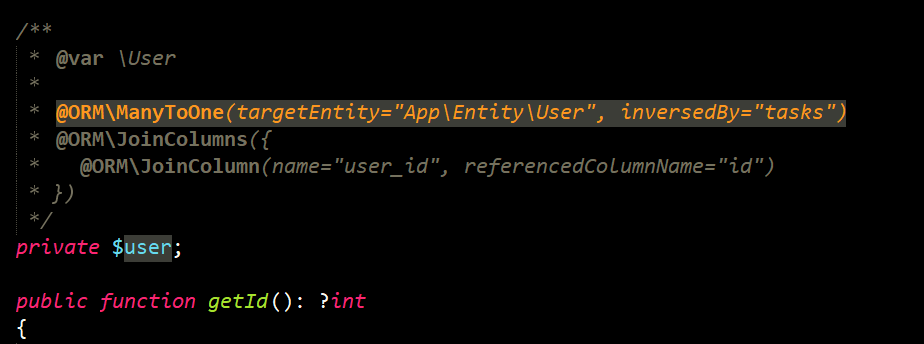
**src/Entity/User.php**

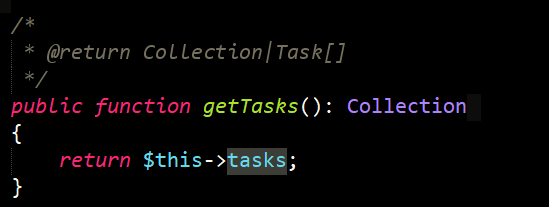
Ahora necesitamos en la clase User implementar métodos que nos permitan obtener todas las tareas que ha creado un usuario, ***esta tendrá una relación OneToMany para obtener el listado de tareas que ha creado un usuario***.

1. Cargar librería de symfony que nos permite devolver como resultado un array Collection(array de colección de objetos). 
2. Crear variable **$tasks** para trabajar con el array collection, creamos la relación OneToMany y modificamos el target con la ruta entera **targetEntity="App\Entity\Task", mappedBy="user"**, y estará relacionada o mapeada a la propiedad user de la entidad Task. 
3. Crear constructor generando una nueva instancia de ArrayCollection. 
4. En la propiedad user de la entidad Task hemos de modificar el **targetEntity** con la ruta completa y ademas con el inversedBy relacionamos de manera bidireccional user con tasks:

mappedBy(en User | $tasks)🡪 RELACION BIDIRECCIONAL

inversedBy(en Tasks | $user)🡨



1. Crear método en User para obtener las tareas de un usuario 

## 3.4 TESTEO ORM.

*class* TaskController extends *AbstractController* {

  public *function* index(): *Response* {

      //\* \*\*\*\*\*\*\*\*\* TEST ORM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

      $doctrine = $this->getDoctrine();

      $entity\_manager = $doctrine->getManager();

      // Importar el repositorio

      $task\_repository = $doctrine->getRepository(*Task*::class);

      //\* OBTENER TODAS LAS TAREAS

      $tasks = $task\_repository->findAll();

      foreach ($tasks as $task) {

        echo "Titulo: " . $task->getTitle() . "<br>";

        echo "Name: " . $task->getUser()->getName() . "<br>";

        echo "Email: " . $task->getUser()->getEmail() . "<br>";

   //echo "Created at: ".$task->getUser()->getCreatedAt();

        echo "<hr>";

       }

      //\* OBTENER TODAS LAS TAREAS DE UN USUARIO

      $user\_repository = $doctrine->getRepository(*User*::class);

      $users = $user\_repository->findAll();

      foreach ($users as $user) {

      echo "<h1>{$user->getName()} {$user->getSurname()}</h1>";

          foreach ($user->getTasks() as $task) {

              echo $task->getTitle() . "<br/>";

          }

      }

      return $this->render('task/index.html.twig', [

          'controller\_name' => 'TaskController',

          //'tasks' => $tasks,

      ]);

