

Roberto Gómez Domínguez



TigerBet

Tu casa de apuestas





A mi familia y amigos y a mis compañeros  
y profesores de DAW.

### Licencia

Esta obra está bajo una licencia Reconocimiento-Compartir bajo la misma licencia 3.0 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.



# INTRODUCCIÓN

---

Este documento responde a la realización del módulo de Proyecto del CFGS en Desarrollo de Aplicaciones Web.

El propósito general consiste en la creación de una aplicación, TIGERBET, que contendrá un blog de noticias relevantes en el mundo del deporte, así como un apartado para realizar apuestas online mediante un sistema de registro y autenticación de usuarios.

La aplicación está realizada utilizando el **framework Symfony** basado en un sistema Modelo Vista Controlador en PHP. Además, se utilizará JS en el lado cliente para la realización de los juegos. Todo esto sobre un esqueleto de HTML5 enriquecido con el motor de plantillas TWIG. Los estilos de la web corren a cargo del lenguaje CSS3.

Prácticamente se incluyen muchas de las tecnologías y conocimientos adquiridos a lo largo del curso: programación general, desarrollo en entorno cliente y servidor, creación y gestión de sistemas de bases de datos y administración de redes y servidores para la implementación de ciertos módulos con el propósito de llegar al producto final.

Este proyecto también incluye muchas horas de investigación, muchos problemas con muchas soluciones, pero más que nada, mucho esfuerzo y mucha ilusión.

En líneas generales, el proyecto se ha dividido en varias fases bien diferenciadas:

- Fase de análisis: donde se analizarán distintas webs reales que presentan contenido como el que se quiere ofrecer.
- Fase de diseño: donde se intentará describir con más detalle cómo se ha planificado y elaborado el desarrollo de la aplicación.
- Fase de implementación y pruebas: se explicará el proceso que se ha seguido para la creación de la aplicación web, así como de la fase de pruebas o "testeo".

## NECESIDADES DEL SECTOR PRODUCTIVO

Primeramente, vamos a identificar las necesidades detectadas en el sector productivo que originan la oportunidad de negocio que se detallará en los siguientes puntos.

The screenshot shows a news article titled "Seguimos para bingo... los juegos 'online' suben su apuesta y crecen casi un 30%" from the 'economía' section of 'LA INFORMACIÓN'. The sub-headline reads: "Las modalidades tradicionales, como el bingo o la quiniela, recaudan hoy la mitad que hace un decenio y la recuperación económica no mejora la cifra." The date is listed as Domingo, 21 de octubre de 2018.

### Cómo ha transformado Internet las apuestas y las loterías

Jueves 18 de octubre de 2018, 15:46h

0: 02:55



Para rastrear los orígenes de las apuestas debe retrocederse varios siglos en el pasado.

Como es natural, en la actualidad esta actividad se aleja mucho de lo que fue en sus orígenes, pero aún así se mantiene vigente aquello que más la caracteriza; la emoción y adrenalina que otorga la **posibilidad de ganar**. Con el auge de la tecnología, las **casas de apuestas** se han mudado a **plataformas online** que son capaces de ofrecer una experiencia más satisfactoria para los apostadores.

The screenshot shows an article from 'infoLibre' titled "El negocio de las apuestas 'online' se multiplica por seis desde 2012". The sub-headline is "EL AUGE DE LOS JUEGOS DE AZAR". The date is Domingo, 21 de octubre de 2018. The article discusses the growth of online gambling in Spain, mentioning the triple increase in spending since 2012, the lack of regulation, and the closure of new betting houses in Murcia and Baleares.

<https://www.albaceteaberto.es/noticia/33856/sociedad/como-ha-transformado-internet-las-apuestas-y-las-loterias.html>

[https://www.infolibre.es/noticias/economia/2018/10/01/el\\_negocio\\_las\\_apuestas\\_online\\_multiplica\\_por\\_seis\\_desde\\_2012\\_87229\\_1011.html](https://www.infolibre.es/noticias/economia/2018/10/01/el_negocio_las_apuestas_online_multiplica_por_seis_desde_2012_87229_1011.html)

## ANÁLISIS DE LA SITUACIÓN ACTUAL

A grandes rasgos podemos decir que, aunque el mercado deportivo sigue creciendo, ya ha alcanzado cierta madurez y, mientras sus ventas comienzan a ralentizarse, las apuestas deportivas están en su gran momento. El mundo de las apuestas trasciende lo deportivo.

Curiosamente, vemos que se va produciendo, además, un cambio de tendencias en la forma de consumo tradicional: las apuestas on-line.

Gracias en parte al aumento de la velocidad de las redes domésticas, así como el rendimiento de las líneas, éste mercado on-line va ganando terreno a pasos agigantados.

La normalización de plataformas como Google Play o Apple Store han hecho que este sector sea el que ocupe gran terreno en la web. Queda claro que el mundo online está cada vez más en nuestro día a día.

Este nuevo mercado y aplicaciones como BET365, Sportium, Betway, etc ... se han ganado un puesto en el terreno de las apuestas on-line.

## OPORTUNIDAD DE NEGOCIO

El sector de las apuestas tiene un público fiel y una muy buena acogida. Muchas personas que apuestan buscan disfrutar de experiencias, riesgo y una forma de entretenimiento más allá del deporte... En TIGERBET se pretenden cubrir todos estos aspectos, porque al fin y al cabo todo son experiencias. Esas son las necesidades que se quieren cubrir con esta aplicación web.

Siguiendo por esa línea, en un futuro se pueden planificar distintas acciones.

Desde TIGERBET apostamos también por el consumo responsable.

# FASES DEL PROYECTO

---

## FASE DE ANÁLISIS

En esta fase se determinarán los requisitos del proyecto, tanto lo que la aplicación tiene que hacer como las propiedades y cualidades que se deben cumplir. Se distinguirán entre requisitos funcionales y no funcionales.

### REQUISITOS FUNCIONALES

La aplicación deberá mantenerse actualizada, con nuevos contenidos, noticias y juegos.

Deberá poder visualizarse en todos los dispositivos Android, no sólo en teléfonos, sino también en tablets de diferentes resoluciones, usaremos para ello un diseño responsive con CSS3 puro o mediante algún framework como Bootstrap.

### REQUISITOS NO FUNCIONALES

La app deberá ser vistosa, con colores llamativos, para atraer la atención visual de nuestros usuarios y tener una temática acorde. Podrá contar con animaciones, música, menús de carga... esto hará que nuestra app sea algo más que una simple pantalla.

Además, se debe facilitar la usabilidad, de forma que el usuario sea capaz de usar la aplicación de forma intuitiva.

La aplicación deberá funcionar perfectamente, de forma fluida, sin cortes. No deberían tener tiempos de carga excesivamente largos y deberá mantener unos niveles mínimos de seguridad.

## FASE DE DISEÑO

La web va a ser la cara visible del proyecto, es importante tener claro su diseño y cómo será su estructura básica, así tendremos una visión general e igual de clara a la hora de ponernos a desarrollar.

El diseño y programación se ven lastrados a medida que avanza el proyecto, por lo que es importante dedicar tiempo antes de ponerse a "picar código".

Se ha diseñado un logo específicamente para esta aplicación y los colores seleccionados harán que la web mantenga cierta homogeneidad.



En cuanto al diseño de la app, tendremos una serie de páginas. Las secciones planificadas son:

- **INICIO:** Página inicial ( index ).
- **DEPORTES:** Apuestas deportivas.
- **CASINO:** Apuestas en juegos de azar.
- **NOTICIAS:** Vinculadas al deporte, tecnología, curiosidades...
- **CONTACTO**
- **LOGIN / MI CUENTA**

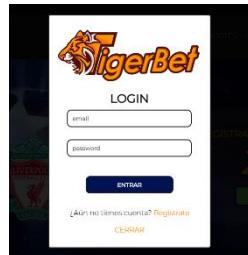
Con todo esto tenemos una pauta con la que comenzar a desarrollar nuestra maqueta con Photoshop para darnos una idea general de lo que será nuestra aplicación.

Nuestra página de inicio tendrá un menú de navegación con las secciones anteriormente indicadas: Inicio, Noticias, Casino, Deportes, Contacto y Login.

Cualquier usuario podrá ver las noticias de nuestro blog.



Sólo los usuarios registrados podrán realizar apuestas online y para ello, crearemos un sistema de login/registro de usuarios que se almacenarán en base de datos.



Tendremos una sección para consultar los resultados de la última jornada de la liga NBA y, por último, tendremos una sección de contacto para que los usuarios puedan ponerse en contacto con los administradores de TIGERBET en caso de tener dudas u ocurrir alguna incidencia con sus apuestas.

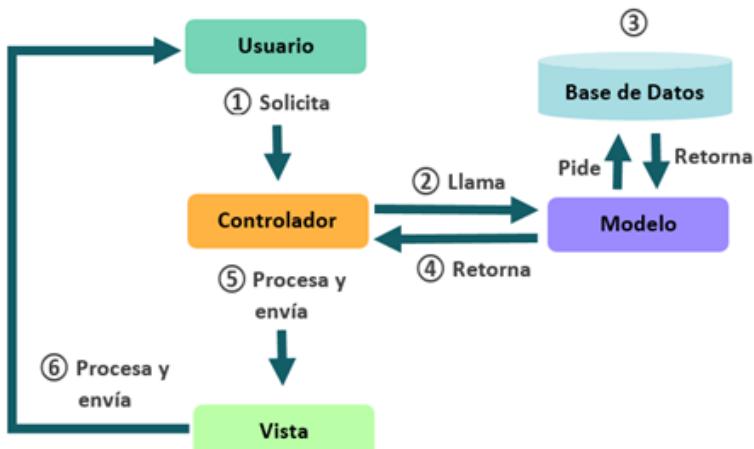
## FASE DE PLANIFICACIÓN

### SYMFONY Y LA ARQUITECTURA MVC

Para este proyecto usaremos Symfony, un proyecto PHP de software libre que permite crear aplicaciones y sitios web rápidos y seguros de forma profesional. Symfony está basado en un patrón de arquitectura MVC (modelo–vista–controlador) que separa la lógica de negocio de la presentación.

La utilización de un patrón MVC supone crear aplicaciones robustas, donde se facilita el mantenimiento y la reutilización de código.

El flujo de un MVC es sencillo. Cuando un usuario requiere una web, se llama a un Controlador. El Controlador se comunica con el Modelo (que recoge los datos de la base de datos). El Modelo entrega los datos, de nuevo, al Controlador, y éste nos sirve una Vista que se renderiza en el navegador, mostrando así la página que hemos solicitado.



En el siguiente enlace podemos ampliar información sobre Symfony y sus ventajas:  
<https://www.diligent.es/framework-symfony-php/>

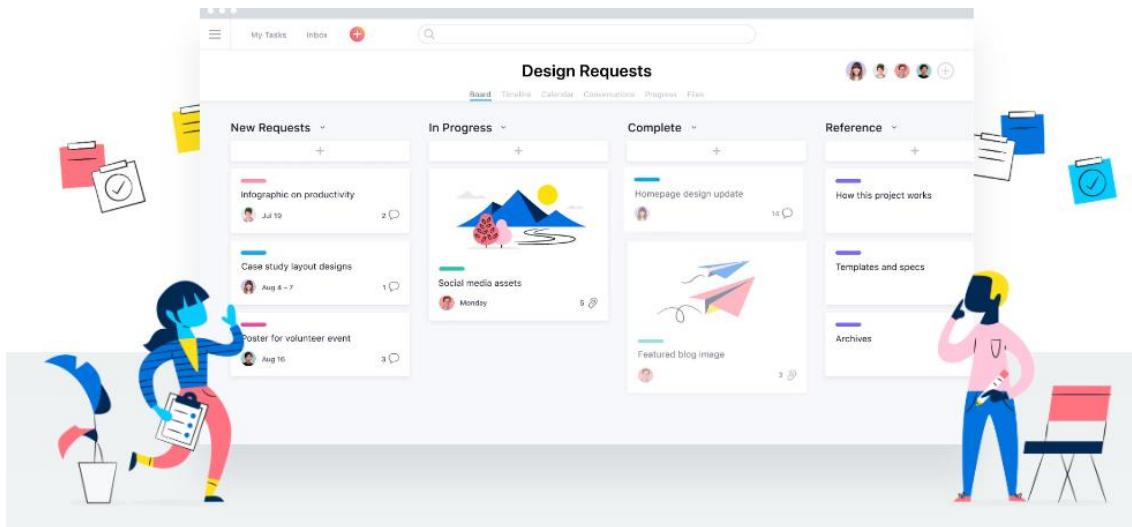
## METODOLOGÍA DE TRABAJO

¿Qué es SCRUM?

SCRUM es un proceso donde se aplican buenas prácticas para el trabajo en equipo y así obtener los mejores resultados posibles dentro de un proyecto.

Aunque este proyecto es personal, utilizaremos una metodología “agile” para concretar plazos de entrega, definir requisitos del proyecto y tener una mejor visión global del proyecto.

Utilizaremos la herramienta online ASANA (<https://asana.com/es>).



Nos registramos en la aplicación y estableceremos plazos para el desarrollo de nuestro proyecto. De esta manera tendremos controlados los tiempos de entrega.

Recordamos que, en un proyecto real, los tiempos son muy importantes puesto que una demora en la entrega supone costes adicionales en el proyecto. Intentaremos por todos los medios cumplir los plazos establecidos. El diagrama de flujo de trabajo quedaría de ésta manera:



Asana es una herramienta muy cómoda, ya que podemos ver de una sola pasada el progreso de nuestro proyecto y una herramienta muy completa cuando trabajamos en equipo.

A screenshot of the Asana project board titled "TigerBet". The board is organized into six columns: DEFINICIÓN, PROTOTIPO, NAVEGACIÓN COMPL3TA, ACCESO A DATOS Y AUTENTICACI..., INTEGRACIÓN DE SERVICIOS, and PRUEBAS. Each column contains several tasks represented by cards with icons and descriptions. For example, the "DEFINICIÓN" column has tasks like "Diseño de pantallas (boceto)" and "Especificación detallada de la funcionalidad". The "PRUEBAS" column includes "Testing final" and "Finalización de docume". The sidebar on the left shows the project navigation and some user information.

## FASE DE IMPLANTACIÓN

### INSTALACIÓN DE COMPOSER

Una vez que tenemos clara la tecnología con la que vamos a trabajar, podemos pasar a instalar Symfony. Actualmente, al igual que otros frameworks, se instala mediante **Composer**. (<https://getcomposer.org/> ).

Composer es un gestor de dependencias para proyectos PHP que permite descargar paquetes desde un repositorio para incluirlos en nuestro proyecto.

La instalación de Composer es sencilla. Podemos instalar mediante la consola de comandos, pero también existe un ejecutable con interfaz gráfica guiada para Windows 32 y 64bits.



Verificamos que tenemos Composer instalado correctamente a través del cmd.

Ejecutamos el comando: **composer --version**.

A screenshot of a Windows Command Prompt window. The title bar says "Símbolo del sistema". The command "C:\>composer --version" is typed, and the output "Composer version 1.7.2 2018-08-16 16:57:12" is displayed. The prompt "C:\>" appears again at the bottom.

## INSTALACIÓN DEL SERVIDOR WAMP

Para poder desplegar nuestra aplicación, necesitaremos un servidor. En nuestro caso instalaremos WAMP. Accedemos a la página oficial y descargamos el ejecutable. Simplemente seguimos los pasos que se nos indican en la interfaz de usuario. (<http://www.wampserver.com/en/>)



Si no queremos instalar **XAMP / WAMP**, tenemos la posibilidad de desplegar nuestro proyecto en local ejecutando el comando: **php bin/console server:run**. De esta forma tendremos nuestra aplicación disponible por el puerto **8000**.

```
eg: C:\wamp64\www\tigerbet>php bin/console server:run
[30;42m [39;49m
[30;42m [OK] Server running on http://127.0.0.1:8000
[39;49m
[30;42m [39;49m
[39;49m // [39;49mQuit the server with CONTROL-C.
```

## NUESTRO IDE FAVORITO

Para éste proyecto, necesitaremos un editor de texto avanzado como **Sublime Text** o un IDE como **NetBeans**. Al ser un proyecto en **PHP**, optaremos por descargar PHPStorm de Jetbrains, que nos dará funcionalidades extras para nuestro proyecto, como el autocompletado, la inclusión automática de clases y muchas otras funcionalidades que nos facilitarán la implementación del código de nuestro proyecto **TIGERBET**.



Podemos descargarlo desde su página oficial: <https://www.jetbrains.com/phpstorm/>

La versión que nos descargamos es una versión de prueba de 30 días, sin embargo podemos encontrar licencias en internet.

## INSTALACIÓN DE SYMFONY

Una vez tenemos todo el entorno instalado, llegó el momento de instalar nuestro framework a través de Composer.

Primeramente, debemos ir al fichero php.ini y des-comentar una línea de código para tener una máquina que trabaje sobre SSL.

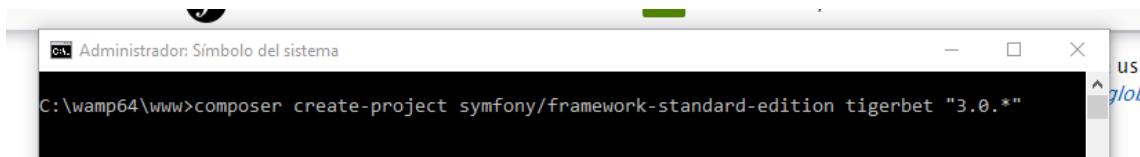
```
968 ;extension=php_oc18.dll
969 ;extension=php_oc18_11g.dll
970 extension=php_openssl.dll
971 ;extension=php_pdo_firebird.dll
972 ;extension=php_pdo_mssql.dll
```

Versiones inferiores de PHP inferiores a la v.5.4 hacen que salte un error en la instalación de Symfony. En ese caso, podemos realizar un upgrade o descargar el paquete que nos ofrece WAMP: <http://wampserver.aviaTechno.net/>

En cualquiera de los dos casos, ya podremos crear nuestro proyecto Tigerbet con Symfony.

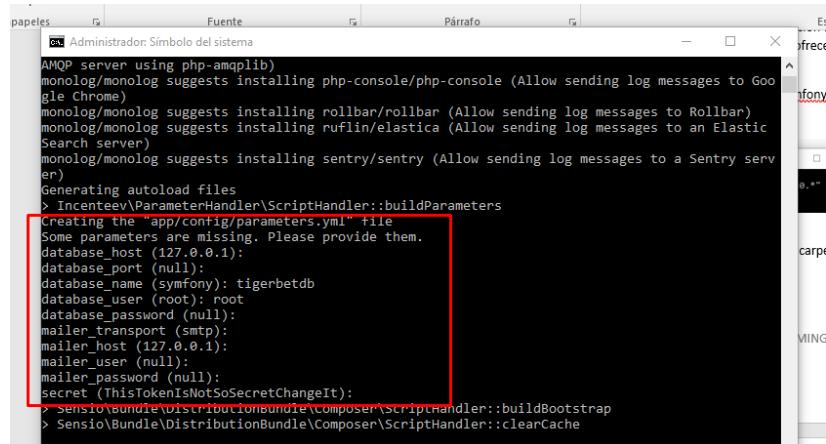
Arrancamos nuestra máquina wamp y ejecutamos el comando:

```
composer create-project symfony/framework-standard-edition tigerbet "3.0.*"
```



Comienzan a instalarse los paquetes necesarios y se crea el árbol de directorios en la carpeta www.

Configuramos la base de datos, el usuario y la password.



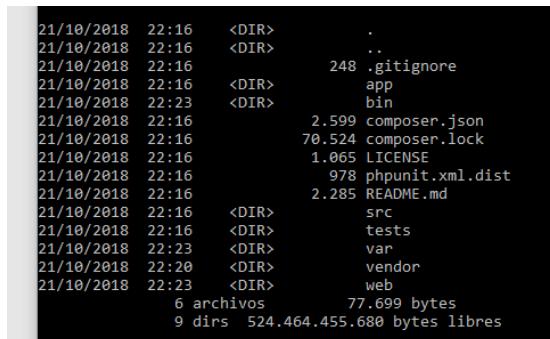
```

papeles    Fuente    Párrafo
ca Administrador: Símbolo del sistema
AMQP server using php-amqplib)
monolog/monolog suggests installing php-console/php-console (Allow sending log messages to Google Chrome)
monolog/monolog suggests installing rollbar/rollbar (Allow sending log messages to Rollbar)
monolog/monolog suggests installing ruflin/elastica (Allow sending log messages to an Elastic Search server)
monolog/monolog suggests installing sentry/sentry (Allow sending log messages to a Sentry server)
Generating autoload files
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1):
database_port (null):
database_name (symfony): tigerbetdb
database_user (root): root
database_password (null):
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
secret (ThisTokenIsNotSoSecretChangeIt):
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache

```

Y comprobamos que nuestros archivos y directorios se han creado correctamente.

