

Desarrollo web sistemático con CodeIgniter, RedBean y Bootstrap

INSTALACIÓN del ENTORNO

1. Descargar la última versión de **CodeIgniter** desde <https://www.codeigniter.com/>
2. Extraer el zip en una carpeta (p.ej. llamada “**ci**”)
3. Borrar la carpeta “**ci / user_guide**” (sólo es documentación de ayuda)
4. Descargar la última versión de **RedBeanPHP** desde <http://www.redbeanphp.com/index.php?p=/download>
5. Extraer el fichero **rb.php** y copiarlo en la carpeta “**ci / application / third_party / rb**” (habrá que crear la carpeta “rb” dentro de “third_party” probablemente)
6. Crear un fichero llamado “**ci / application / libraries / Rb.php**”, con el siguiente contenido (tal cual - no modificar)

```
<?php
class Rb {
    function __construct() {
        include(APPPATH.'./config/database.php');
        include(APPPATH.'./third_party/rb/rb.php');
        $host = $db[$active_group]['hostname'];
        $user = $db[$active_group]['username'];
        $pass = $db[$active_group]['password'];
        $db = $db[$active_group]['database'];
        R::setup("mysql:host=$host;dbname=$db", $user, $pass);
        R::setAutoResolve ( true );
    }
}
```

CONFIGURACIÓN BÁSICA del ENTORNO

1. Crear un fichero llamado “**ci / .htaccess**” con el siguiente contenido.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

2. Editar el fichero “**ci / application / config / database.php**”, y cambiar los valores del array asociativo \$db['default'] en sus claves 'hostname', 'username', 'password' y 'database' para que tomen los valores apropiados para nuestra Base de Datos.
3. Editar el fichero “**ci / application / config / autoload.php**”
 - Modificar las líneas apropiadas para que contengan
 - \$autoload['libraries'] = ['rb'];
 - \$autoload['helper'] = ['url'];
4. Editar el fichero “**ci / application / config / config.php**”
 - Modificar la línea apropiada para que contenga la ruta hasta la carpeta ci (incluida)
 - \$config['base_url'] = '<http://localhost/MiAplicacion>';
 - Asumimos que estamos trabajando en “localhost” y desplegaremos el **contenido** de “**ci**” bajo “**/ MiAplicacion**” o lo vincularemos con un soft-link

CREAR un HELPER de ENMARCADO

1. Crear un fichero llamado “ **ci / application / helpers / frame_helper.php** ”, con el siguiente contenido

```
function frame($controlador, $rutaVista, $datos = []) {
    if (session_status () == PHP_SESSION_NONE) {session_start ();}
    if (isset ( $_SESSION ['nombreUsuario'] )) {
        $datos ['_header'] ['usuario'] ['nombre'] = $_SESSION ['nombreUsuario'];
    }
    $controlador->load->view ( '_templates/head',$datos );
    $controlador->load->view ( '_templates/header', $datos );
    $controlador->load->view ( '_templates/nav', $datos );
    $controlador->load->view ( $rutaVista, $datos );
    $controlador->load->view ( '_templates/footer', $datos );
    $controlador->load->view ( '_templates/end' );
}
```

2. Editar el fichero “ **ci / application / config / autoload.php** ” e incluir este helper para tener la función “enmarcar” disponible en cualquier controlador.

```
$autoload['helper'] = ['url', 'frame']
```

3. Crear bajo el directorio “ **ci / application / views / _templates** ” los siguientes ficheros.
 - head.php (incluimos aquí las cabeceras de bootstrap 4 offline - para que funcione habrá que descargarse en la carpeta assets/js y assets/css los 4 ficheros de bootstrap4)

```
<!DOCTYPE html >
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="stylesheet" href="<?=base_url()>assets/css/bootstrap.min.css">

  <script src="<?=base_url()>assets/js/jquery-3.2.1.slim.min.js" ></script>
  <script src="<?=base_url()>assets/js/popper.min.js" ></script>
  <script src="<?=base_url()>assets/js/bootstrap.min.js"></script>

  <title>Mi título</title>
</head>
<body>
```

- header.php

```
<header class="container">
  AQUÍ IRÍA un ENCABEZADO que QUERAMOS que SALGA en todas nuestras PÁGINAS, quizá el usuario
  actualmente autenticado guardado en una sesión, es decir <?= $_header ['usuario'] ['nombre'] ?>
</header>
```

- nav.php

```
<nav class="container navbar navbar-expand-sm bg-dark navbar-dark rounded">

  <a class="navbar-brand" href="<?=base_url()>">
    
  </a>
```

```

        <ul class="navbar-nav">
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#">
                    MENU (CONT)
                </a>

                <div class="dropdown-menu">
                    <a class="dropdown-item"
href="<?=base_url()>cont/accion1">accion1</a>
                    <a class="dropdown-item"
href="<?=base_url()>cont/accion2">accion2</a>
                </div>
            </li>
        </ul>
    </nav>

```

- footer.php

```

<footer class="container">
    AQUÍ IRÍA un FOOTER que QUERAMOS que SALGA en todas nuestras PÁGINAS
</footer>