  }

}

# 4. REGISTRO DE USUARIOS.

## 4.1 CREAR FORMULARIO.

Creamos un directorio en src 🡪 Form/Type y creamos el archivo RegisterType.php

**RegisterType.php**

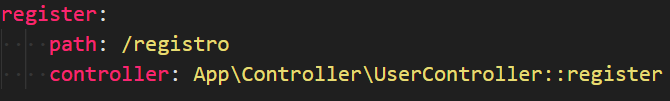


Vamos a generar el formulario en el controlador y renderizarlo en la vista

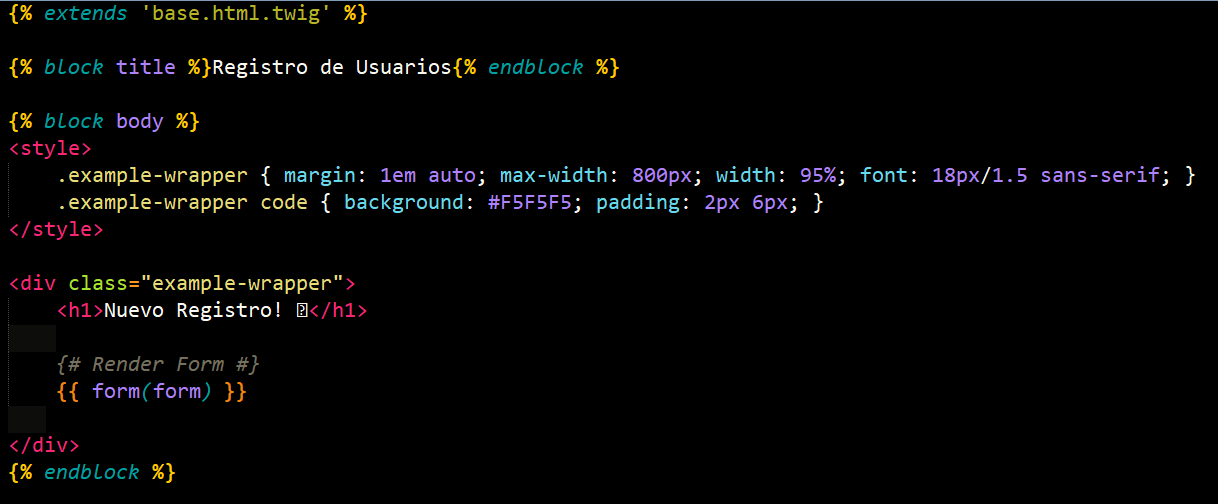
**UserController.php**



Ruta



**Vista 🡪 register.html.twig**



## 4.2 GUARDAR DATOS DEL FORMULARIO DE REGISTRO.

### 4.2.1 ENCRIPTAR PASSWORD. ENCODERS.

En primer lugar vamos a implementar el encoder para encriptar los passwords, para ello vamos a **config/packages/security.yaml**



En el controlador UserController llamamos a :



Y se lo pasamos como argumento al método además de Request, un parámetro de tipo UserPasswordEncoderInterface, y su variable.



Veamos el método al completo:



### 4.2.2 FECHAS.

En cuanto a las fechas he tenido problemas a la hora de guardar la fecha, para solucionarlo hemos modificado los setters en la entidad User eliminado el Tipado que tenia :

public *function* getCreatedAt()

    {

        return $this->createdAt;

    }

    public *function* setCreatedAt(): *self*

    {

        //$this->createdAt = $createdAt; // ORIGINAL

        $this->createdAt = new *\DateTime*("now");

        return $this;

    }

    public *function* getUpdatedAt()

    {

        return $this->updatedAt;

    }

    public *function* setUpdatedAt(): *self*

    {

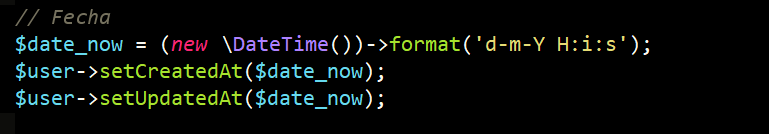
        //$this->updatedAt = $updatedAt; // ORIGINAL

        $this->updatedAt = new *\DateTime*("now");

        return $this;

    }

En el controlador:

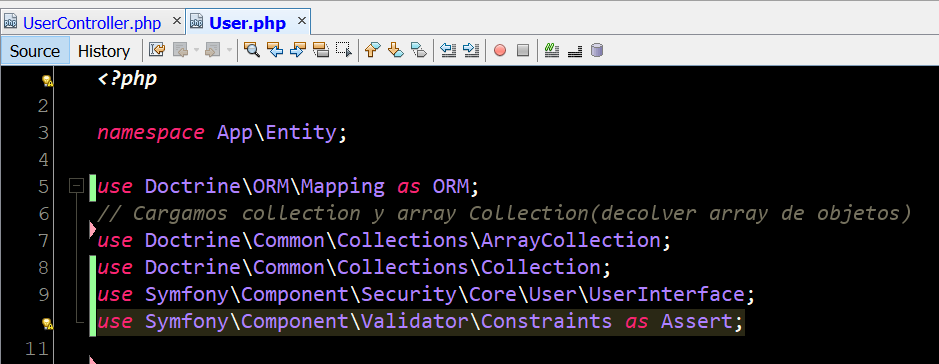


## 4.3 VALIDACION DEL FORMULARIO DE REGISTRO.

Vamos validar los campos del formulario. Añadimos en el if(), donde validamos si se ha enviado el form el método **isValid()**