Veremos más detalladamente cada uno de los directorios creados, pero de primeras, el directorio app será configuraciones, en src alojaremos nuestros controladores y vistas. El directorio vendor tiene todos los módulos y dependencias del proyecto. Por último, el directorio web contendrá los bundles y recursos estáticos de la aplicación. Probamos nuestra aplicación en el navegador: localhost/tigerbet/web

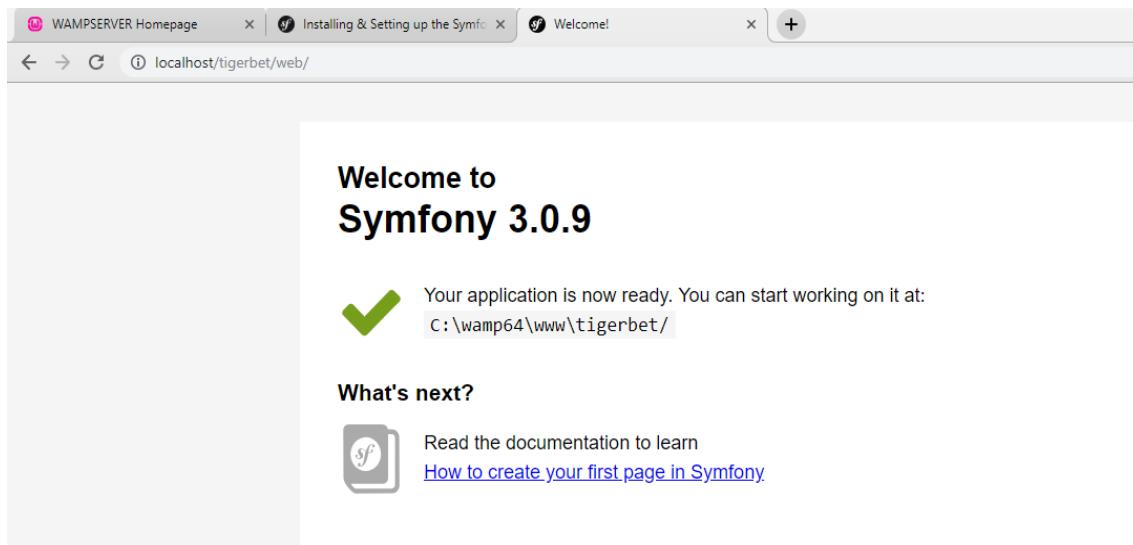


```

21/10/2018 22:16 <DIR> .
21/10/2018 22:16 <DIR> ..
21/10/2018 22:16 248 .gitignore
21/10/2018 22:16 <DIR> app
21/10/2018 22:23 <DIR> bin
21/10/2018 22:16 2.599 composer.json
21/10/2018 22:16 70.524 composer.lock
21/10/2018 22:16 1.065 LICENSE
21/10/2018 22:16 978 phpunit.xml.dist
21/10/2018 22:16 2.285 README.md
21/10/2018 22:16 <DIR> src
21/10/2018 22:16 <DIR> tests
21/10/2018 22:23 <DIR> var
21/10/2018 22:20 <DIR> vendor
21/10/2018 22:23 <DIR> web
6 archivos 77.699 bytes
9 dirs 524.464.455.680 bytes libres

```

De esta forma, tenemos el esqueleto de nuestra aplicación corriendo en el servidor.



## DESARROLLO DE TIGERBET

---

### ÁRBOL DE DIRECTORIOS

Una vez que tenemos el proyecto inicial de Symfony funcionando, vamos a echar un ojo a los directorios que se han creado en nuestra carpeta raíz tigerbet.

**app:** para configuraciones, plantillas y traducciones de nuestra aplicación.

**bin:** contiene el ejecutable de la consola de symfony.

**src:** aloja todo nuestro código fuente.

**test:** es una copia del directorio src. Es donde se realizan los test con PHPUnit.

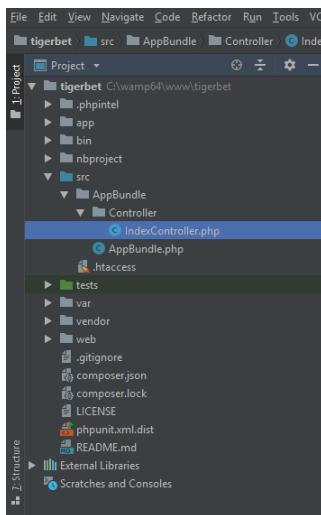
**bin:** contiene el ejecutable de la consola de symfony.

**var:** contiene los archivos de caché.

**vendor:** es el directorio donde se alojan las librerías.

**web:** contiene todos los estáticos de nuestra aplicación.

## CONTROLADORES Y HOLA MUNDO

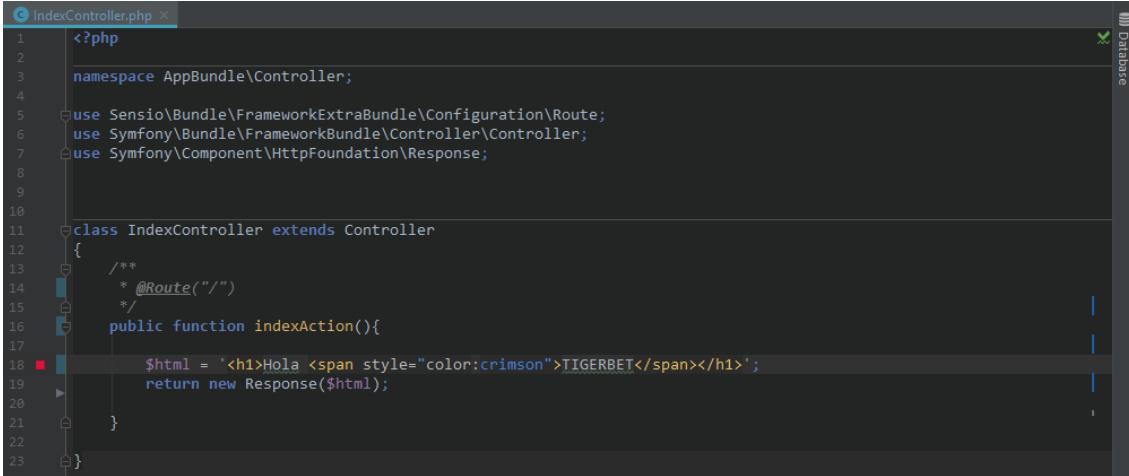


Nuestro proyecto contiene, como hemos visto, un árbol de directorios organizado por funcionalidades. Dentro del directorio src tendremos todo nuestro código fuente como hemos visto. Crearemos nuestro primer controlador y nuestro primer "HOLA MUNDO" con Symfony.

En el directorio src/AppBundle, creamos un directorio Controller y creamos nuestro controlador principal: IndexController.php.

Un controlador es una clase que extiende de la superclase Controller. Se encarga de obtener la información de la petición HTTP, y devuelve una respuesta, en forma de objeto Response.

Esta respuesta, puede ser una template (html), un objeto JSON, una imagen, un PDF, una redirección, un 404, etc. El controlador, además, contiene toda la lógica que necesita la aplicación.



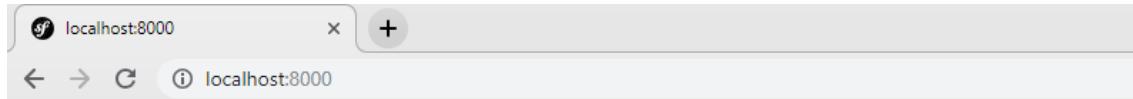
```

1 <?php
2
3 namespace AppBundle\Controller;
4
5 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
6 use Symfony\Bundle\FrameworkBundle\Controller\Controller;
7 use Symfony\Component\HttpFoundation\Response;
8
9
10
11 class IndexController extends Controller
12 {
13     /**
14      * @Route("/")
15     */
16     public function indexAction(){
17
18         $html = '<h1>Hola <span style="color:crimson">TIGERBET</span></h1>';
19         return new Response($html);
20
21     }
22 }
23

```

Para mostrar nuestro “**HOLA MUNDO**”, necesitamos 2 cosas. Por un lado una ruta, que en este ejemplo, será el directorio raíz “`/`” y por otro lado un controlador, que obtiene la petición del usuario y la transforma en un objeto Response, que devuelve una vista.

La ruta viene dada por el decorador `@Route("/")`. El resultado de nuestra codificación es:



## Hola TIGERBET

Veremos más adelante cómo configurar el fichero `app/config/routing.yml` para eliminar las notaciones `@Route` y completar el enrutamiento de la aplicación.

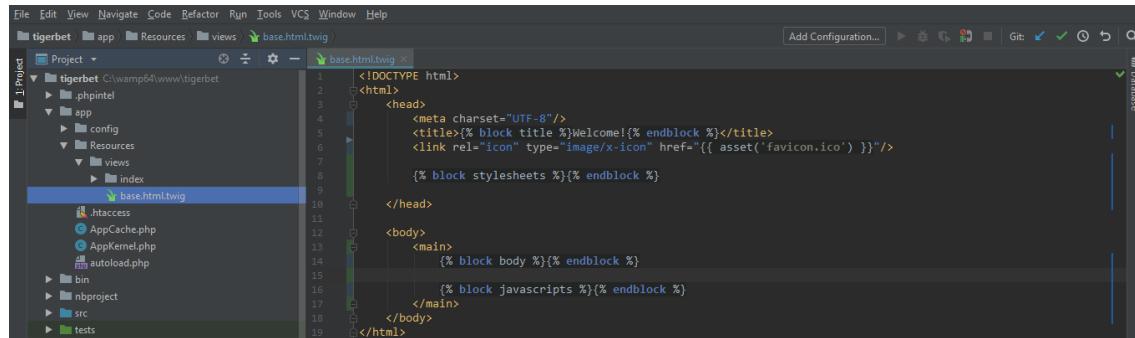
## VISTAS EN SYMFONY

El **sistema de vistas en Symfony** está basado en templates. El motor de plantillas de Symfony **es TWIG**.

Twig permite aplicar filtros, realizar bucles y muchas otras opciones que iremos viendo a lo largo del proyecto. Tomaremos como referencia la página oficial de Twig: <https://twig.symfony.com/>

Podemos crear una serie de plantillas y aplicarlas a cada una de las vistas. Además, estas templates admiten herencia y podemos incluir html customizado para las vistas. De esta forma podremos crear un archivo header.php y footer.php que se podrán heredar en cada una de las vistas.

Nuestra **plantilla base** se encuentra en el directorio: **app/Resources/views/base.html.twig**



The screenshot shows a code editor interface with the following details:

- File Path:** tigerbet/app/Resources/views/base.html.twig
- Code Content:**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>{% block title %}Welcome!{% endblock %}</title>
        <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}"/>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        <main>
            {% block body %}{% endblock %}
            {% block javascripts %}{% endblock %}
        </main>
    </body>
</html>
```
- Project Structure:**
  - Project: tigerbet (C:\wamp64\www\tigerbet)
  - app:
    - config
    - Resources:
      - views
      - index
  - bin
  - nbproject
  - src
  - tests

Tenemos distintos **bloques** que podrán ser sustituidos dependiendo del controlador al que llamemos. De la misma forma, podemos sustituir los estilos y el código javascript.

Crearemos una plantilla para nuestro **controlador de inicio** y haremos que herede de la **plantilla base**.

Modificamos nuestro fichero **base.html.twig**

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>{% block title %}Welcome!{% endblock %}</title>
        <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}"/>
        {% block stylesheets %}{% endblock %}
        <style>
            header,footer{width: 100%; padding: 10px; background: slategrey; color: #fff}
        </style>
    </head>
    <body>
        <main>
            <header>
                Header-template
            </header>

            <section class="main">
                {% block body %}{% endblock %}
            </section>

            <footer>
                Footer-template
            </footer>
        </main>
    </body>
</html>
```

y creamos el fichero: **app/Resources/index/index.html.twig**. Hacemos que el nuevo archivo **extienda** del twig **base**, y hacemos que los bloques del fichero base se sustituyan por los de nuestra vista de la ruta **@Route("/")**. Damos estilo al header y footer para diferenciar de nuestro **bloque body customizado**.

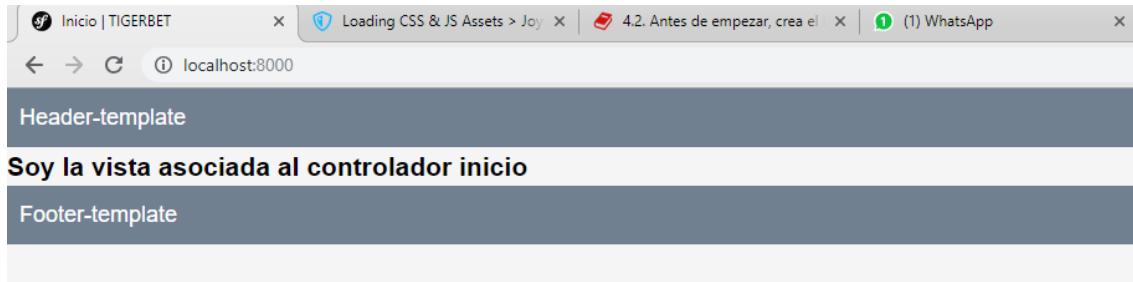
```
{% extends 'base.html.twig' %}

{% block title %}Inicio | TIGERBET{% endblock %}

{% block body %}
    <h3>Soy la vista asociada al controlador inicio</h3>
{% endblock %}

{% block stylesheets %}
    <style>
        *::after, *::before{margin:0; padding:0; box-sizing: border-box;}
        body { background: #F5F5F5; font: 10px/1.5 sans-serif; }
        h1 { line-height: 1.2; margin: 0 0 .5em; font-size: 36px; }
    </style>
{% endblock %}
```

El resultado será el siguiente:



Hemos conseguido crear una vista partiendo de un template y customizar los bloques de nuestra vista.

De la misma forma podemos customizar el **CSS** y **JS** de nuestras vistas, además, podremos crear tantos controladores como necesitemos, y asociarlos a una vista determinada.

Con esto podremos comenzar a crear nuestra aplicación **TIGERBET**, aunque aún nos queda mucho camino que recorrer.

## ENRUTAMIENTO AVANZADO

Uno de los puntos fuertes de Symfony es la capacidad de enrutamiento. De esta forma podemos manejar todas las peticiones y rutas del proyecto.

Hasta ahora enrutamos por medio de anotaciones: `@Route("/mi_ruta")`, pero ahora vamos realizar un enrutamiento avanzado editando el fichero: **app/routing.yml**.

Primero creamos un fichero **routing.yml** dentro de nuestro **AppBundle**. La ruta completa será: **app/AppBundle/Resources/config/routing.yml**. El fichero tendrá las rutas específicas del **bundle**, y para que Symfony reconozca estas rutas, debemos registrar el fichero en el **routing.yml**.

Podemos ahora quitar las anotaciones dentro de nuestro controlador de inicio.

Las configuraciones de los ficheros **routing.xml** quedarían de esta forma:

```

1 # app/config/routing.yml
2 rutas_index:
3     resource: "@AppBundle/Resources/config/routing.yml"
4     prefix: /
5
6 app:
7     resource: "@AppBundle/Controller/"
8     type: annotation
9

```

```

1 #AppBundle/Resources/config/routing.yml
2 ruta_principal:
3     path: /
4     defaults: {_controller: AppBundle:Index:index}
5

```

Quitamos la anotación **@Route** del controlador y vemos que nuestra aplicación sigue funcionando igual.



## TEMPLATE PADRE: INCLUYENDO EL FOOTER

Visto el sistema de plantillas de Symfony es momento retocar nuestra plantilla padre. El footer de nuestra web siempre será el mismo en toda la navegación, por lo que podremos implementar el código dentro de nuestro template.

```

22
23 <footer>
24     <div class="wrapper">
25         <p>Roberto Gómez Domínguez</p>
26         <p>TIGERBET - Proyecto 2ºDAW</p>
27         <p>rgomezdominguez@gmail.com</p>
28     </div>
29 </footer>

```

Y ahora incluiremos los estilos del footer dentro de la plantilla padre **footer.style.css**



El pie de página será ahora homogéneo para todas las páginas de la web.

## TEMPLATE PADRE: INCLUYENDO EL HEADER

De la misma forma que el pie de página, incluimos el **header** y damos algunos estilos generales. Incluimos dentro de la etiqueta head una llamada a las **fuentes de google** y algunas **metas**.

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <!-- METAS -->
5     <meta charset="UTF-8">
6     <meta name="robots" content="noindex">
7     <meta name="theme-color" content="#ff9500">
8     <meta name="description" content="TigerBet, casa de apuestas">
9     <meta name="keywords" content="apuestas deportivas,casino, blog deportes, proyecto2daw">
10    <meta name="author" content="Roberto Gómez Domínguez">
11    <meta name="viewport" content="width=device-width, initial-scale=1.0">
12
13    <title>{% block title %}Welcome!{% endblock %}</title>
14
15    <!-- FAVICON -->
16    <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}"/>
17    <!-- TIPOGRAFÍA -->
18    <link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">
19    <!-- CSS GENERAL DE LA WEB -->
20    <link rel="stylesheet" href="{{ asset('css/general.style.css') }}"/>
21    <link rel="stylesheet" href="{{ asset('css/header.style.css') }}"/>
22    <link rel="stylesheet" href="{{ asset('css/footer.style.css') }}"/>
23    <!-- ESTILOS PARA CADA PÁGINA -->
24    {% block stylesheets %}{% endblock %}
25

```

Nuestra web se va viendo a cada paso mucho más mejorada. **Refrescamos el navegador** y obtenemos claros resultados:

Bienvenido Casa de apuestas online Login

**TigerBet**

INICIO DEPORTES CASINO DEPORTES CASINO NOTICIAS CONTACTO MI CUENTA LOGIN

Ahora funciono mediante enrutamientos avanzados, sin la anotación @Route

Roberto Gómez Domínguez TIGERBET - Proyecto 2ºDAW rgomezdominguez@gmail.com

Hagamos un resumen de lo que leamos hasta ahora. Tenemos un **template padre** del que van a heredar las plantillas hijas. Todas las plantillas hijas van a contener el mismo **header** y **footer**, con los mismos **estilos** (evidentemente la web tiene que ser homogénea). Pero en el pantallazo anterior tenemos un menú de navegación completo, que no es el definitivo. Si nos fijamos bien, vemos que la pestaña **LOGIN** y **MI CUENTA**, son incompatibles... Si no me he registrado o logueado no tiene sentido que pueda acceder a MI CUENTA, mientras que, si ya estoy logueado, lo que no tiene sentido es que aparezca el LOGIN de nuevo. Además, según los requisitos, un usuario logueado va a poder navegar por toda la web, mientras que un usuario no registrado, no podrá acceder a ciertas secciones como **DEPORTES** y **CASINO**, que son secciones para realizar apuestas dentro de la web.

**¿Cómo solucionamos?** Bien, tendremos que plantear un sistema de registro/logueo, donde los usuarios registrados puedan acceder a todas las secciones mientras que los no logueados tendrán ciertas restricciones en la navegación web. Es aquí donde entrará en juego nuestro **modelo ER** y por tanto las **BASES DE DATOS**.

## NAVEGACIÓN COMPLETA

Antes de comenzar a recoger datos en el back-end, vamos a maquetar la web y crear una navegación entre todas las secciones. Al final de éste capítulo tendremos una **navegación completa entre páginas**.

Para poder navegar crearemos los controladores, que nos servirán las vistas. Con Symfony podemos crear los controladores desde la **consola de comandos**, y automáticamente, se generarán todas las

dependencias. Abrimos el **Símbolo del Sistema** y nos situamos sobre nuestro directorio, donde tenemos alojada la aplicación. Generamos el Bundle:

```
php bin/console generate:bundle --namespace=Acme/Bundle/EjemplotigerBundle
```

```
C:\wamp64\www\tigerbet>php bin/console generate:bundle --namespace=Acme/Bundle/EjemplotigerBundle
[37;44m [37;44m Welcome to the Symfony bundle generator! [37;44m
[37;44m [37;44m Are you planning on sharing this bundle across multiple applications? [39m [33mno[39m:
[37;44m
```

Configuramos el **Bundle** asignándole un nombre y un scope y seleccionamos el tipo de anotaciones para las configuraciones.

```
[32mAre you planning on sharing this bundle across multiple applications? [39m [33mno[39m: no
Your application code must be written in [33mbundles[39m. This command helps
you generate them easily.

Give your bundle a descriptive name, like [33mBlogBundle[39m.
[32mBundle name [33mAcme/Bundle/EjemplotigerBundle [39m: EjemplotigerBundle
[32mBundles are usually generated into the [32msrc/[39m directory. Unless you're
doing something custom, hit enter to keep this default!

[32mTarget Directory [39m [33msrc/[39m: src
What format do you want to use for your generated configuration?

[32mConfiguration format (annotation, yml, xml, php) [39m [33mannotation [39m: yml
[37;44m [39;49m
[37;44m Bundle generation [39;49m
[37;44m [39;49m

> Generating a sample bundle skeleton into [32mC:\wamp64\www\tigerbet\app/../src/EjemplotigerBundle [39m [33mOK![39m
> Checking that the bundle is autoloaded: [32mOK [39m
```

Los ficheros y dependencias se han creado correctamente.

```

[37;44m          [39;49m
[37;44m  Bundle generation [39;49m
[37;44m          [39;49m

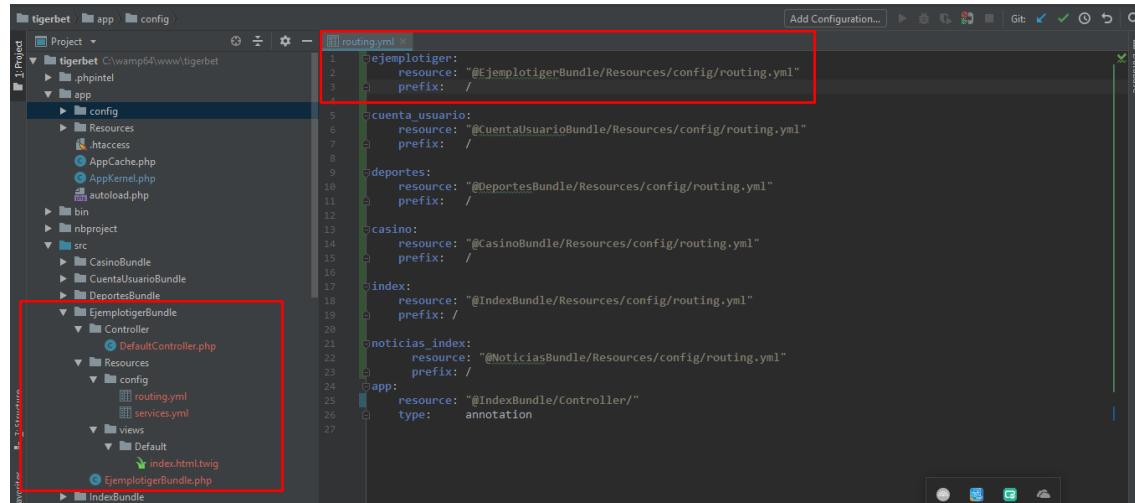
> Generating a sample bundle skeleton into [32mC:\wamp64\www\tigerbet\app\../src/EjemplotigerBundle[39m [33mOK![39m
> Checking that the bundle is autoloaded: [32mOK[39m
> Enabling the bundle inside [32mC:\wamp64\www\tigerbet\app\AppKernel.php[39m: [32mOK[39m
> Importing the bundle's routes from the [32mC:\wamp64\www\tigerbet\app\config\routing.yml[39m file: [32mOK[39m
> Importing the bundle's services.yml from the [32mC:\wamp64\www\tigerbet\app\config\config.yml[39m file: [32mOK[39m

[37;44m          [39;49m
[37;44m  Everything is OK! Now get to work :). [39;49m
[37;44m          [39;49m

C:\wamp64\www\tigerbet>

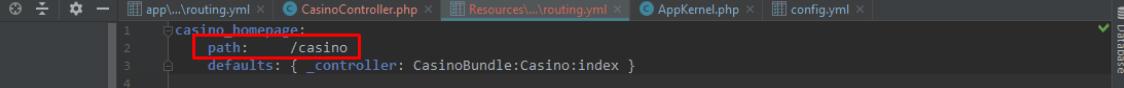
```

Verificamos que los ficheros se han creado.



Además, en el fichero **app/config/routing.yml** se ha registrado nuestra ruta. Creamos todos los controladores, uno por cada sección: **deportesBundle**, **casinoBundle**, **noticiasBundle**, etc...

Lo único que nos queda es cambiar el **path** del fichero **Resources/config/routing.yml** de cada uno de los Bundles y agregarle la ruta a la que el usuario hace el request.



```

1 1 casinohomepage:
2 2 path: /casino
3 3 defaults: { _controller: CasinoBundle:Casino:index }
4

```

En la **parte del front**, conseguimos la **navegación completa** a través de la web, a excepción de la pestaña login, que renderizará una ventana modal para el registro/login de usuarios mediante Ajax.



Bienvenido Casa de apuestas online Login

**TigerBet**

- [INICIO](#)
- [DEPORTES](#)
- [CASINO](#)
- [NOTICIAS](#)
- [CONTACTO](#)
- [MI CUENTA](#)
- [LOGIN](#)

Contenido deportes

Bienvenido Casa de apuestas online Login

**TigerBet**

- [INICIO](#)
- [DEPORTES](#)
- [CASINO](#)
- [NOTICIAS](#)
- [CONTACTO](#)
- [MI CUENTA](#)
- [LOGIN](#)

Contenido casino

Bienvenido Casa de apuestas online Login

**TigerBet**

- [INICIO](#)
- [DEPORTES](#)
- [CASINO](#)
- [NOTICIAS](#)
- [CONTACTO](#)
- [MI CUENTA](#)
- [LOGIN](#)

Contenido noticias

## CAPA FRONT: CONTENIDOS ESTÁTICOS

En este punto tenemos dos alternativas: meternos con la capa back (conexiones a bases de datos y modelos) o maquetar nuestra web. En el proyecto **se ha decidido primero tener una maqueta front** para hacernos a la idea del aspecto de la web y más adelante, crear los modelos a partir de la base de datos y obtener contenidos dinámicos.

No nos vamos a entretener en explicar el css y js utilizado, puesto que es mucho más interesante la creación de modelos y conexión a base de datos. Sí que es interesante la forma en que embebemos los scripts y estilos. Tendremos unos estilos generales, que **insertaremos en el template padre** (del que heredan el resto de vistas).

Mediante las etiquetas **{% include "mi\_fichero" %}** podremos hacer includes de los estilos y scripts js, ya sea en el template padre, o en las plantillas que heredan.

```

100 <!-- include --> /projecto_2-pm/p>
101     <p><a href="mailto:rgomezdominguez@gmail.com">rgomezdominguez@gmail.com</a></p>
102     </div>
103     </footer>
104     <!-- fin FOOTER -->
105   </main>
106
107   <script src="{{ asset('js/menu.script.js') }}" type="application/javascript"></script>
108
109 </body>
110 </html>

```

Generamos el CSS:

```

52 .col1-15{ width: 15%; }
53 .col1-85{ width: 85%; }
54 ■ .col1-33{ width: 33.33%; background: #ffffff; margin: 0px 20px; }
55 .col1-50{ width: 50%; position: relative; }
56
57 /*FLEX*/
58 header .wrapper, header div#rss, header div#rss .login, footer .wrapper, header .rss-icons,
nav#menu-normal .wrapper, nav#menu-normal ul, nav#menu-normal ul li a, .col-flex, .grid3,section#contacto
.wrapper,form,section#resultados #table, section#resultados #table, section#slider .slider-content,
section#slider .slider-content .col1-50 { display: -webkit-box; display: -moz-box; display: -ms-flexbox;
display: -webkit-flex; display: flex; }
header .wrapper, header .rss-icons, header div#rss, footer .wrapper, .flex{ justify-content: space-between;
flex-wrap: wrap; align-items: center; }

```

Y los scripts JS:

```

1  'use strict';
2  (function(d,w){
3      document.addEventListener('DOMContentReady',
4          'DOMContentLoaded', getPosicionHastaElTop, false);
5
6      function getPosicionHastaElTop(){
7          w.addEventListener("scroll", function(e){
8              let main = document.querySelector('main');
9              let distanciaAlTOP = main.getBoundingClientRect().top;
10             let menu = d.getElementById('menu-normal');
11
12             if(distanciaAlTOP < -150){
13                 menu.classList.add('fixed-menu');
14             }else{
15                 menu.classList.remove('fixed-menu');
16             }
17         }, true);
18     }
19 })(document, window));

```

En este caso, el script calcula la posición del contenido que estamos visualizando respecto al "top" de la ventana. Si la distancia del contenido al borde superior es mayor a 150px, el menú de navegación se va a desplazar con nosotros a medida que hacemos scroll, y además su opacidad será menor.

Después de **aplicar los estilos y las transiciones**, tendremos una web bastante atractiva, eso sí, con contenidos estáticos. Maquetamos el resto de páginas (mismo proceso).

The screenshot shows the homepage of the TigerBet website. At the top, there's a dark header bar with the TigerBet logo on the left and navigation links for INICIO, DEPORTES, CASINO, NOTICIAS, CONTACTO, MI CUENTA, and LOGIN. Below the header, there are three main content blocks. The first block, labeled 'DEPORTES', features a collage of sports-related silhouettes (soccer, basketball, tennis, etc.) and a 'Regístrate para acceder' button. The second block, labeled 'CASINO', shows a photo of people at a roulette table and a 'Regístrate para acceder' button. The third block, labeled 'NOTICIAS', shows a stack of newspapers and a 'Regístrate para acceder' button. At the bottom center, the text 'TIGER BET TM' is displayed.

## ACCESO A DATOS

Es aquí donde ponemos toda la carne en el asador. Primeramente, crearemos una base de datos con las tablas o entidades que obtenemos del **Diagrama ER** más otra tabla más para las noticias y el formulario de contacto.

En un segundo paso crearemos los modelos para obtener el mapeado de las consultas a la base de datos, y por último, modificaremos nuestras secciones de forma que nuestra web pasará de tener un contenido estático, a un contenido dinámico.

### EL MODELO ENTIDAD RELACIÓN

Las bases de datos nos permiten almacenar grandes cantidades de datos y usar esos datos de una manera eficaz y eficiente.

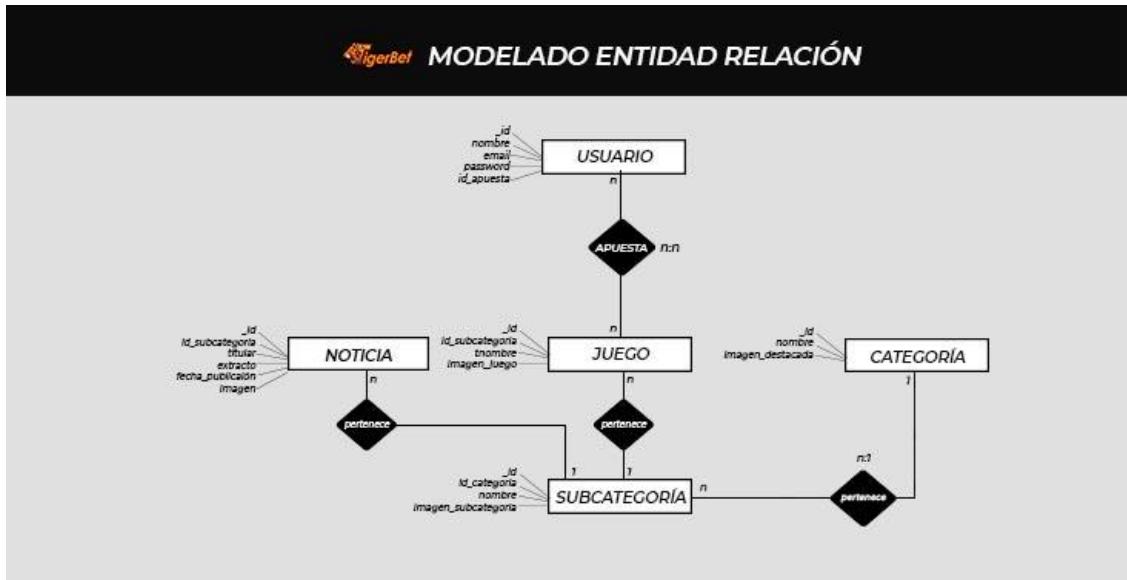
Aunque las **bases de datos no relacionales** como **MongoDB** están ganando mucho terreno en ciertas disciplinas, son las **bases de datos relacionales** como **Oracle** o **mySQL** las más utilizadas.

A grandes rasgos, el **MODELO ENTIDAD-RELACIÓN (ER)** es un método para diseñar los esquemas que debemos implementar en un sistema gestor de bases de datos (BBDD) como es mySQL.

El ER está compuesto por varios elementos:

- **ENTIDADES:** son cosas u objetos claramente diferenciados e independientes. Las entidades darán lugar a tablas en la BBDD.
- **ATRIBUTOS:** son las características de las entidades y serán nuestras columnas en las tablas de las entidades. Generan las **claves primarias (pk)**.
- **RELACIONES:** son los vínculos de dependencia entre entidades, hacen que se generen las **claves foráneas (fk)** e incluso nuevas entidades.

No entraremos mucho más en definiciones, puesto que no es el cometido de ésta documentación. Pasaremos directamente a explicar el ER generado para nuestra web.



Los **USUARIOS** de nuestra aplicación podrán apostar a los distintos juegos que se publiquen en nuestra web. Por otro lado, nuestra web tendrá en un principio tres **CATEGORÍAS**: noticias, casino y deportes. Dentro de cada una de las categorías, podremos tener **SUBCATEGORÍAS**, es decir, en la categoría NOTICIAS, tendremos noticias de fútbol, baloncesto, wrestlemania, etc... en la categoría de CASINO tendremos distintos juegos o subcategorías de típicos juegos de casino: ruleta, bingo, blackjack, ... y por último, en la subcategoría DEPORTES, tendremos apuestas para juegos deportivos (diferentes a las de los casinos).

Un usuario podrá apostar a tantos juegos como quiera (siempre que tenga saldo suficiente), por lo que se genera una relación **N:N** por lo que de ahí obtendremos una nueva tabla **APUESTA** que tendrá una clave compuesta por la clave primaria del usuario (**id\_usuario**) y la PK del juego (**id\_juego**).

Por otro lado, tendremos las noticias. Una noticia pertenecerá a una subcategoría (fútbol, baloncesto, tenis, ...) pero una subcategoría, por ejemplo "balonmano" podrá tener muchas noticias que se refieran a ese deporte, por lo que la relación que tenemos es una **1:N** con la correspondiente propagación de PK.

Para finalizar, tendremos las CATEGORÍAS y SUBCATEGORÍAS. Una categoría, por ejemplo "casino" tendrá muchas subcategorías (bingo, ruleta, tragaperras, ...) pero un "bingo" (subcategoría) únicamente va a pertenecer a una categoría, que es la de "casino". Volvemos a encontrarnos con una relación **1:N** por lo que volvemos a tener una propagación de clases.

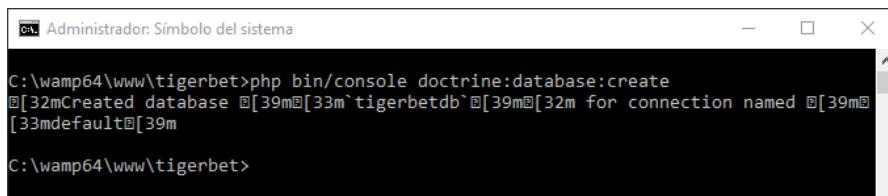
Una vez aclarado el diagrama ER, podemos **pasar nuestro modelo a una base de datos MySQL**.

## CREACIÓN DE LA BASE DE DATOS

El siguiente paso es crear las tablas según nuestro **modelo ER**. Si echamos un vistazo a nuestro diagrama, vemos que tendremos varias entidades:

1. **Usuario:** con atributos id, nombre, email, password, fecha de nacimiento y check edad.
2. **Juego:** id, id de categoría, nombre e imagen.
3. **Apuesta:** de la relación **muchos a muchos** tenemos una nueva entidad, la entidad apuesta, que se conoce como tabla pivote y contendrá como atributos: id, id\_usuario, id\_juego, id\_subcategoria, fecha de apuesta, cantidad apostada.
4. **Noticia:** id, id\_subcategoria, titular, extracto, fecha de publicación e imagen.
5. **Categoría:** tendremos 3 categorías, con id y nombre: deportes, casino y noticias.
6. **Subcategoría:** Con una relación **uno a muchos**. Atributos: id, id\_categoria, nombre e imagen.

Iniciamos nuestro **servidor WAMP**. Abrimos nuestra consola de comandos sobre el directorio de nuestro proyecto y creamos la base de datos **tigerbetdb**.



```
C:\wamp64\www\tigerbet>php bin/console doctrine:database:create
[32mCreated database [39m[33m`tigerbetdb`[39m[32m for connection named [39m[33mdefault[39m
C:\wamp64\www\tigerbet>
```

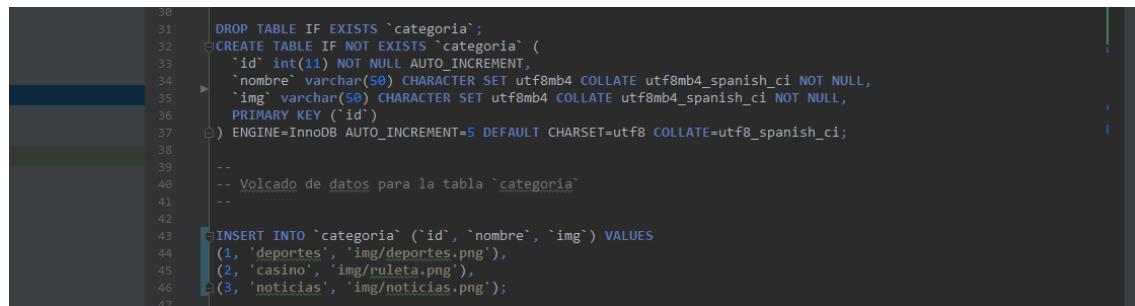
Los datos para la configuración inicial de la BBDD se piden al principio de la instalación de Symfony, pero podemos editar esta configuración dentro del fichero **app/config/parameters.yml**.

Podemos comprobar cómo se ha creado nuestra base de datos a través de la interfaz gráfica de mysql, **phpmyadmin**. Entramos, y cambiamos la codificación de caracteres: **utf8mb4\_spanish\_ci**.

## CREACIÓN DE TABLAS

Creamos las tablas dentro de la BBDD. Los scripts para crear las tablas e insertar los datos se encuentran en el archivo: **src/sql/tablas.sql**, importando ese archivo en la interfaz de mysql se generarán las tablas y las tuplas.

Creamos primero la **tabla categoría: id, nombre y ruta de imagen**.



```
30
31  DROP TABLE IF EXISTS `categoria`;
32  CREATE TABLE IF NOT EXISTS `categoria` (
33    `id` int(11) NOT NULL AUTO_INCREMENT,
34    `nombre` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci NOT NULL,
35    `img` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_spanish_ci NOT NULL,
36    PRIMARY KEY (`id`)
37  ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci;
38
39
40  -- Volcado de datos para la tabla `categoria`
41  --
42
43  INSERT INTO `categoria` (`id`, `nombre`, `img`) VALUES
44    (1, 'deportes', 'img/deportes.png'),
45    (2, 'casino', 'img/ruleta.png'),
46    (3, 'noticias', 'img/noticias.png');
47
```

Creamos e insertamos datos de la **tabla subcategoría: id, id\_categoria, nombre e imagen**. Contiene un id\_categoria que es la FK de la tabla categoría.