Ahora en la entidad User hemos de añadir las restricciones



Ahora podemos añadir las restricciones en formato annotation:

### 4.3.1 VALIDACIONES EN FORMATO ANNOTATION.

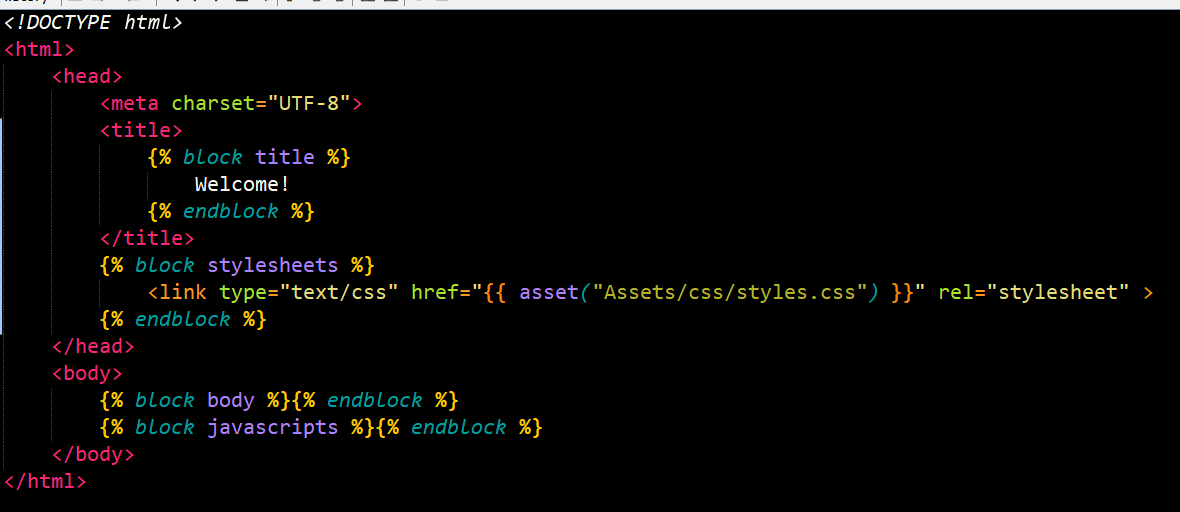


Al heredar de UserInterface hemos de redefinir los métodos de esta Interface en la entidad User:



## 4.4 CARGAR ESTILOS PERSONALIZADOS.

Para cargar los estilos nos vamos a la plantilla base e insertamos la ruta a la hoja de estilos.



El fichero de estilos estará en **public/Assets/css/styles.css**

## 4.5 CARGAR ESTILOS BOOTSTRAP 4.(ACABAR)

## 4.6 ESTILOS DEL FORMULARIO.

## 4.7 GUARDAR DATOS.

Una vez implementados los métodos de la interface UserInterface, añadidos los encoders para la password y las fechas podemos almacenar el objeto user



## 4.8 LOGIN DE USUARIO.

Para el login Symfony dispone de una librería, solamente nos encargaremos de la autenticación y todo lo demás lo hará el framework.

MAS SENCILLO USAR EL CMD

**php bin/console make:auth**

Para que funcione hemos de heredar de UserInterface y añadir los métodos como hicimos en el registro(Ya hecho).

### 4.8.1 CONFIGURACION LOGIN.

Comprobar: En el fichero config/packages/security.yaml.

provider🡪Un provider es digamos donde va a buscar los datos para realizar el login. Para ello añadimos un **provider** y varias clausulas en **main:**



Solucion Victor:



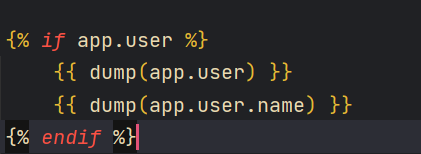
Tanto el formulario como el controller se genera automáticamente solo hemos de revisar las rutas si son las adecuadas.

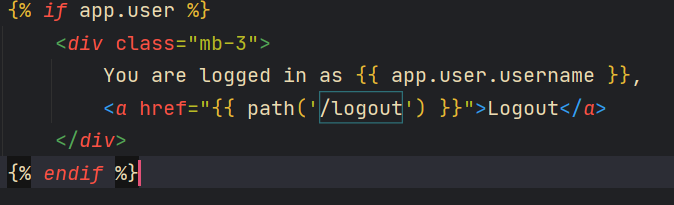
REVISAR RUTAS

## 4.9 OBTENER USUARIO IDENTIFICADO.

Este objeto estará disponible en toda la aplicación.

Usaremos el objeto **app.user**





## 4.10 MODIFICACION MENU EN FUNCION DE SI ESTAMOS LOGGED.



## 4.11 MESSAGE FLASH.