```

53
54  DROP TABLE IF EXISTS `subcategoria`;
55  CREATE TABLE IF NOT EXISTS `subcategoria` (
56      `id` int(11) PRIMARY KEY AUTO_INCREMENT,
57      `id_categoria` int(11) NOT NULL,
58      `nombre` varchar(50) COLLATE utf8mb4_spanish_ci NOT NULL,
59      `img` varchar(100) COLLATE utf8mb4_spanish_ci NOT NULL,
60      FOREIGN KEY (`id_categoria`)
61          REFERENCES categoria(id)
62          ON UPDATE CASCADE ON DELETE CASCADE
63  ) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;
64
65
66  -- Volcado de datos para la tabla `subcategoria`
67
68  INSERT INTO `subcategoria`(`id`, `id_categoria`, `nombre`, `imagen`) VALUES
69  (1, 1, 'futbol', 'img/futbol.png'),
70  (2, 1, 'baloncesto', 'img/baloncesto.png'),
71  (3, 2, 'ruleta', 'img/ruleta.png'),
72  (4, 2, 'bingo', 'img/bingo.png'),
73  (5, 3, 'fulbol', 'img/futbol.png'),
74  (6, 3, 'baloncesto', 'img/baloncesto.png'),
75  (7, 3, 'tenis', 'img/tenis.png'),
76  (8, 3, 'otros', 'img/otros.png'),
77  (9, 1, 'tenis', 'img/tenis.png'),
78  (10, 1, 'rugby', 'img/rugby.png');
79
80

```

Creamos la tabla contacto para recibir los mensajes de los usuarios a través del formulario de contacto.

```

86
87  DROP TABLE IF EXISTS `contacto`;
88  CREATE TABLE IF NOT EXISTS `contacto` (
89      `id` int(11) NOT NULL AUTO_INCREMENT,
90      `nombre` varchar(100) COLLATE utf8_spanish_ci NOT NULL,
91      `email` varchar(100) COLLATE utf8_spanish_ci NOT NULL,
92      `comentario` varchar(150) COLLATE utf8_spanish_ci NOT NULL,
93      `fecha` date NOT NULL,
94      PRIMARY KEY (`id`)
95  ) ENGINE=MyISAM AUTO_INCREMENT=11 DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci;
96

```

Creamos la tabla **usuarios**, que contendrá los datos para la autenticación de los usuarios registrados, e insertamos un primer usuario para poder realizar más adelante las pruebas de autenticación.

```

138
139  DROP TABLE IF EXISTS `usuarios`;
140  CREATE TABLE IF NOT EXISTS `usuarios` (
141      `id` int(11) NOT NULL AUTO_INCREMENT,
142      `nombre` varchar(50) COLLATE utf8_spanish_ci NOT NULL,
143      `email` varchar(120) COLLATE utf8_spanish_ci NOT NULL,
144      `password` varchar(50) COLLATE utf8_spanish_ci NOT NULL,
145      `fecha_ingreso` date DEFAULT NULL,
146      `check_edad` boolean DEFAULT FALSE,
147      PRIMARY KEY (`id`)
148  ) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci;
149
150  --
151  -- Volcado de datos para la tabla `usuarios`
152  --
153
154  INSERT INTO `usuarios` (`id`, `nombre`, `email`, `password`, `fecha_ingreso`, `check_edad`) VALUES
155  (1, 'roberto', 'rgomezdominguez@tigerbet.com', 'tigerbet123', '2018/10/11', TRUE);
156

```

Creamos la **tabla noticias**, que contendrá un id\_subcategoria que será la FK que apunta a las subcategorías.

```

102
103  DROP TABLE IF EXISTS `noticias`;
104  CREATE TABLE IF NOT EXISTS `noticias` (
105      `id` int(11) NOT NULL AUTO_INCREMENT,
106      `id_subcategoria` int(11) NOT NULL,
107      `slug` varchar(50) COLLATE utf8mb4_spanish_ci NOT NULL,
108      `titular` varchar(50) COLLATE utf8mb4_spanish_ci NOT NULL,
109      `extracto` varchar(150) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT NULL,
110      `contenido` text COLLATE utf8mb4_spanish_ci,
111      `fecha_publicacion` date DEFAULT NULL,
112      `imagen_destacada` varchar(50) COLLATE utf8mb4_spanish_ci DEFAULT NULL,
113      `galeria` varchar(255) COLLATE utf8mb4_spanish_ci DEFAULT NULL,
114      PRIMARY KEY (`id`),
115      UNIQUE KEY `titular` (`titular`),
116      KEY `id_subcategoria` (`id_subcategoria`)
117  ) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;
118
119

```

Insertamos unas cuantas noticias dentro de la tabla.

```

122
123  INSERT INTO `noticias` (`id`, `id_subcategoria`, `slug`, `titular`, `extracto`, `contenido`,
124      `fecha_publicacion`, `imagen_destacada`, `galeria`) VALUES
(1, 6, 'denver-nuggets', 'DENVER NUGGETS', 'Los Nuggets, dirigidos por el español Jordi Fernán están
onfire!', '<p>Grave afrenta a la estabilidad de grave afrenta a la estabilidad de un vaso es un vaso y un
plato es un plato lo más importante que se puede hacer por vosotros es lo que vosotros podáis hacer por
nosotros grave afrenta a la estabilidad de ¿eh? mucho españoles los vecinos el alcalde A veces moverse es

```

Queda una tabla por insertar, que es la **tabla apuesta**. Esta tabla pivoté tendrá 2 FK, una que apunta a la tabla usuario y otra a la **tabla subcategoría**.

```

179 DROP TABLE IF EXISTS `apuesta`;
180 CREATE TABLE IF NOT EXISTS `apuesta` (
181     `id` int(11) NOT NULL AUTO_INCREMENT,
182     `id_subcategoria` int(11) NOT NULL,
183     `id_usuario` int(11) NOT NULL,
184     `cantidad_apostada` int(11) NOT NULL,
185     PRIMARY KEY (`id`),
186     FOREIGN KEY (`id_subcategoria`, `id_usuario`)
187         REFERENCES subcategoria(id), usuario(id)
188         ON UPDATE CASCADE ON DELETE CASCADE
189     ) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci;
190
191 ALTER TABLE `apuesta`
192     ADD CONSTRAINT `subcategoria_apuesta` FOREIGN KEY (`id_subcategoria`) REFERENCES `subcategoria` (`id`);
193 ALTER TABLE `apuesta`
194     ADD CONSTRAINT `usuario_apuesta` FOREIGN KEY (`id_usuario`) REFERENCES `usuarios` (`id`);

```

## CREACIÓN DE ENTIDADES

Una vez que tenemos nuestra BBDD, tablas creadas e insertados los datos, es momento de crear las entidades.

Como sabemos, nuestra aplicación mostrará una serie de noticias, que debemos mostrar, por lo tanto, es lógico que tengamos una **entidad Noticia**. Crearemos esa entidad por medio de nuestro **CMD usando el comando entity: php bin/console generate:doctrine:entity**

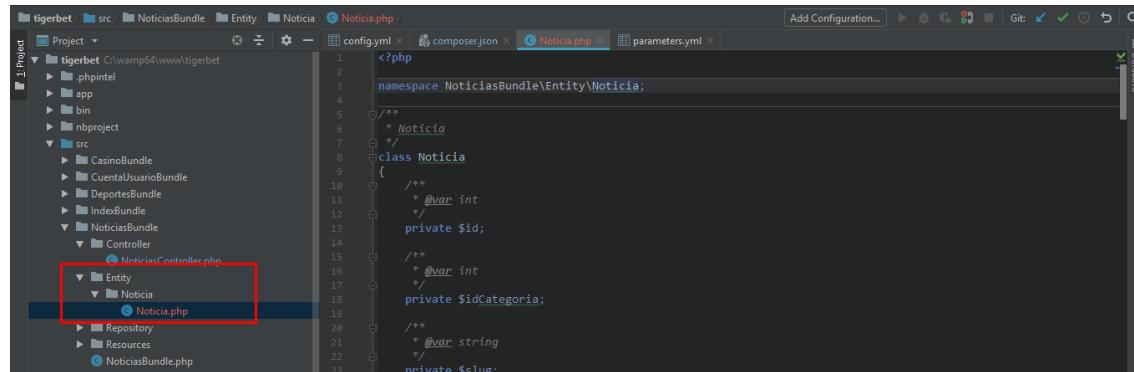
```

The Entity shortcut name [39m [33mNoticiasBundle:Noticias[39m]: NoticiasBundle:Noticias/Noticia
Determine the format to use for the mapping information.
Configuration format (yml, xml, php, or annotation) [39m [33mannotation[39m]: yml
Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named [33mid[39m).
Available types: [39m [33marray[39m, [33msimple_array[39m, [33mjson_array[39m, [33mobject[39m,
[33mboolean[39m, [33minteger[39m, [33msmallint[39m, [33mbigint[39m, [33mstring[39m, [33mtext[39m, [33mdatetime[39m, [33mdatetimetz[39m,
[33mdate[39m, [33mtime[39m, [33mdecimal[39m, [33mfloat[39m, [33mbinary[39m, [33mblob[39m, [33mguid[39m.

```

Creamos la entidad Noticia dentro del NoticiasBundle. Seguidamente se nos pedirán los atributos de la clase que se creará y el tipo de dato.

La **Entity Noticia se creará de forma automática en nuestro Bundle** y también se crearán los **getters y setters**.



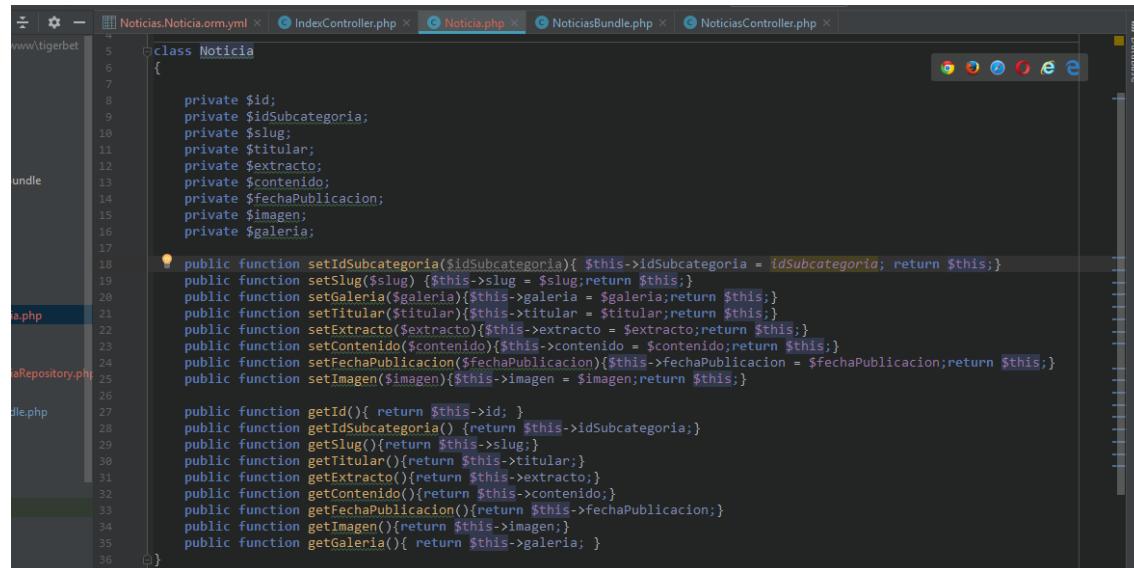
```
<?php
namespace NoticiasBundle\Entity\Noticia;

/**
 * Noticia
 */
class Noticia
{
    /**
     * @var int
     */
    private $id;

    /**
     * @var int
     */
    private $idCategoria;

    /**
     * @var string
     */
    private $slug;
```

Las siguientes entidades, las crearemos directamente sobre nuestro editor favorito. Simplemente, que sepamos que podemos crear estas entidades de dos formas.



```
class Noticia
{
    private $id;
    private $idSubcategoria;
    private $slug;
    private $titular;
    private $extracto;
    private $contenido;
    private $fechaPublicacion;
    private $Imagen;
    private $galeria;

    public function setIdSubcategoria($idSubcategoria){ $this->idSubcategoria = $idSubcategoria; return $this;}
    public function setSlug($slug) { $this->slug = $slug; return $this;}
    public function setGaleria($galeria){ $this->galeria = $galeria; return $this;}
    public function setTitular($titular){ $this->titular = $titular; return $this;}
    public function setExtracto($extracto){ $this->extracto = $extracto; return $this;}
    public function setContenido($contenido){ $this->contenido = $contenido; return $this;}
    public function setFechaPublicacion($fechaPublicacion){ $this->fechaPublicacion = $fechaPublicacion; return $this;}
    public function setImagen($Imagen){ $this->Imagen = $Imagen; return $this;}

    public function getId(){ return $this->id; }
    public function getIdSubcategoria(){ return $this->idSubcategoria; }
    public function getSlug(){ return $this->slug; }
    public function getTitular(){ return $this->titular; }
    public function getExtracto(){ return $this->extracto; }
    public function getContenido(){ return $this->contenido; }
    public function getFechaPublicacion(){ return $this->fechaPublicacion; }
    public function getImagen(){ return $this->Imagen; }
    public function getGaleria(){ return $this->galeria; }
```

## ACCESO A DATOS CON DOCTRINE

Symfony viene integrado con **Doctrine** que es un **ORM**, una librería de mapeo relacional de objetos, que permite la persistencia de datos y la lectura hacia y desde una BBDD, es decir, es una librería para la conexión de base de datos y poder realizar acciones de creación, lectura, actualización y borrado de datos, que es lo que conocemos como **CRUD**.

Tendremos muy a mano la página oficial de Doctrine para realizar las consultas a la BBDD en el desarrollo: <https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html>

Para realizar pruebas y ver que recuperamos los datos en el front, vamos al controlador de la página de inicio, **src/IndexBundle/Controller/IndexController.php** y utilizando los métodos que nos provee Doctrine, vamos a recoger todas las noticias que tenemos en BBDD. Haremos un **var\_dump** para ver los resultados.

```

8
9     public function indexAction(){
10         //recogemos el repositorio desde la clase Noticia
11         $repositorio = $this->getDoctrine()->getRepository( persistentObjectName: Noticia::class);
12         //Ejecutamos la consulta
13         $noticias = $repositorio->findAll();
14
15         //Imprimimos la primera noticia en el front
16         var_dump($noticias[0]);
17
18
19         return $this->render( view: 'index/index.html.twig', []);
20     }
21
22 }
```

El método **getRepository(class)** devuelve un objeto de repositorio que proporciona muchas formas de recuperar entidades del tipo especificado. En nuestra aplicación, recogeremos un objeto (de tipo array) que contendrá entidades **Noticias**. Podremos iterar este objeto y acceder a sus propiedades mediante los métodos **set** y **get**.

Lo que hacemos recogiendo el repositorio, es mapear los datos de la tabla noticias.

El resultado del **var\_dump** en el front será:

The screenshot shows a browser window with the URL `tigerbet:8000`. The page displays the source code of a Symfony application. The code is annotated with PHPDoc-style comments, showing relationships between entities like `NoticiasBundle\Entity\Noticias` and `NoticiasBundle\Entity\Noticia`. It also includes annotations for `private` fields and various methods like `get` and `set`.

Below the code, the TigerBet website homepage is visible. The header features the TigerBet logo, navigation links for **INICIO**, **DEPORTES**, **CASINO**, **NOTICIAS**, **CONTACTO**, **MI CUENTA**, and **LOGIN**. There is also a social media sharing section with icons for Facebook, Twitter, and Google+. The main banner features the Chelsea FC logo and the text "REGÍSTRATE AHORA Y LLÉVATE".

At the bottom of the browser window, there is a performance monitoring bar showing metrics such as response time (200 ms), route (@ ruta\_principal), and memory usage (17.5 MB).

Hemos obtenido los datos de la primera noticia. Con los **get** y **set** accedemos a los atributos del objeto.

The screenshot shows a browser window displaying the retrieved news article data in a structured format. The data includes:

- ID = 1
- ID\_SUBCATEGORIA = 6
- SLUG = denver-nuggets
- TITULAR = DENVER NUGGETS
- EXTRACTO = Los Nuggets, dirigidos por el español Jordi Fernán están onfire!
- CONTENIDO =
- FECHA = 2018-07-01
- RUTA\_IMAGEN = img/huggtes.png

Below the data, the TigerBet website homepage is visible, identical to the one shown in the previous screenshot.

Este proceso será el que llevemos a cabo para recuperar las noticias en el front. A través de una librería (Doctrine) recuperamos las noticias de la BBDD, mapeamos estas noticias en una clase (Noticia) y accedemos a los atributos y métodos del objeto.

La ventaja de utilizar este tipo de librerías, es que nos centramos mucho más en nuestra aplicación, los datos, consultas, etc... y nos despreocupamos totalmente de las conexiones, el cerrar la BBDD después de la consulta, etc... ¿Por qué inventar la rueda cuando ya está inventada?

## MAPEO DE LA BASE DE DATOS

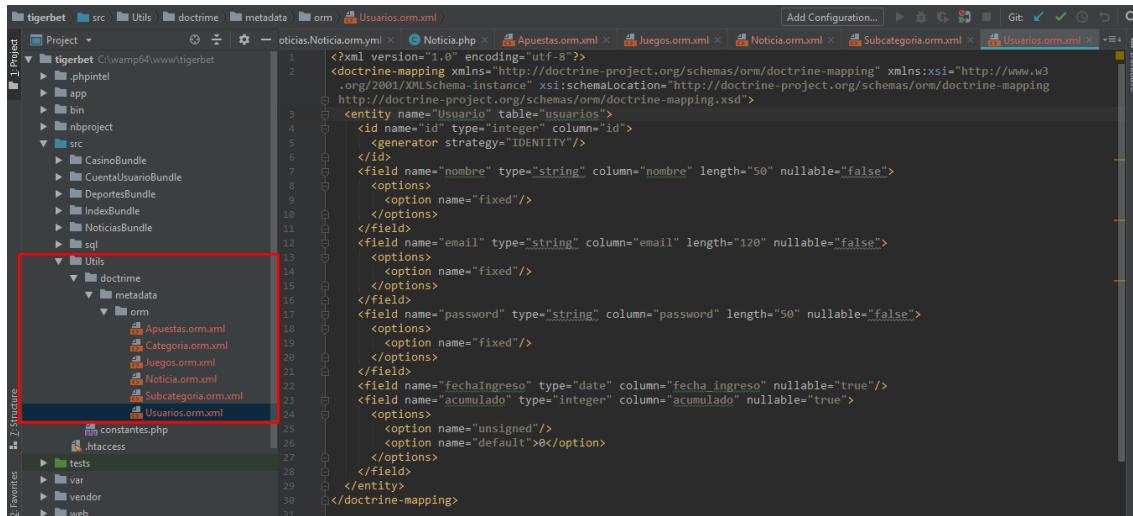
Como hemos visto, Doctrine nos permite traer la estructura de la base de datos a **un esquema xml**. Para ello usamos la consola de comandos y ejecutamos:

```
php bin/console doctrine:mapping:convert xml ./src/Utils/doctrine/metadata/orm --from-database --force
```

```
C:\wamp64\www\tigerbet>php bin/console doctrine:mapping:convert xml ./src/Utils/doctrine/metadata/orm --from-database --force
Processing entity "Apuestas"
Processing entity "Categoria"
Processing entity "Juegos"
Processing entity "Noticia"
Processing entity "Subcategoria"
Processing entity "Usuarios"

Exporting "xml" mapping information to "C:\wamp64\www\tigerbet\src\Utils\doctrine\metadata\orm"
C:\wamp64\www\tigerbet>
```

En nuestro proyecto, se han generado estos xml. Doctrine nos abstrae de la base de datos de modo que, si tenemos que migrar nuestra aplicación a Oracle, MariaDB, etc... siempre podremos mapear las tablas.



El siguiente paso es importar esos metadatos en un fichero **yml** de configuración. Para ello creamos un nuevo Bundle o Módulo al que llamaremos **ModelosBundle** donde generaremos nuestras clases desde el xml que hemos obtenido.

**php bin/console doctrine:mapping:import ModelosBundle yml**

```
C:\wamp64\www\tigerbet>php bin/console doctrine:mapping:import ModelosBundle yml
Importing mapping information from "[33mC:\wamp64\www\tigerbet\src\ModelosBundle\Resources\config\doctrine\Apuestas.orm.yml[39m"
> writing "[33mC:\wamp64\www\tigerbet\src\ModelosBundle\Resources\config\doctrine\Categoría.orm.yml[39m"
> writing "[33mC:\wamp64\www\tigerbet\src\ModelosBundle\Resources\config\doctrine\Juegos.orm.yml[39m"
> writing "[33mC:\wamp64\www\tigerbet\src\ModelosBundle\Resources\config\doctrine\Noticia.orm.yml[39m"
> writing "[33mC:\wamp64\www\tigerbet\src\ModelosBundle\Resources\config\doctrine\Subcategoria.orm.yml[39m"
> writing "[33mC:\wamp64\www\tigerbet\src\ModelosBundle\Resources\config\doctrine\Usuarios.orm.yml[39m"

C:\wamp64\www\tigerbet>
```

y en nuestro editor, comprobamos que se crean el Bundle y se generan los ficheros de configuración.

```

    type: entity
    table: apuestas
    indexes:
        id_juego:
            columns:
                - id_juego
        id_usuario:
            columns:
                - id_usuario
    id:
        id:
            type: integer
            nullable: false
            options:
                unsigned: false
            id: true
            generator:
                strategy: IDENTITY
    fields:
        apostado:
            type: integer
            nullable: false
            options:
                unsigned: false
        fechaApuesta:
            type: datetime
            nullable: false
            options:
                default: CURRENT_TIMESTAMP
    manyToOne:
        idJuego:
            column: id_juego
            targetEntity: Juegos
            cascade: { }
            fetch: LAZY
            mappedBy: null
            inverseBy: null
            joinColumns:
                id_juego:
                    referencedColumnName: id
                    orphanRemoval: false
        idUsuario:
            targetEntity: Usuarios
            cascade: { }
            fetch: LAZY
            mappedBy: null
            inverseBy: null
            joinColumns:
                id_usuario:
                    referencedColumnName: id
                    orphanRemoval: false

```

Doctrine además reconoce las relaciones creadas en las tablas a partir del modelo ER. Así, por ejemplo, la entidad **apuestas** que se relaciona con los **usuarios** y **juegos** con una relación de uno a muchos (**many to one**) se refleja también en nuestro archivo de configuración yml.

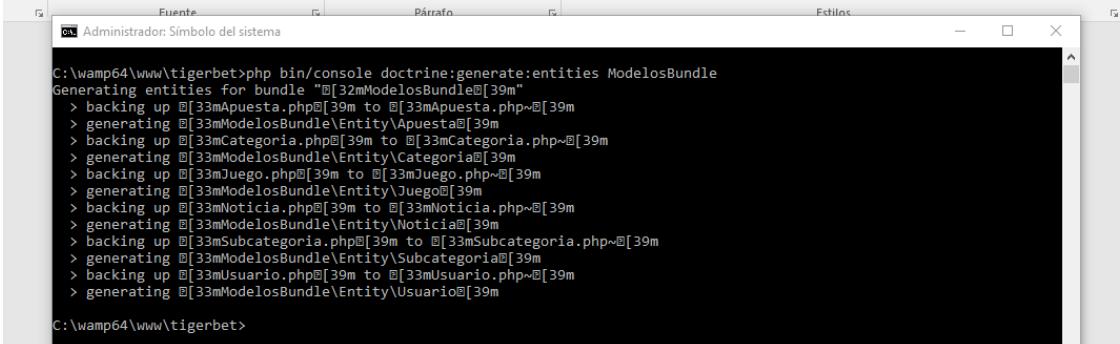
```

    type: entity
    table: apuestas
    indexes:
        id_juego:
            columns:
                - id_juego
        id_usuario:
            columns:
                - id_usuario
    id:
        id:
            type: integer
            nullable: false
            options:
                unsigned: false
            id: true
            generator:
                strategy: IDENTITY
    fields:
        apostado:
            type: integer
            nullable: false
            options:
                unsigned: false
        fechaApuesta:
            type: datetime
            nullable: false
            options:
                default: CURRENT_TIMESTAMP
    manyToOne:
        idJuego:
            column: id_juego
            targetEntity: Juegos
            cascade: { }
            fetch: LAZY
            mappedBy: null
            inverseBy: null
            joinColumns:
                id_juego:
                    referencedColumnName: id
                    orphanRemoval: false
        idUsuario:
            targetEntity: Usuarios
            cascade: { }
            fetch: LAZY
            mappedBy: null
            inverseBy: null
            joinColumns:
                id_usuario:
                    referencedColumnName: id
                    orphanRemoval: false

```

Ahora generaremos las clases con las que podremos interactuar.

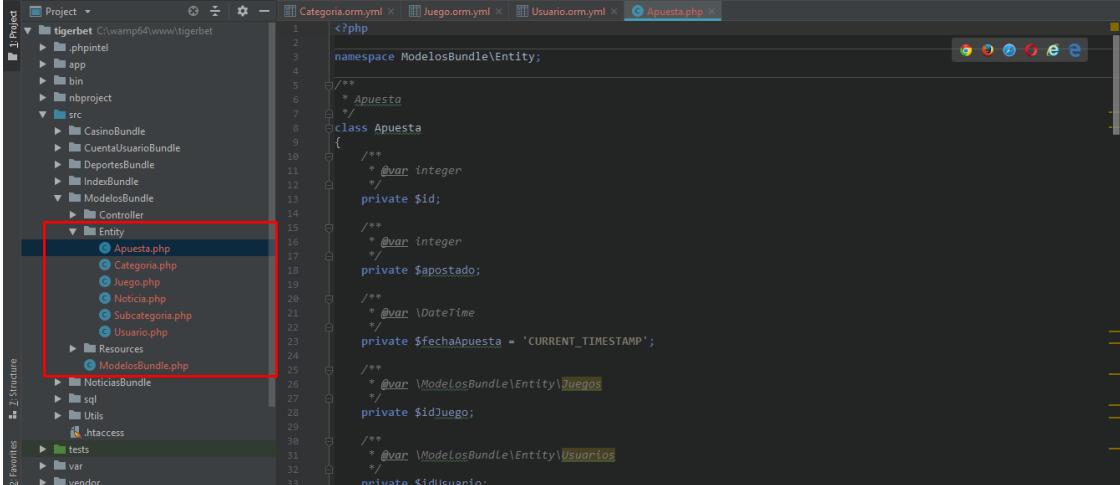
```
php bin/console doctrine:generate:entities ModelosBundle
```



```
C:\wamp64\www\tigerbet>php bin/console doctrine:generate:entities ModelosBundle
Generating entities for bundle "32mModelosBundle@39m"
> backing up 33mApuesta.php@39m to 33mApuesta.php~@39m
> generating 33mModelosBundle\Entity\Apuesta@39m
> backing up 33mCategoria.php@39m to 33mCategoria.php~@39m
> generating 33mModelosBundle\Entity\Categoría@39m
> backing up 33mJuego.php@39m to 33mJuego.php~@39m
> generating 33mModelosBundle\Entity\Juego@39m
> backing up 33mNoticia.php@39m to 33mNoticia.php~@39m
> generating 33mModelosBundle\Entity\Noticia@39m
> backing up 33mSubcategoria.php@39m to 33mSubcategoria.php~@39m
> generating 33mModelosBundle\Entity\Subcategoria@39m
> backing up 33mUsuario.php@39m to 33mUsuario.php~@39m
> generating 33mModelosBundle\Entity\Usuario@39m

C:\wamp64\www\tigerbet>
```

y comprobamos que se crean nuestras clases con los getters y setters.



```
<?php
namespace ModelosBundle\Entity;

/**
 * Apuesta
 */
class Apuesta
{
    /**
     * @var integer
     */
    private $id;

    /**
     * @var integer
     */
    private $apostado;

    /**
     * @var \DateTime
     */
    private $fechaApuesta = 'CURRENT_TIMESTAMP';

    /**
     * @var ModelosBundle\Entity\Juegos
     */
    private $idJuego;

    /**
     * @var ModelosBundle\Entity\Usuarios
     */
    private $idUsuario;
}
```

Se crean los modelos con los que vamos a poder interactuar en nuestra aplicación web.

A modo informativo, del mismo modo que generamos entidades a partir de la BBDD, podemos generar tablas a partir de un modelo, borrarlas y volver a crearlas.

## DOCTRINE: SQL, QUERYBUILDER, DQL

Como hemos visto, Doctrine tiene métodos como **findAll()** o **findOneBy...()** que permiten de una forma muy sencilla, el acceso a datos. Sin embargo, estos métodos no satisfacen todas las consultas que debemos realizar para nuestro proyecto.

Tenemos unas cuantas técnicas de consulta a BBDD, y utilizaremos unas u otras según nos convenga.

- podemos hacer consultas **SQL nativas**.
- Consultas **DQL**: Doctrine Query Language es un lenguaje muy intuitivo y muy parecido a SQL que está basado en consultas a entidades.
- **QueryBuilder**: es como su propio nombre indica, un constructor de queries, también muy sencillo de utilizar.

Utilizaremos una consulta con **QueryBuilder** las últimas 3 noticias por fecha de publicación.

```

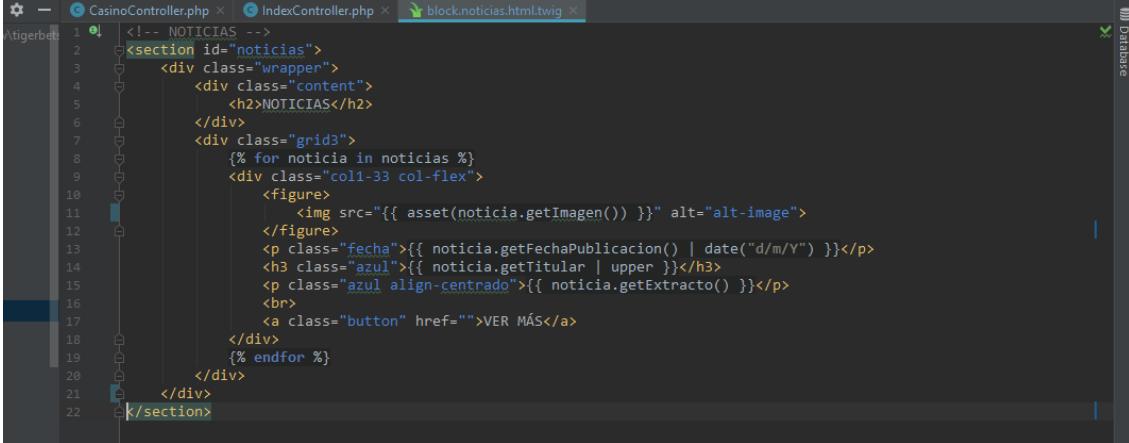
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
    $repo_noticias = $this->getDoctrine()->getRepository( persistentObjectName: Noticia::class );
    $query_noticias = $repo_noticias->createQueryBuilder( alias: 'n' )
        ->setMaxResults( maxResults: 3 )
        ->addOrderBy( sort: 'n.fechaPublicacion', order: 'DESC' )
        ->getQuery();
    $noticias = $query_noticias->getResult();
    //var_dump($noticias);

    //Renderización de vista y paso de parámetros
    return $this->render( view: 'index/index', [
        'subcategorias' => $subcategorias,
        'noticias'       => $noticias
    ]);
}

```

Recogemos el repositorio de noticias a través de Doctrine y comenzamos a construir la query, seleccionando las 3 últimas noticias (**setMaxResults**) ordenadas por fecha de publicación descendente (**addOrderBy**) y por último ejecutamos la query (**getResult**). Almacenamos el resultado en la variable noticias (**\$noticias**), que pasamos a la vista.

Lo único que nos falta es iterar nuestro objeto en la vista de forma que podamos pintar nuestras noticias en el grid que hemos construido.



```

1 1 <!-- NOTICIAS -->
2 2 <section id="noticias">
3 3   <div class="wrapper">
4 4     <div class="content">
5 5       <h2>NOTICIAS</h2>
6 6     </div>
7 7     <div class="grid3">
8 8       {% for noticia in noticias %}
9 9         <div class="col1-33 col-flex">
10 10           <figure>
11 11             
12 12           </figure>
13 13           <p class="fecha">{{ noticia.getFechaPublicacion() | date("d/m/Y") }}</p>
14 14           <h3 class="azul">{{ noticia.getTitular() | upper }}</h3>
15 15           <p class="azul align-centrado">{{ noticia.getExtracto() }}</p>
16 16           <br>
17 17           <a class="button" href="">VER MÁS</a>
18 18       </div>
19 19     <% endfor %>
20 20   </div>
21 21 </section>

```

En nuestra vista, construida a partir de **twig**, iteramos nuestra variable **noticias** mediante un bucle for. Se pinta así el resultado en la página de inicio.



Noticia	Imagen	Título	Fecha	Opción
ESTUDIANTES - BARCA		Los azulgranas pierden su segundo partido en tres días	15/07/2018	<a href="#">VER MÁS</a>
PENSIONES & RULETA		Chicago necesita ingresos, así que el gobierno municipal	15/07/2018	<a href="#">VER MÁS</a>
DALER KUZYAEV		El mediocampista del Zenit parecía ser el único ru	14/07/2018	<a href="#">VER MÁS</a>

Podemos comprobar mediante la fecha de publicación de las noticias, cómo obtenemos las últimas noticias ordenadas por fecha descendente.

En las siguientes secciones (**Blog de noticias, deportes y casino**) iremos usando el resto de técnicas: SQL Nativo y DQL.

También tenemos que decir, que interactuar con la base de datos desde el controlador, no es la forma más correcta. Para ello Symfony nos trae unas clases especiales llamadas **Repository** que son las encargadas de interactuar con nuestra BBDD.

Hablaremos de los repositorios más adelante.

## NOTICIAS: GENERACIÓN DE POSTS

Si utilizamos la consola de desarrollador, veremos que cada **button VER MÁS** apunta a la propia noticia.

The screenshot shows a news grid with three items. Each item includes a thumbnail, a date, a title, a brief description, and a 'VER MÁS' button.

- ESTUDIANTES - BARCA**  
Los azulgranas pierden su segundo partido en tres días  
[VER MÁS](#)
- PENSIONES & RULETA**  
Chicago necesita ingresos, así que el gobierno municipal  
[VER MÁS](#)
- DALER KUZYAEV**  
El mediocampista del Zenit parecía ser el único ru  
[VER MÁS](#)

A developer's browser tools (Elements tab) are overlaid on the page, showing the HTML structure and the CSS styles applied to the 'VER MÁS' button. The CSS for the button is:

```
.button {
    text-align: center;
    width: 65px;
    color: #ffffff;
    background: #011352;
    padding: 10px 40px;
    border: 2px solid #011352;
    border-radius: 7px;
    transition: all 300ms ease-in-out;
}
```

En el controlador de Noticias dentro del **NoticiasBundle** hemos creado un nuevo método; **blogAction** donde hacemos una nueva llamada a la BBDD recogiendo el slug de la URL para traernos al front la noticia seleccionada por el usuario.

```

16
17     public function blogAction($slug = null){
18         $repo_noticias = $this->getDoctrine()->getRepository( persistentObjectName: "ModelosBundle:Noticia");
19         $noticia = $repo_noticias->findBy(array('slug' => $slug));
20
21         //redireccionar a 404 - Noticia no encontrada
22         if(empty($noticia)){
23             return $this->redirect( url: '/noticias');
24         }
25
26         return $this->render( view: 'noticias/noticia.html.twig', [
27             'noticia' => $noticia
28         ]);
29     }
30
31 }
```

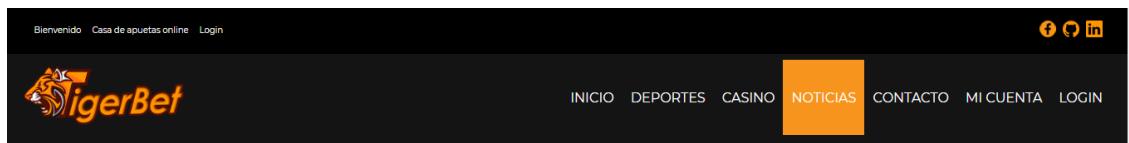
Lo que realmente hacemos en el controlador, es recoger la noticia por GET, consultar a la BBDD por esa URL y mediante el método **findBy**, traer al frente esa noticia.



The screenshot shows the TigerBet website interface. At the top, there's a navigation bar with links for INICIO, DEPORTES, CASINO, NOTICIAS, CONTACTO, MI CUENTA, and LOGIN. The main content area features a large image of a basketball game between ESTUDIANTES and BARCA. Below the image, the text "Los azulgranas pierden su segundo partido en tres días" is displayed. To the right, there's a sidebar titled "SECCIONES" with three sections: "CASINO" (showing a roulette table), "DEPORTES" (showing a soccer match), and "NOTICIAS".

Y así, con muy pocas líneas de código conseguimos mostrar todas las entradas de nuestro blog mediante una plantilla **Twig** personalizada para el blog de noticias.

Hemos conseguido, por un lado, en la página de inicio mostrar las 3 últimas noticias y que cada noticia apunte a una misma plantilla dinámica que cambiará atendiendo a la petición get que reciba el controlador. El paso para crear la sección de noticias no es nada complicado, lo único que haremos será crear una vista grid donde se mostrarán todas las noticias y cada una de ellas apuntará a la plantilla que hemos creado para los single-post.



El procedimiento es siempre el mismo, desde el controlador, llamamos al modelo (en este caso **Noticia**) que recupera las filas de la BBDD. El modelo entrega los datos al controlador, y el controlador pasa estos datos a la vista TWIG que los manipula de forma que mostramos nuestra sección.

El procedimiento para la creación del grid de **Casino** y **Deportes** es el mismo que para **Noticias**. Lo único que cambia es la **query para traer los datos**. Si observamos la base de datos, dentro de la **tabla juegos**, tenemos lo siguiente:

The screenshot shows the MySQL Workbench interface with a database tree on the left containing 'sys', 'tigerbetdb', 'Nueva', 'apuestas', 'categoria', 'contacto', 'juegos' (which is selected), 'noticia', 'subcategory', and 'usuarios'. To the right is a table named 'juegos' with the following data:

		id	id_subcategoria	nombre	imagen
		1	2	tragaperras	img/tragaperras.jpg
		2	2	ruleta	img/ruleta.png
		3	2	bingo	img/bingo.png
		11	3	futbol	img/futbol.png
		12	3	baloncesto	img/baloncesto.png
		13	3	wrestlemania	img/wrestlemania.jpg
		14	2	Engancha el pato	img/hookaduck.jpg
		15	3	Carreras de caracoles	img/carrera-caracoles.jpg

La tabla **juegos** contiene tanto los juegos para apuestas de deportes como las apuestas para casino. El **id\_subcategoria** es lo que distingue que un juego pertenezca a la sección de casino o de deportes.

Por tanto, para el grid de casino traerá los juegos con **id\_subcategoria 2**, mientras que la sección de deportes traerá las tuplas con **id\_subcategoria 3**. La consulta a base de datos será:

```

13 public function indexAction()
14 {
15     $games = null;
16
17     //Seleccionar las entradas que sean de la subcategoria CASINO
18     $repositorio_casino = $this->getDoctrine()->getRepository(persistentObjectName: Juego::class);
19     $games = $repositorio_casino->findBy(array('idSubcategoria' => '2'));
20
21     return $this->render( view: 'casino/casino', ['games'=> $games]);
22

```

Y de esta forma tendremos un grid deportes en la **sección de deportes**

The image shows a grid of four sports categories. The top row contains 'FÚTBOL' with a soccer ball image and 'BALONCESTO' with a basketball image. Both have 'ENTRAR AL JUEGO' buttons. The bottom row contains 'LUCHA LIBRE' with a wrestling match image and 'SNAIL RACING' with two snails racing on a track. A sidebar on the right titled 'SECCIONES' lists 'CASINO' (with a roulette wheel image), 'DEPORTES' (with a soccer/basketball silhouette image), and 'NOTICIAS'.

Y una sección grid de juegos de casino en la **sección de casino**.

The image shows a grid of three casino game sections. The top row contains 'TRAGAPERRAS' with a slot machine image and 'RULETA' with a roulette wheel image, both with 'ENTRAR AL JUEGO' buttons. The bottom row contains another 'CASINO' section with a roulette wheel image. A sidebar on the right titled 'SECCIONES' lists 'CASINO' (with a roulette wheel image) and 'DEPORTES'.

Al igual que en las noticias, usaremos el parámetro get para recoger los juegos o deportes. Veremos esto más adelante, cuando nos metamos a explicar el JS de los juegos y deportes.

## FORMULARIO DE CONTACTO

Hemos dejado de lado una parte importante de nuestra página de inicio, **el formulario de contacto.**

The screenshot shows the TigerBet website's contact page. At the top, there's a dark header with the 'TigerBet' logo on the left and a navigation menu with links for INICIO, DEPORTES, CASINO, NOTICIAS, CONTACTO (which is highlighted in orange), MI CUENTA, and LOGIN. Below the header is a large blue section containing a stylized orange tiger logo. To the right of the logo is a form titled 'CONTACTA CON NOSOTROS' with three input fields: 'Nombre', 'Email', and 'Comentarios'. At the bottom right of this section is a green 'ENVIAR' button. On the left side of the blue section, there are two short paragraphs of text in Spanish.

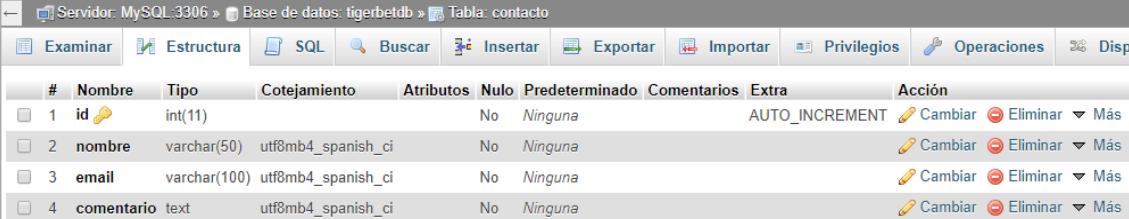
Es mejor estar quieto y en otras no a veces es mejor estar quieto y en otras no lo que no van a hacer nunca las máquinas es fabricar máquinas una cosa es ser solidario y otra es serlo a cambio de nada salvo algunas cosas.

Un vaso es un vaso y un plato es un plato mejor para mí el suyo, beneficio político ese señor del que usted me habla y en ocasiones es mejor estar en movimiento cuanto mejor peor para todos

En nuestra web hemos creido conveniente tener un formulario de contacto donde los usuarios puedan ponerse en contacto con el administrador en caso de tener dudas o consultas, e incluso poder reportar incidencias en cuanto a sus apuestas, juegos, etc...

Aunque Symfony permite crear formularios de una forma muy sencilla, a partir de entidades, hemos preferido realizar este formulario a la vieja usanza. Cuando el usuario pulse el botón de enviar se van a desencadenar una serie de eventos que tendrán como finalidad recoger los datos del formulario y persistir esos datos en nuestra BBDD.

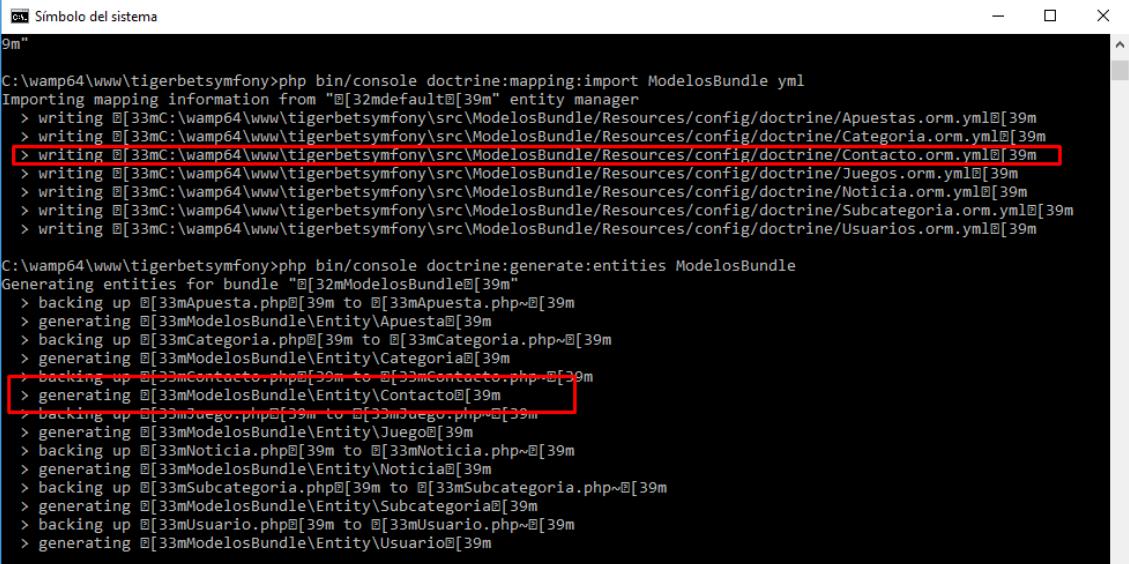
Hemos creado una **tabla contacto** en nuestra base de datos con la siguiente estructura.



#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	<b>id</b>	int(11)			No	Ninguna		AUTO_INCREMENT	Cambiar  Eliminar  Más
2	<b>nombre</b>	varchar(50)	utf8mb4_spanish_ci		No	Ninguna			Cambiar  Eliminar  Más
3	<b>email</b>	varchar(100)	utf8mb4_spanish_ci		No	Ninguna			Cambiar  Eliminar  Más
4	<b>comentario</b>	text	utf8mb4_spanish_ci		No	Ninguna			Cambiar  Eliminar  Más

Al crear una nueva tabla, tenemos que usar doctrine para realizar un mapeo y obtener nuestras clases para la que será nuestra nueva **entidad Contacto**.

Generamos los mappers y las clases dentro de nuestro proyecto en el **BundleContacto**.



```

Símbolo del sistema
9m"
C:\wamp64\www\tigerbetsymfony>php bin/console doctrine:mapping:import ModelosBundle yml
Importing mapping information from "C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Apuestas.orm.yml" [39m
> writing C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Categoría.orm.yml [39m
> writing C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Contacto.orm.yml [39m
> writing C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Juegos.orm.yml [39m
> writing C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Noticia.orm.yml [39m
> writing C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Subcategoria.orm.yml [39m
> writing C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Resources\config\doctrine\Usuarios.orm.yml [39m

C:\wamp64\www\tigerbetsymfony>php bin/console doctrine:generate:entities ModelosBundle
Generating entities for bundle "C:\wamp64\www\tigerbetsymfony\src\ModelosBundle"
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Apuesta.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Apuesta.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Apuesta [39m
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Categoría.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Categoría.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Categoría [39m
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Contacto.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Contacto.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Contacto [39m
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Juego.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Juego.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Juego [39m
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Noticia.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Noticia.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Noticia [39m
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Subcategoria.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Subcategoria.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Subcategoria [39m
> backing up C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Usuario.php [39m to C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Usuario.php~ [39m
> generating C:\wamp64\www\tigerbetsymfony\src\ModelosBundle\Entity\Usuario [39m

```

Y en nuestro código se generan de forma automática las clases que podremos utilizar en el proyecto.

Creamos también un **controladorContacto** que será el encargado de insertar los datos que provienen del formulario en nuestra **tabla contacto** de la BBDD.

A modo de resumen, cuando el usuario rellene el formulario y pulse el **botón enviar**:

- Tendremos que **validar** primeramente los datos de entrada, que los campos no estén vacíos, limpiar las entradas para evitar inyecciones SQL, etc...
- Seguidamente tendremos que **recoger los datos** del formulario y acumularlos en un objeto.
- Hacer una **llamada ASYNC** a nuestro controladorContacto, que llamará al modelo para insertar en BBDD,
- Dependiendo de la **response** tendremos que actuar de una forma u otra en el front, para informar al usuario si hemos recibido sus datos o por el contrario ha ocurrido algún error.

Para esto utilizaremos en el lado cliente JS y la librería jQuery que hará la llamada al controlador. El controlador llamará al modelo que insertará el registro en BBDD y de nuevo se devolverá el control en el front donde informaremos al usuario de la transacción.

## INICIALIZACIÓN SCRIPTS FORMULARIO

Para comenzar incluimos en el index unos cuantos scripts. El CDN de jQuery.

```

20
21  {% block javascript %}
22    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
23    <script src="{{ asset('js/vars.script.js') }}></script>
24    <script src="{{ asset('js/form.script.js') }}></script>
25    <script src="{{ asset('js/contacto.script.js') }}></script>
26  {% endblock %}
27

```

Un script de inicialización de variables. Para nuestro formulario inicializamos el **objeto FORM**.

```

1 var USER = {};
2 var POPUP = {};
3 var FORM = {};
4 var TRAGAPERRAS = {};

```

Un script de inicialización: **init.script.js** encargado de lanzar objetos cuando se desencadene un evento. (en el caso del formulario, el click sobre el botón ENVIAR).



```

1  'use strict';
2  document.addEventListener('DOMContentLoaded', function(argument) {
3
4      /*FORMULARIO DE CONTACTO*/
5      let btn_enviarcomentario = document.getElementById('send-form');
6
7      if(btn_enviarcomentario != null){
8          btn_enviarcomentario.addEventListener('click', (ev)=>{
9              ev.preventDefault();
10
11             FORM.init(document.forms[1].elements);
12
13         }, true);
14     }
15

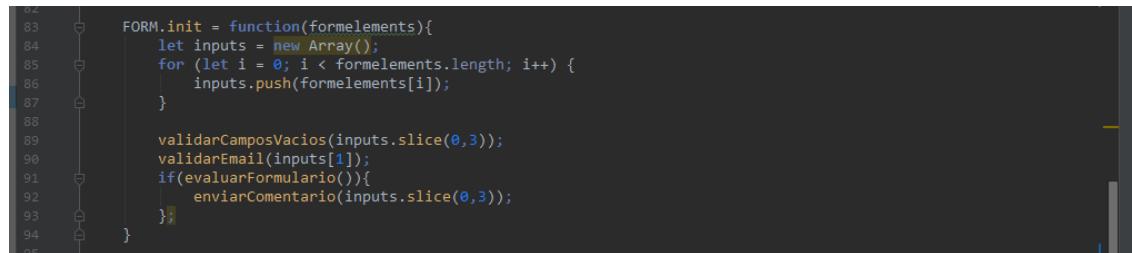
```

Un script **contacto.script.js** que validará el formulario, creará una llamada POST al controlador del formulario y actuará de una forma u otra atendiendo al **response** que se devuelva a la vista.

## VALIDACIÓN DE DATOS DE ENTRADA

En nuestro script contacto tenemos toda la lógica para la validación del formulario.

Tenemos varios campos, y dependiendo del tipo de cada uno, tendremos que realizar distintos tipos de validaciones.



```

82
83     FORM.init = function(formelements){
84         let inputs = new Array();
85         for (let i = 0; i < formelements.length; i++) {
86             inputs.push(formelements[i]);
87         }
88
89         validarCamposVacios(inputs.slice(0,3));
90         validarEmail(inputs[1]);
91         if(validarFormulario()){
92             enviarComentario(inputs.slice(0,3));
93         }
94     }
95

```

Por un lado, tendremos que los campos no estén vacíos. Esto lo haremos mediante la función `validarCamposVacios( )`:

```

55
56    function validarCamposVacios(elements){
57        campos_vacios = [];
58        for (let i = 0; i < elements.length; i++) {
59            elements[i].nextElementSibling.innerHTML = '';
60            if(elements[i].value.trim().length == 0){
61                elements[i].nextElementSibling.innerHTML = 'El campo no puede quedar vacío';
62                campos_vacios.push(false);
63            }else{
64                campos_vacios.push(true);
65            }
66        }
67    }
68

```

Si alguno de los campos queda vacío, tendremos que mostrar un mensaje al usuario pidiéndole que rellene esos campos.



The screenshot shows a contact form titled "CONTACTA CON NOSOTROS". It features a large orange tiger logo on the left. The form fields include "Nombre" (with an error message "El campo no puede quedar vacío"), "Email" (containing "roberto@gmail.com"), and "Comentarios". A large text area for comments has an error message "El campo no puede quedar vacío". At the bottom is an orange "ENVIAR" button.

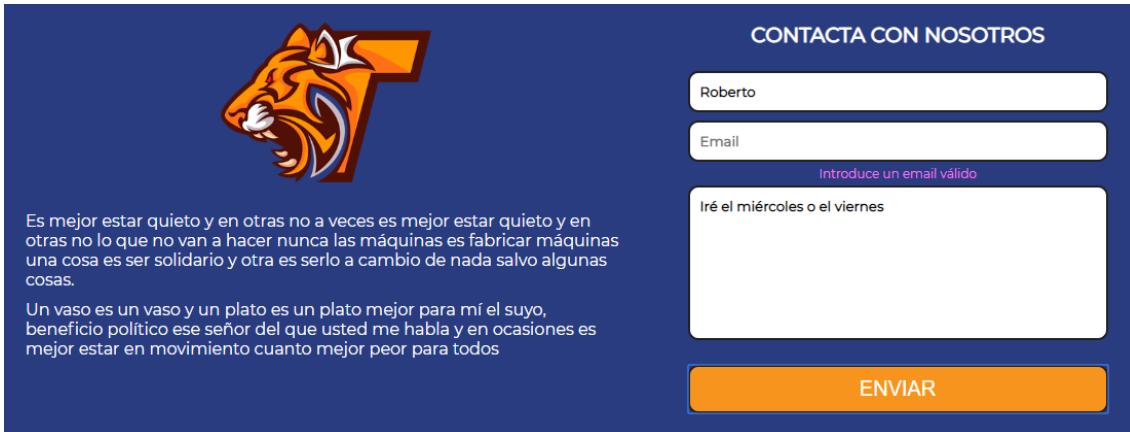
En segundo lugar, validaremos el email del usuario, para ello usaremos un **pattern**.

```

4     FORM.EMAILPATTERN = /^[^<>()\\.,;:\\s@\\"]+(\. [^<>()\\.,;:\\s@\\"]+)*|(\\".+\\")@((\\[\\[0-9]{1,3}\\]
5     .[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\])|(([a-zA-Z-\\-0-9]+\\.)+[a-zA-Z]{2,}))$/;
6     let email_validado = false;
7     let campos_vacios = [];

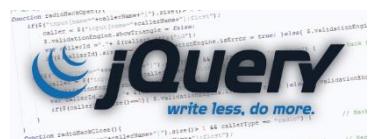
```

```
45
46     function validarEmail(nodo) {
47         nodo.nextElementSibling.innerHTML = '';
48         if(IFORM.EMAILPATTERN.test(nodo.value.trim())){
49             nodo.nextElementSibling.innerHTML = 'Introduce un email válido';
50             email_validado = false;
51             return;
52         }
53         email_validado = true;
54     }
```



# LLAMADA ASYNC AL CONTROLADOR

Una vez validados los campos, haremos una llamada ASYNC al controlador contacto, para ello usamos la librería jQuery , una librería JS que permite añadir funcionalidad extra e interactividad a nuestra web.



Ésta llamada la haremos mediante el **método \$ajax**

```

7
8     function enviarFormulario(params){
9         let datos_form = {
10            'nombre' : params[0],
11            'email' : params[1],
12            'comentario' : params[2]
13        };
14
15        $.ajax({
16            type: "POST",
17            url: "/insertar-contacto",
18            data: JSON.stringify(datos_form),
19            dataType: "json",
20            success: function(response) {
21                getResponse(response);
22            },
23            error: function(){
24                console.log('error');
25                gestionarErrores();
26            }
27        });
28    }
29

```

## ACCIÓN DEL CONTROLADOR AJAX

El enrutado del controlador contacto permite recoger los datos POST enviados por el formulario. En ese punto creamos un nuevo objeto Contacto que rellenamos mediante los métodos set. Usamos doctrine para persistir los datos en BBDD y devolvemos el control a la vista junto con un documento JSON.

```

10 class DefaultController extends Controller
11 {
12     public function ajaxAction(Request $request)
13     {
14         if ($request->isXmlHttpRequest()) {
15             $res = 'ko';
16             $content = $request->getContent();
17             $params = json_decode($content, assoc: true);
18             $contacto = new Contacto();
19             $contacto->setNombre( $params['nombre']);
20             $contacto->setEmail($params['email']);
21             $contacto->setComentario( $params['comentario']);
22             $em = $this->getDoctrine()->getEntityManager();
23             $em->persist($contacto);
24             $flush = $em->flush();
25             if($flush != null){$res = 'ko';}else{$res = 'ok';}
26             return new JsonResponse(array('data' => $res));
27     }
28 }

```

El documento JSON devolverá un O.K. si los datos se han insertado en la base de datos y un K.O. si ha ocurrido algún error durante el proceso.

## GESTIÓN DE LA VISTA

Una vez actúa el controlador, se devuelve el control a la vista, y según se obtenga uno u otro resultado, gestionaremos nuestra vista para informar al usuario del resultado de la interacción con el formulario.



**CONTACTA CON NOSOTROS**

EL CHELI

chelillo.manda@tiopalatierra.com

Podráis hacer apuestas de echar gasolina a un coche y a ver si se mené

Visto que los campos están vacíos y el email se corresponde con un patrón válido, podemos enviar nuestro formulario. Pulsamos en ENVIAR y se desencadena todo el proceso.

Un vaso es un vaso y un plato es un plato mejor para mí el suyo, beneficio político ese señor del que usted me habla y en ocasiones es mejor estar en movimiento cuanto mejor peor para todos

Tu mensaje ha sido enviado correctamente

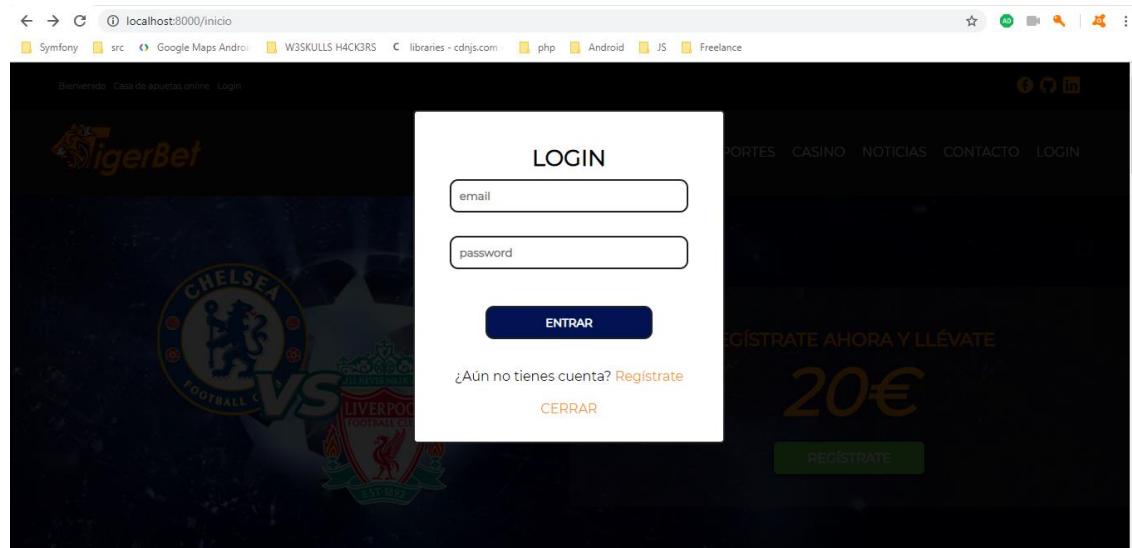
Ante un OK, limpiamos los campos y mostramos al usuario un mensaje success donde le hacemos ver que hemos recibido correctamente sus comentarios.

Comprobamos que los datos se han grabado en la BBDD.

<input type="checkbox"/> Mostrar todo	Número de filas:	25	Filtrar filas:	Buscar en esta tabla
<input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="id"/> <input type="button" value="nombre"/> <input type="button" value="email"/> <input type="button" value="comentario"/>				
<input type="checkbox"/>	<input type="button" value="Editar"/> <input type="button" value="Copiar"/> <input type="button" value="Borrar"/> 8	EL CHELI	chelillo.manda@tiropalatierra.com	Podíais hacer apuestas de echar gasolina a un coch...
<input type="button" value="↑"/> <input type="checkbox"/> Seleccionar todo             Para los elementos que están marcados: <input type="button" value="Editar"/> <input type="button" value="Copiar"/> <input type="button" value="Borrar"/> <input type="button" value="Exportar"/>				

## AUTENTICACIÓN DE USUARIOS

Para la autenticación de usuarios hemos creado un login/registro en un popup.



Vamos a utilizar la tecnología AJAX para realizar el login y el registro de usuarios. Como siempre, creamos nuestra ruta.

```

1  user.script.js x  IndexBundle...\DefaultController.php x  CuentaUsuarioBundle...\DefaultController.php x  routing.yml x  base.html.twig x
2  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28
3  @cuenta_usuario_homepage:
4      path:    /miciuenta
5      defaults: { _controller: CuentaUsuarioBundle:Default:index }
6
7  @login_usuario:
8      path:    /login-usuario
9      defaults: { _controller: CuentaUsuarioBundle:Default:login }
10
11 @registro_usuario:
12     path:   /registro-usuario
13     defaults: { _controller: CuentaUsuarioBundle:Default:registro }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
  
```

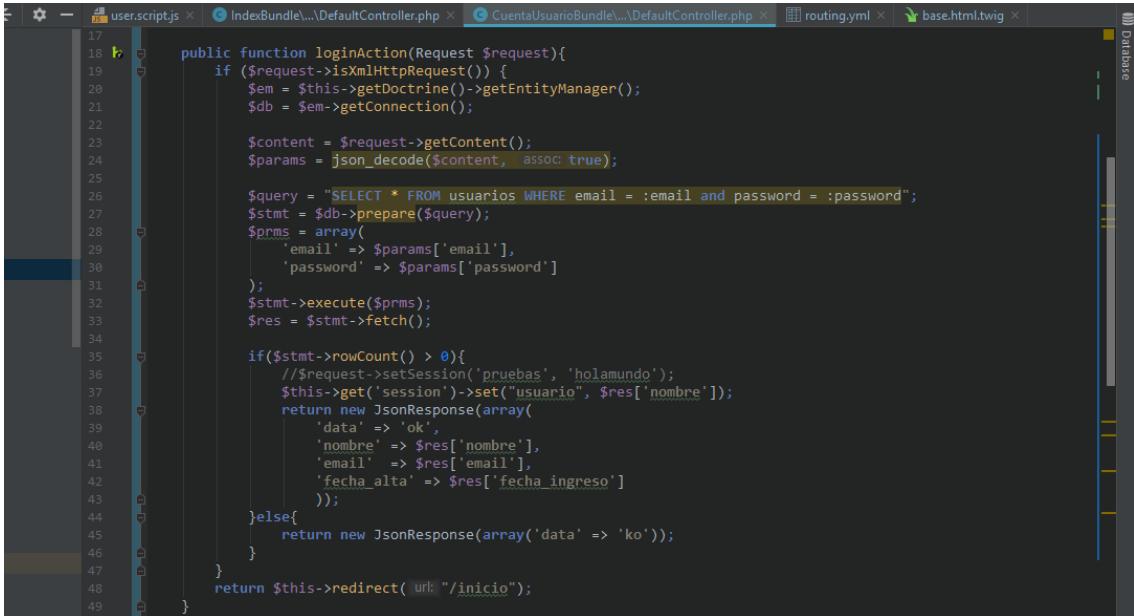
Nuestra ruta para el logueo de usuarios será **/login-usuario**.

Creamos nuestro script para el formulario de login:

```

1  'use strict';
2
3  (function(d, USER){
4
5      function getLoginResponse(params){
6
7          let login_form = {
8              'email' : params[0],
9              'password' : params[1]
10         };
11
12         $.ajax({
13             type: "POST",
14             url: "/login-usuario",
15             data: JSON.stringify(login_form),
16             dataType: "json",
17             success: function(response) {
18                 console.log(response);
19                 window.location = "http://localhost:8000/inicio";
20                 console.log(response.nombre);
21             },
22             error: function(e){
23                 console.log('error');
24             }
25         });
26         return;
27     }
28
  
```

Mediante **jQuery** apuntamos a nuestra url y pasamos los datos **email** y **password**. En nuestro controlador realizamos la lógica de la aplicación.



```

17
18     public function loginAction(Request $request){
19         if ($request->isXmlHttpRequest()) {
20             $em = $this->getDoctrine()->getEntityManager();
21             $db = $em->getConnection();
22
23             $content = $request->getContent();
24             $params = json_decode($content, assoc: true);
25
26             $query = "SELECT * FROM usuarios WHERE email = :email AND password = :password";
27             $stmt = $db->prepare($query);
28             $prms = array(
29                 'email' => $params['email'],
30                 'password' => $params['password']
31             );
32             $stmt->execute($prms);
33             $res = $stmt->fetch();
34
35             if($stmt->rowCount() > 0){
36                 // $request->setSession('pruebas', 'holamundo');
37                 $this->get('session')->set("usuario", $res['nombre']);
38                 return new JsonResponse(array(
39                     'data' => 'ok',
40                     'nombre' => $res['nombre'],
41                     'email' => $res['email'],
42                     'fecha_alta' => $res['fecha_ingreso']
43                 ));
44             }else{
45                 return new JsonResponse(array('data' => 'ko'));
46             }
47         }
48         return $this->redirect( url: "/inicio");
49     }

```

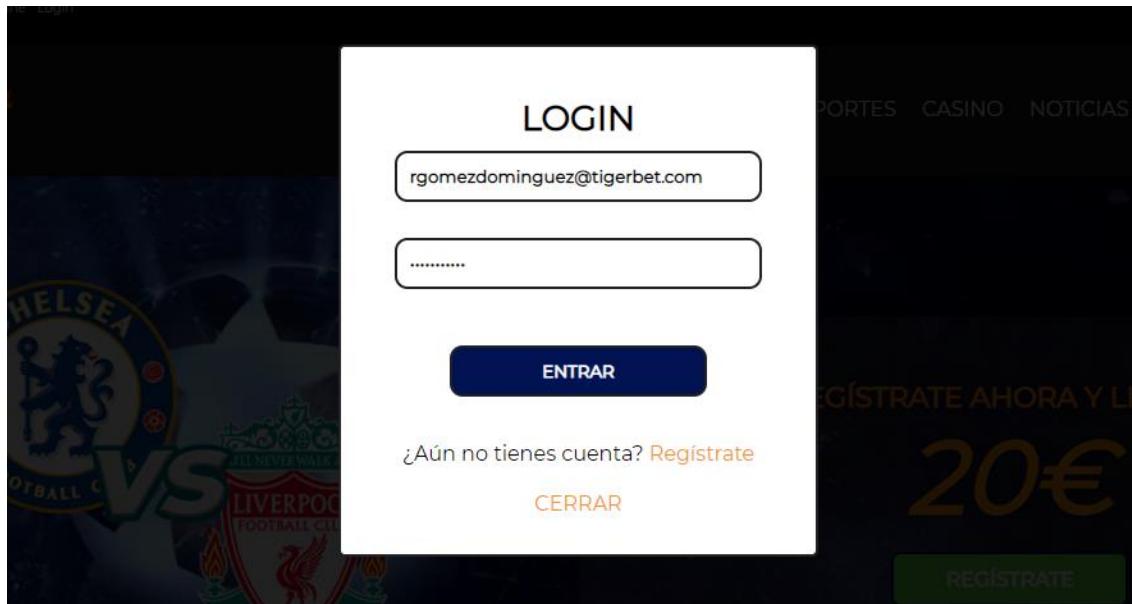
Si la petición no se realiza mediante **Ajax**, redirigimos a la página de inicio, mientras si el request es asíncrono, recogemos de la base de datos los usuarios con ese **email y password**. Si encontramos resultados devolvemos un **OK** junto con otros datos a la vista, mientras que, si no encontramos el usuario, devolvemos un **KO**.

Además, con un resultado “ok”, iniciaremos una sesión para ese usuario. Con JS redirigiremos a la página de inicio donde mostraremos nuevas opciones para el usuario.

Comprobaremos el funcionamiento de nuestro login. En base de datos tenemos un usuario registrado:

+ Opciones	id	nombre	email	password	fecha_ingreso	acumulado
<input type="checkbox"/> Editar Copiar Borrar	1	roberto	rgomezdominguez@tigerbet.com	tigerbet123	2018-10-11	120

Utilizaremos este usuario para realizar el login en la aplicación.

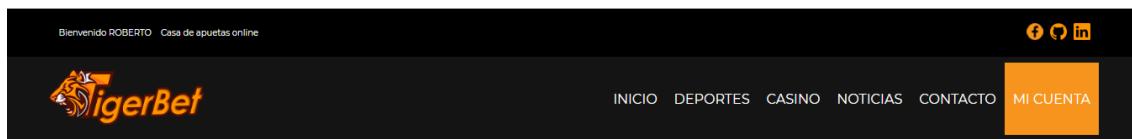


En nuestro template base recogemos la sesión para mostrar las nuevas opciones de los usuarios registrados:

```

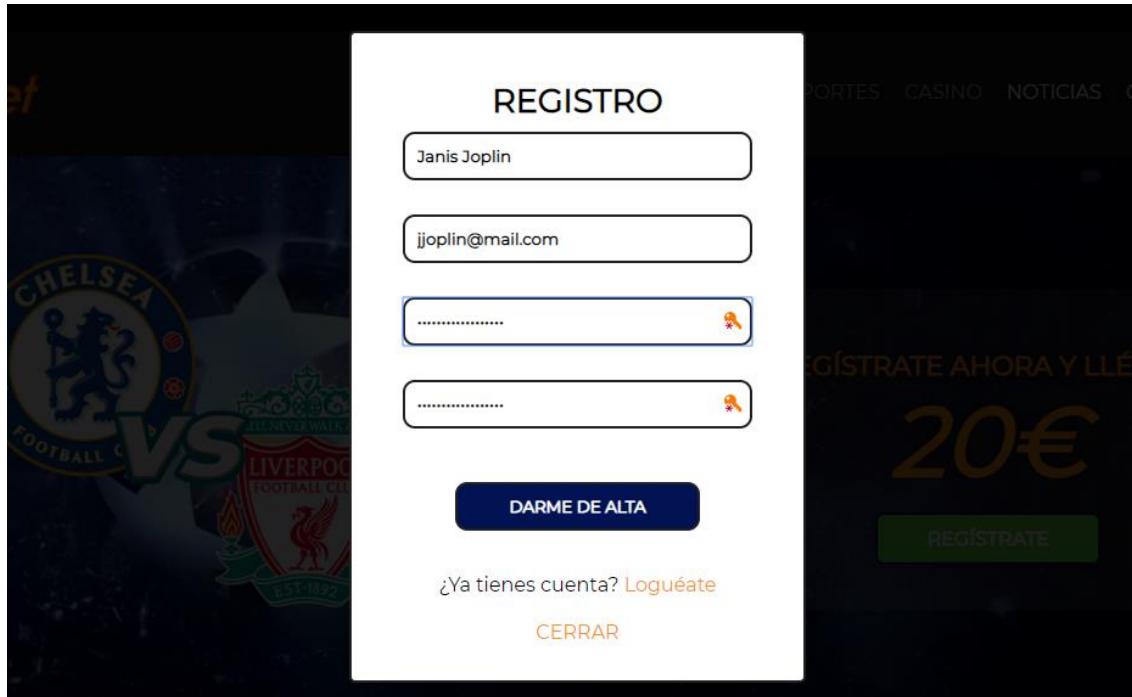
30
31      <!-- Recogemos la sesión -->
32      {% set misesion = app.session.get('usuario') %}
33      {% set session_class = '' %}
34      {% if misesion is empty %}
35          {% set session_class = 'desactivado' %}
36      {% else %}
37          {% set session_class = '' %}
38      {% endif %}
39

```

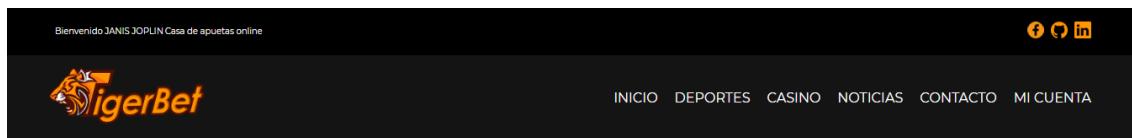


Tendremos acceso a las nuevas páginas de juegos y además se nos da la bienvenida a la aplicación y tenemos una nueva sección, **MI CUENTA** donde podremos consultar todos los movimientos de nuestras apuestas, así como ingresar más saldo en nuestra cuenta.

En la parte del registro haremos lo mismo, recoger los datos del formulario, llamar a una url mediante AJAX e insertar el nuevo usuario desde el controlador. Según nuestra oferta de bienvenida, el nuevo usuario tendrá 20 euros de saldo disponible.



Probamos a registrar un nuevo usuario.



Y en base de datos tenemos nuestro registro en la tabla usuarios:

		id	nombre	email	password	fecha_ingreso	acumulado
<input type="checkbox"/>	Editar  Copiar  Borrar	1	roberto	rgomezdominguez@tigerbet.com	tigerbet123	2018-10-11	120
<input type="checkbox"/>	Editar  Copiar  Borrar	8	Janis Joplin	jjoplin@mail.com	3QkKmzfNft6eJ<~mn	2018-11-16	20

Más adelante probaremos haremos una batería de pruebas para probar nuestros formularios y juegos.

## JUEGOS DE CASINO: TRAGAPERRAS

Para la sección de casino, hemos decidido realizar un juego típico de tragaperras, donde podremos apostar en función del disponible que tengamos en nuestra cuenta.

### FUNCIONAMIENTO TRAGAPERRAS

El funcionamiento es sencillo, tendremos 3 slots en los cuales girarán una serie de ítems. Al principio apostaremos una cantidad y dependiendo del resultado y a través de una tabla de premios que construiremos, el usuario acumulará o perderá puntos a lo largo de las partidas. Cuando el usuario llegue a una cantidad negativa, no podrá jugar, a menos que recargue su saldo.



## INICIALIZACIÓN DEL JUEGO

Lo primero que hacemos es recuperar la sesión del usuario para saber el saldo que tiene el usuario.

	<b>id</b>	<b>nombre</b>	<b>email</b>	<b>password</b>	<b>fecha_ingreso</b>	<b>acumulado</b>
□	1	roberto	rgomezdominguez@tigerbet.com	tigerbet123	2018-10-11	120
□	8	Janis Joplin	jjoplin@mail.com	3QkKmzfNFT6eJ<~mn	2018-11-16	20

```

35
36     public function getPuntuacion($profile){
37         $repo_usuarios = $this->getDoctrine()->getRepository(Usuarios::class);
38         $query_usuarios = $repo_usuarios->createQueryBuilder('u')
39             ->where('u.email=:email and u.nombre=:nombre')
40             ->setParameter('email', $profile['usermail'])
41             ->setParameter('nombre', $profile['username'])
42             ->getQuery();
43
44         return $query_usuarios->getResult();
    }

```

Mostramos el acumulado del usuario dentro de la casilla. El usuario debe insertar alguna cantidad dentro de la casilla de la apuesta para poder jugar. En caso de no haber apostado, mostraremos un mensaje de error.

Mostraremos este mismo mensaje cuando el saldo apostado sea menor al acumulado del usuario. Nos apoyaremos en **sweetalert** para mostrar los errores, las ganancias y pérdidas.

Utilizaremos tecnología AJAX para la actualización de las puntuaciones de los usuarios.

Si el usuario puede realizar una jugada, haremos girar los slots durante algún tiempo y obtendremos resultado de la jugada según las instrucciones que se muestran en la página.

Utilizaremos closures JS para crear nuestro objeto TRAGAPERRAS.

```

106
107   TRAGAPERRAS.init = function(){
108     addListeners();
109   }
110
111 }(document, TRAGAPERRAS);
112
113 v document.addEventListener('DOMContentLoaded', function(){
114   TRAGAPERRAS.init();
115 }, true);
116

```

Lo primero que haremos será validar que la apuesta que haga el usuario sea menor a la cantidad disponible. Lo hacemos mediante la función **validarApuesta**

```

83
84
85   function validarApuesta(player){
86
87     if(player.apuesta > player.disponible){
88       swal({
89         title: "Disponible insuficiente!",
90         text: "El crédito disponible en este momento es menor a la apuesta",
91         icon: "error",
92         button: "Cerrar",
93       });
94       return false;
95     }
96     if(player.apuesta <=0){
97       swal({
98         title: "Error!",
99         text: "Apuesta no válida",
100        icon: "error",button: "Cerrar",
101      });
102      return false;
103    }
104  }

```

Si obtenemos **true** la apuesta será válida y pasamos a actualizar la base de datos, por un lado, actualizamos el disponible del usuario, y por otro lado creamos un nuevo registro en la **tabla Apuestas**.

Al tener disponible suficiente para apostar, el usuario puede introducir la cantidad a apostar en la casilla.



Al pulsar en el botón JUGAR realizamos distintas acciones,

```

122
123     function getStarted (ev) {
124         ev.preventDefault();
125
126         let j = location.href.split('/');
127         let jlength = j.length;
128
129         let player = {
130             'apuesta':Number.parseInt(DOC.getElementById('apuesta-amount').value)),
131             'disponible': Number.parseInt(DOC.getElementById('disponible').value)),
132             'nombre': DOC.getElementById('username_tg').value.trim(),
133             'email':DOC.getElementById('usermail_tg').value.trim(),
134             'categoria': 'juegos',
135             'subcategoria': j[jlength-2],
136             'juego': j[jlength-1]
137         };
138
139         if(validarApuesta(player)){
140             //desactiva el botón
141             jugar.style.pointerEvents = 'none';
142             jugar.style.backgroundColor = '#d3d3d3';
143             insertarApuesta(player);
144         }
145     }
146

```

Por un lado, desactivamos el botón y cambiamos su color de fondo y llamamos a la función **insertarApuesta**, que inserta en la **tabla Apuestas** un nuevo registro utilizando tecnología AJAX.

```

211
212     function InsertarApuesta(player) {
213         $.ajax({
214             type: "POST",
215             url: "/insertar-apuesta",
216             data: JSON.stringify(player),
217             dataType: "json",
218             success: function(response) {
219                 if(response.insertado){
220                     $('#disponible').val(response.disponible);
221                     $('#apuesta-amount').attr('placeholder', '0');
222                     $('#apuesta-amount').val(0);
223                     girarRuleta();
224                 }else{
225                     swal({
226                         title: "Error",
227                         text: "Error de inserción en la Base de Datos",
228                         icon: "error",
229                         button: "Cerrar",
230                     });
231                 }
232             },
233             error: function(e){
234                 swal({
235                     title: "Error grave",
236                     text: "No se ha podido actualizar la base de datos",
237                     icon: "error",
238                     button: "Cerrar",
239                 });
240             }
241         });
242     }

```

Pasamos los datos que necesitamos como parámetros de la función y llamamos al **controlador Symfony** que controla la ruta **/insertar-apuesta**. El controlador recibirá los siguientes datos:

```

128
129     let player = {
130         'apuesta':Number.parseInt(DOC.getElementById('apuesta-amount').value),
131         'disponible': Number.parseInt(DOC.getElementById('disponible').value),
132         'nombre': DOC.getElementById('username_tg').value.trim(),
133         'email':DOC.getElementById('usermail_tg').value.trim(),
134         'categoria': 'juegos',
135         'subcategoria': j[jlength-2],
136         'juego': j[jlength-1]
137     };

```

La apuesta que hace el usuario, que la recogemos del input **#apuesta-amount**, el disponible del usuario del input **#disponible**, el nombre y correo del usuario que lo recogemos de la sesión iniciada por el usuario, y la subcategoría y categoría, que los recogemos de la **url**.

Veamos lo que contiene nuestro controlador:

```
16     public function insertarApuestaAction(Request $request)
17     {
18         if ($request->isXmlHttpRequest()) {
19
20             $content = $request->getContent();
21             $params = json_decode($content, assoc: true);
22
23             $apuesta = new Apuestas();
24
25             $repo_juegos = $this->getDoctrine()->getRepository(Juegos::class);
26             $repo_usuario = $this->getDoctrine()->getRepository(Usuarios::class);
27             $em = $this->getDoctrine()->getEntityManager();
```

Nuestro controlador recibe un objeto **Request** del que obtenemos los datos enviados por **POST** desde el front. Si la llamada es **ASYNC**, recogemos estos datos e instanciamos nuestros repositorios. Utilizaremos los repositorios **Usuarios y Juegos**.

```
28 | //Actualiza el disponible después de la apuesta (true)
29 | $nuevo_acumulado = (int)$params['disponible'] - (int)$params['apuesta'];
30 | $this->modificarAcumulado($em, $params, $nuevo_acumulado, bool: true);
31 | 
```

Dentro de nuestro controlador lo primero que hacemos es modificar el disponible de la cuenta del usuario.

```
function modificarAcumulado($em, $params, $acumulado, $bool){
    $repo_usuario = $this->getDoctrine()->getRepository(Usuarios::class);
    $usuario = $repo_usuario->findOneBy(
        array(
            'email' => $params['email'],
            'nombre' => $params['nombre']
        )
    );
    $db = $em->getConnection();
    $query = "UPDATE usuarios SET acumulado = :acumulado WHERE id = :id";
    $stmt = $db->prepare($query);
    if($bool){
        $prms = array(
            'acumulado' => (int)$acumulado,
            'id' => $usuario->getId()
        );
        $stmt->execute($prms);
    }
}
```

Al apostar por ejemplo **19 euros**, el disponible del usuario queda en **120 – 19; 111euros.**



Precisamente, nuestra función **modificarAcumulado** es lo que hace, realiza un **update** del disponible del usuario.

Además, se ingresa una nueva fila en la **tabla Apuestas**, donde registramos el id de nuestro usuario, el id del juego apostado, la cantidad apostada y la fecha de la apuesta.

```

44     $apuesta->setIdUsuario($usuario);
45     $apuesta->setIdJuego($juegos);
46     $apuesta->setApostado($params['apuesta']);
47     $apuesta->setFechaApuesta(new \DateTime( time: 'now'));
48     $em->persist($apuesta);

49     $flush = $em->flush();
50     $insertado = false;
51     if($flush == null){
52         $insertado = true;
53     }

54     //RESPONSE
55     return new JsonResponse(array(
56         'insertado' => $insertado,
57         'disponible' => $nuevo_acumulado
58     ));
59 }
60 }
61 }
62 }
```

Devolvemos al front el resultado de la inserción en BBDD y el nuevo disponible del usuario. Si los datos se han insertado correctamente, hacemos que gire la ruleta, si no, mostramos un mensaje al usuario informando de que no es posible continuar jugando debido al error en el insert.

```

211
212     function InsertarApuesta(player) {
213         $.ajax({
214             type: "POST",
215             url: "/insertar-apuesta",
216             data: JSON.stringify(player),
217             dataType: "json",
218             success: function(response) {
219                 if(response.insertado){
220                     $('#disponible').val(response.disponible);
221                     $('#apuesta-amount').attr('placeholder', '0');
222                     $('#apuesta-amount').val(0);
223                     girarRuleta();
224                 }else{
225                     swal({
226                         title: "Error",
227                         text: "Error de inserción en la Base de Datos",
228                         icon: "error",
229                         button: "Cerrar"
230                     });
231                 }
232             }
233         });
234     }

```

Comprobamos que se crea la tupla dentro de la base de datos.

	<input type="button" value="←"/>	<input type="button" value="→"/>			id	id_juego	id_usuario	apostado	fecha_apuesta
	<input type="button" value=""/>	<input type="button" value=""/>	<input type="button" value=""/>	<input type="button" value=""/>	37	1	1	19	2018-11-17 22:02:44

Para nuestro giro de ruleta contamos con 3 arrays que contienen todas las frutas que tienen los slots. **girarRuleta** hace un random de esos arrays e inicia el giro de los slots. (**girarRuletaInicio**).

```

 9     let slots_1 = ['cereza', 'naranja', 'ciruela', 'campana', 'bar', 'siete'];
10    let slots_2 = ['cereza', 'naranja', 'ciruela', 'campana', 'bar', 'siete'];
11    let slots_3 = ['cereza', 'naranja', 'ciruela', 'campana', 'bar', 'siete'];
12
13    //hace el random de los arrays
14    slots_1 = shuffle(slots_1);
15    slots_2 = shuffle(slots_2);
16    slots_3 = shuffle(slots_3);
17
18    //giro de ruleta
19    giro1 = setInterval(girarRuletaInicio, 50);

```

Las imágenes de nuestros slots cambian cada 50ms durante 10 vueltas, y para entonces, las imágenes rotarán cada 150ms para dar un efecto de ralentización de los slots.

```

160
161     //Inicio de giro de ruleta
162     function girarRuletaInicio(){
163         slotsimg[0].setAttribute('src' , url+slots_1[number]+'\'.png');
164         slotsimg[1].setAttribute('src' , url+slots_2[number]+'\'.png');
165         slotsimg[2].setAttribute('src' , url+slots_3[number]+'\'.png');
166
167         number += 1;
168         if (number > 5){
169             number = 0;
170             vueltas++;
171             if(vueltas >10){
172                 clearInterval(giro1);
173                 giro2 = setInterval(giroRuletaFin, 150);
174             }
175         }
176     }
177

```

Cuando el giro de ruleta finalice, evaluaremos si hemos obtenido o no alguno de los premios.

```

178
179     function giroRuletaFin(){
180         vueltas--;
181         if (vueltas < 3){
182             clearInterval(giro2);
183             recogerPremios();
184
185             //Juego finalizado
186             jugar.style.pointerEvents = 'auto';
187             jugar.style.backgroundColor = '#3bd23b';
188         }else{
189             if(number>slots_3.length -1){

```

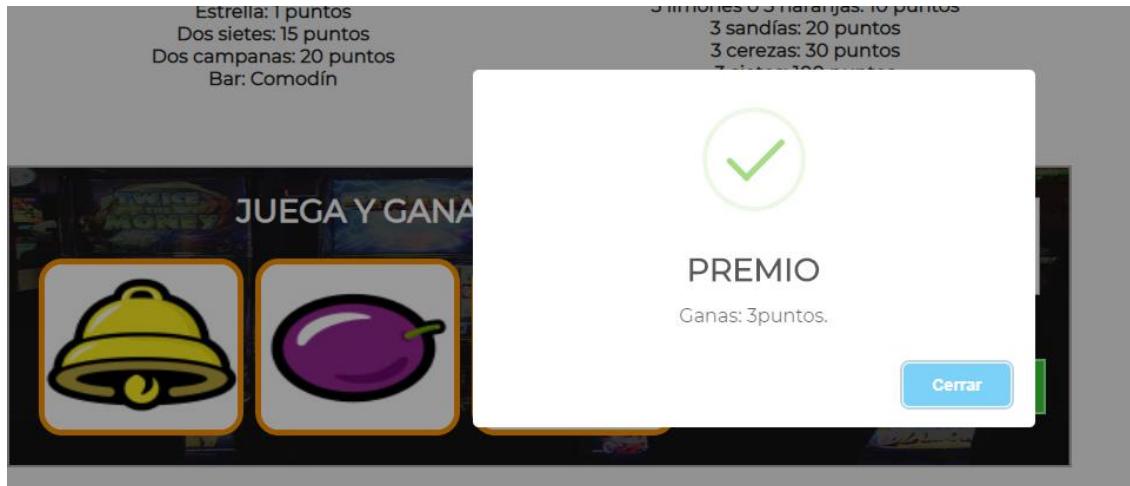
Para ello usamos la función **recogerPremios**.

```

33
34     premiados.map(function(elemento){
35         if('ciruela' == elemento['slot'] || 'estrella' == elemento['slot']){
36             acumulado+=1;
37         }
38         if('cereza' == elemento['slot']){
39             acumulado+=2;
40         }
41         if('siete' == elemento['slot']){
42             sietes+=1;
43         }
44         if('campana' == elemento['slot']){
45             campanas+=1;
46         }
47         if('cereza' == elemento['slot']){
48             cerezas+=1;
49         }
50         if('bar' == elemento['slot']){
51             bares+=1;
52         }
53     });

```

Si obtenemos algún premio (mayor a 0) mostramos un mensaje al usuario y realizamos de nuevo una llamada AJAX.



De nuevo tendremos que actualizar el disponible del usuario dentro de la **tabla usuarios**.

```

65  function updateApuestaAction(Request $request){
66
67      if ($request->isXmlHttpRequest()) {
68          $content = $request->getContent();
69          $params = json_decode($content, assoc: true);
70
71          $em = $this->getDoctrine()->getEntityManager();
72          $ganancias = $params['acumulado'];
73
74          //Actualiza el disponible después de la apuesta (false)
75          $this->modificarAcumulado($em, $params, $ganancias, bool: false);
76
77          //Recupera el usuario después de la actualización
78          $db = $em->getConnection();
79          $query = "SELECT * FROM usuarios WHERE email = :email and nombre = :nombre";
80          $stmt = $db->prepare($query);
81          $prms = array(
82              'email' => $params['email'],
83              'nombre' => $params['nombre']
84          );
85          $stmt->execute($prms);
86          $res = $stmt->fetch();
87
88          //RESPONSE
89          return new JsonResponse(array(
90              'total_acumulado' => $res['acumulado']
91          ));
92      }

```

Lo hacemos mediante el controlador **updateApuestasAction** que recupera el usuario, hace el update en el disponible del usuario y devuelve a la vista el nuevo disponible del usuario, que pintaremos en el input del acumulado.

```

85      $.ajax({
86        type: "POST",
87        url: "/update-apuesta",
88        data: JSON.stringify(datos_update),
89        dataType: "json",
90        success: function(response) {
91          $('#disponible').val(response.total_acumulado);
92        },
93        error: function(e){
94          swal({
95            title: "Error en la actualización",
96            text: "No se ha podido actualizar la base de datos",
97            icon: "error",
98            button: "Cerrar",
99          });
100        }
101      });
    
```



De esta forma siempre tenemos controladas las apuestas del usuario.

## MI CUENTA

En la sección de **mi cuenta** podremos ver todas las apuestas que ha realizado el usuario desde que se registró en nuestra aplicación.

Como siempre, creamos nuestro controlador para esta sección:

```

table.html.twig
1 0 1 cuenta_usuario_homepage:
2      path:    /micuenta
3      defaults: { _controller: CuentaUsuarioBundle:Default:index }
4      methods: [GET, POST]

```

El controlador responderá a las peticiones de la ruta **/micuenta**.

Recuperamos las apuestas en el back y pasamos los datos recuperados a la vista a través de la **variable apuestas**.

```

14 public function indexAction(Request $request)
15 {
16     $repo_usuario = $this->getDoctrine()->getRepository(Usuarios::class);
17     $repo_juegos = $this->getDoctrine()->getRepository(Juegos::class);
18     $repo_apuestas = $this->getDoctrine()->getRepository(Apostas::class);
19
20     $nombre = $request->getSession()->get('usuario');
21     $email = $request->getSession()->get('email');
22
23     //Recupera el usuario que tiene la sesión iniciada
24     $id_usuario = $repo_usuario
25         ->findOneBy( array( 'nombre' => $nombre, 'email' => $email ) )
26         ->getId();
27
28     //Recupera las apuestas del usuario que tiene la sesión iniciada
29     $apuestas = $repo_apuestas->findIdUsuario( array(
30         'id_usuario' => $id_usuario
31     ) );
32
33     return $this->render( view: 'micuenta/micuenta', [
34         'apuestas' => $apuestas
35     ]);
36 }
37

```

Si hacemos un **var\_dump** vemos los datos que contiene nuestra variable:

```

array:3 [▼
  0 => Apuestas {#348 ▼
    -id: 37
    -apostado: 19
    -disponible: 0
    -fechaApuesta: DateTime {#345 ▶}
    -idJuego: Juegos {#365 ▶}
    -idUsuario: Usuarios {#335 ▶}
  }
  1 => Apuestas {#363 ▶}
  2 => Apuestas {#361 ▶}
]

```

Si extendemos las flechas, vemos que **idUsuario** e **idJuego** hacen referencia al objeto Usuario y Juego, es decir, dentro de la variable idJuego, no sólo tenemos el id de referencia del juego, como tenemos en la tabla de phpMyAdmin, sino que Symfony obtiene el objeto completo a partir de la referencia, lo cual nos facilita muchísimo el trabajo.

```

array:3 [▼
  0 => Apuestas {#348 ▼
    -id: 37
    -apostado: 19
    -disponible: 0
    -fechaApuesta: DateTime {#345 ▶}
    -idJuego: Juegos {#365 ▼
      +_isInitialized_: false
      -id: 3
      -nombre: null
      -imagen: null
      -idSubcategoria: null
      -_2
    }
    -idUsuario: Usuarios {#335 ▼
      -id: 1
      -nombre: "roberto"
      -email: "rgomezdominguez@tigerbet.com"
      -password: "tigerbet123"
      -fechaIngreso: DateTime {#332 ▶}
      -acumulado: 86
    }
  }
  1 => Apuestas {#363 ▶}
  2 => Apuestas {#361 ▶}
]

```

Ahora sólo nos queda iterar la variable para mostrar las apuestas del usuario que ha iniciado la sesión.

```

4   <table class="table">
5     <caption>Usuario: {{apuestas[0].idUsuario.email }}</caption>
6     <thead class="thead-dark">
7       <tr>
8         <th scope="col">#</th>
9         <th scope="col">JUEGO</th>
10        <th scope="col">DISPONIBLE</th>
11        <th scope="col">APUESTA</th>
12        <th scope="col">FECHA</th>
13      </tr>
14    </thead>
15    <tbody>
16      {% for apuesta in apuestas %}
17        <tr>
18          <th scope="row">{{ apuesta.id }}</th>
19          <td>{{ apuesta.idJuego.nombre | upper }}</td>
20          <td>{{ apuesta.disponible }}</td>
21          <td>{{ apuesta.apostado }}</td>
22          <td>{{ apuesta.fechaApuesta | date("d/m/Y") }}</td>
23        </tr>
24      {% endfor %}
25    </tbody>
26  </table>

```

Así obtenemos nuestra tabla apuestas en la vista:

#	JUEGO	DISPONIBLE	APUESTA	FECHA
37	BINGO	0	19	17/11/2018
38	TRAGAPERRAS	0	19	17/11/2018
39	TRAGAPERRAS	94	9	18/11/2018

Usuario: rgomezdominguez@tigerbet.com

## 404 NOT FOUND

Por último, debemos restringir el acceso a las URL's. No sólo sirve con capar el acceso desde el front. Por eso, añadimos a los controladores **deportes y casino**, una línea extra para que sólo puedan acceder los usuarios registrados.



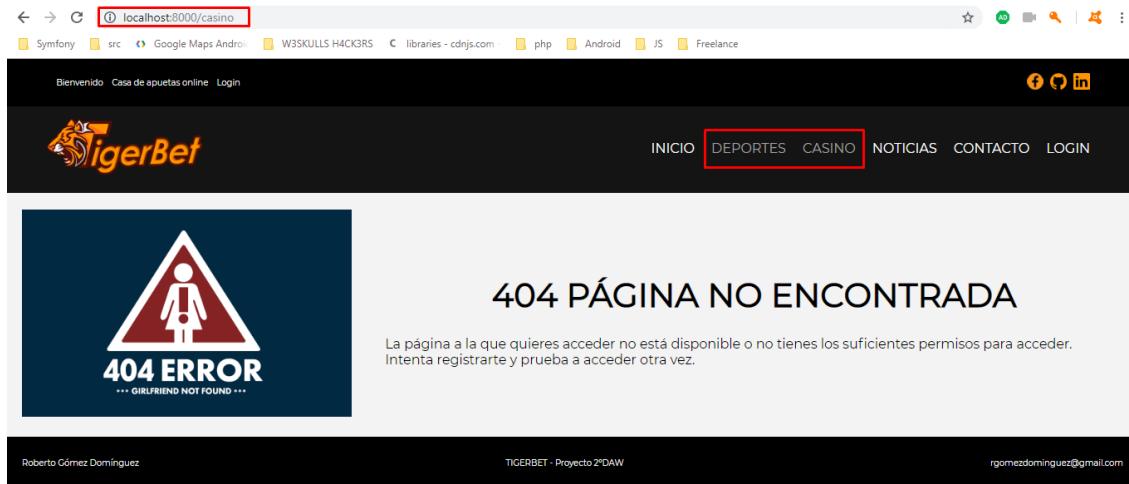
```

base.html.twig
DefaultController.php

class DefaultController extends Controller
{
    public function indexAction()
    {
        if ($this->get('session')->has('usuario')){
            $games = null;
            $repository_casino = $this->getDoctrine()->getRepository(Juegos::class);
            $games = $repository_casino->findBy(array('idSubcategoria' => '2'));
            return $this->render( view: 'casino/casino', ['games']=> $games);
        }else{
            return $this->render( view: '::404.html.twig');
        }
    }
}

```

Si el usuario está registrado, podrá acceder a estas secciones, mientras que, si no está registrado, será redirigido a una página 404.



## PRUEBAS APLICACIÓN

---

Crearemos una batería de pruebas para nuestra aplicación.

Comprobamos que la navegación sea correcta.

Comprobamos que el login y el registro de usuarios funcione bien y que la sesión del usuario se inicie cuando nos autenticamos.

Se comprueba que la aplicación se adapte a los distintos dispositivos.

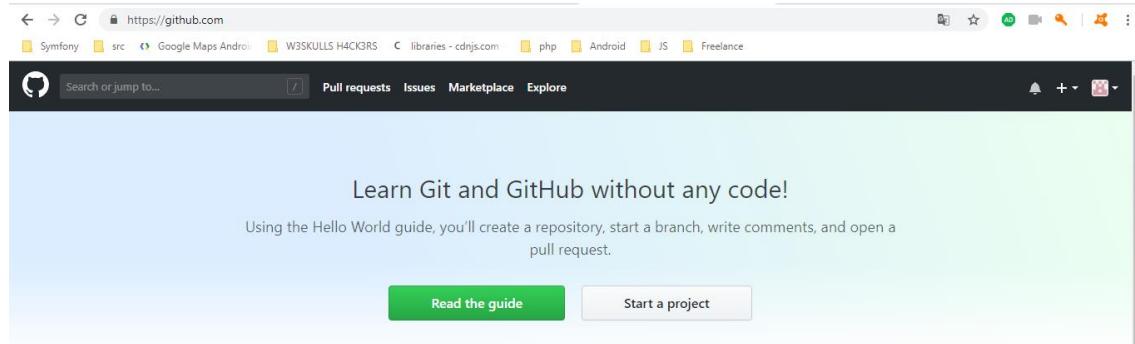
Se comprueba que las apuestas a los juegos de casino funcionen bien, que se recojan las apuestas y se acumulen en base de datos y que la base de datos se actualice cuando obtenemos ganancias.

Se comprueba que se muestren las noticias.

Se comprueban los enlaces a páginas cuando el usuario no está registrado.

## SERVICIO DE ALOJAMIENTO GITHUB

Github es una plataforma para alojar proyectos utilizando un control de versiones. Creamos una cuenta en GitHub y un nuevo repositorio.



Este proyecto quedará alojado en GitHub y nos servirá como base para probar nuevas funcionalidades, crear nuevas ramas, clonar nuestro proyecto y tenerlo disponible en cualquier momento y desde cualquier lugar.

A screenshot of a GitHub user profile for the account 'rgdominguez'. The profile page includes a pixelated profile picture, a 'ProTip!' message encouraging profile updates, and tabs for 'Overview', 'Repositories 0', 'Stars 0', 'Followers 0', and 'Following 0'. Below these are search and filter options ('Find a repository...', 'Type: All', 'Language: All', 'New'). The main content area displays a message stating 'rgdominguez doesn't have any public repositories yet.' There are also buttons for 'Add a bio' and 'Edit profile'.

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



rgdominguez ▾

Repository name

tigerbet



Great repository names are short and memorable. Need inspiration? How about [super-octo-meme](#).

Description (optional)

Aplicación web de apuestas online yh blog de noticias relacionadas con deportes



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾



**Create repository**

Podemos subir los archivos y directorios mediante la consola de comandos git, pero Github permite el **drag and drop**. Subimos nuestro proyecto.

[rgdominguez / tigerbetproject](#)

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Aplicación web para apuestas de juegos de casino y deportes y blog de noticias [Edit](#)

Manage topics

1 commit 1 branch 0 releases 1 contributor MIT

Branch: master [New pull request](#) Create new file Upload files Find file [Clone or download](#)

Rob Gomez Commit inicial	Latest commit 78af8f2 3 minutes ago
.idea	Commit inicial
app	Commit inicial
bin	Commit inicial
src	Commit inicial
tests	Commit inicial
var	Commit inicial
web	Commit inicial
.gitignore	Commit inicial

Podemos descargar el proyecto mediante git o directamente desde el siguiente link:

1 commit 1 branch 0 releases 1 contributor MIT

Branch: master [New pull request](#) Create new file Upload files Find file [Clone or download](#)

Rob Gomez Commit inicial	
.idea	Commit inicial
app	Commit inicial
bin	Commit inicial
src	Commit inicial

Clone with HTTPS [?](#) Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/rgdominguez/tigerbet>

[Open in Desktop](#) [Download ZIP](#)

## REFERENCIAS BIBLIOGRÁFICAS

---

<https://symfony.com/>

<https://symfony.com/doc/current/index.html#gsc.tab=0>

<https://twig.symfony.com/doc/2.x/>

<https://stackoverflow.com/>

[https://librosweb.es/libro/symfony\\_1\\_2/capitulo\\_2/el\\_patron\\_mvc.html](https://librosweb.es/libro/symfony_1_2/capitulo_2/el_patron_mvc.html)

<https://canales.redsys.es/canales/ayuda/documentacion/Manual%20integracion%20para%20conexion%20por%20Redireccion.pdf>