```

- end.php

```

</body>
</html>

```

4. Cada vez que deseemos enmarcar y desplegar una vista desde un controlador, procederemos de la siguiente manera.
 - “Empaquetaremos los datos que necesite cada parte de nuestro “marco”, muchos de ellos probablemente obtenidos de consultas a nuestros métodos de los modelos

```

$datos ['_header'] ['unDatoParaElHeader'] = ...;
$datos ['unDatoParaElBody'] = ...;
$datos ['unaColeccionDeDatosParaElBody'] = [ ..., ..., ... ] ;
$datos ['_footer'] ['unDatoParaElFooter'] = ...;

```

- Desplegamos la vista enmarcada, desde el controlador así:

```

frame ( $this, 'bean/vistaAccion1', $datos );

```

5. Hay veces que no queremos enmarcar la vista, p.ej. para vistas devueltas por acciones AJAX, que pueden venir en XML o bien en HTML que se “incrustarán” en un <div>. En ese caso desplegaremos nuestra vista, de la forma estándar en CodeIgniter

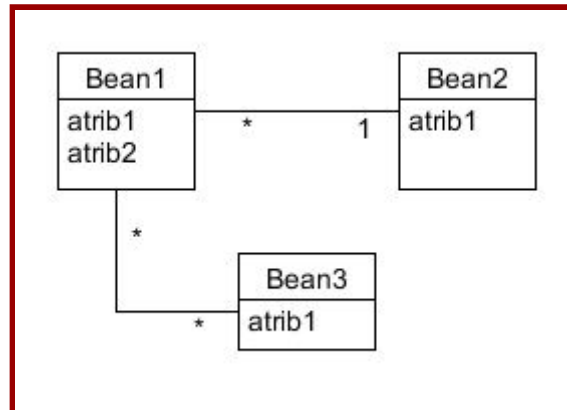
```

$this -> load -> view ( 'bean/vistaAccion1', $datos );

```

CONSTRUYENDO una ACCIÓN de un BEAN

1. Asumimos que tenemos un “**Modelo de dominio**”, en el que podemos ver todos los “**beans**” o “conceptos” que manejará nuestra aplicación con sus atributos definidos y las relaciones entre ellos. Utilizar por ejemplo, aplicaciones como [UMLet](#)



2. Asumimos que utilizaremos alguna **herramienta de planificación**, preferiblemente alguna online que implemente un método ágil como SCRUM, p.ej. [TAIGA](#), para decidir qué **acciones** deberemos implementar para cada BEAN (cada una será una historia de usuario), en qué orden las haremos y qué dificultad previsiblemente entraña cada una de ellas.

Asignaremos cada una de ellas a un miembro del equipo, y a un periodo de desarrollo (sprint) determinado. Dividiremos las historias de usuario en tareas.

Al menos habrá una titulada “DONE ...”, que significa que hasta que no se consigan todas las tareas DONE no se considerará que la Historia de Usuario está finalizada.

Los criterios DONE son los criterios impuestos por el cliente de nuestra aplicación.

El resto de tareas son procesos adicionales que deberemos desarrollar para conseguir las tareas DONE, p.ej. (“Instalar servidor Apache”, “Formación en jquery sobre Bootstrap”, etc.)

The screenshot shows a Taiga project dashboard with a dark header bar. The header displays a progress bar at 0%, 0 total points, 0 completed points, 3 open tasks, 0 closed tasks, and 0 local notifications. Below the header, there are four columns: HISTORIA DE USUARIO, NUEVA, EN CURSO, and CERRADA. The HISTORIA DE USUARIO column contains two user stories: #12 Acción1 en Bean1 (0 points) and #16 Acción 2 en Bean1 (0 points). The EN CURSO column contains two tasks: #13 DONE Hacer la acción1 en el Bean1, con éxito (assigned to Howard Wollowitz) and #14 DONE No poder hacer la acción 1 en el Bean si el atr1 es null (assigned to Penny Cuoco). The CERRADA column is empty. The NUEVA column contains one task: #15 DONE Realizar la acción 2 en el Bean 1 (assigned to Alberto Garay).

3. **CONTROLADOR:** Crearemos (en el caso de que no existiera ya) un fichero llamado “**ci / application / controllers / bean.php**”, donde “bean” es el nombre de un “bean” de nuestra aplicación (p.ej. “usuario”, “factura”, “ciudad”, etc.) para el que queramos desarrollar una determinada acción (p.ej. “crear”, “modificar”, etc.). El contenido de este fichero será del estilo:

```
class Bean extends CI_Controller {
    public function accion1 () {
        // Aquí escribiremos el código asociado a la acción1
    }
}
```

NOTA 1: A partir de ahora, podremos invocar la ejecución de esta acción desde cualquier vista de nuestra aplicación escribiendo la URL, p.ej. en el action de un form

```
<form action="<?= base_url() ?> bean/accion1" method="post">
```

NOTA 2: A partir de ahora, nuevas acciones para este mismo bean se materializarán como nuevos métodos de esta clase

4. El código genérico de la acción de un bean tiene el siguiente aspecto.

```
public function accion1 () {
    // 1. CARGAMOS y EJECUTAMOS algún método de un modelo, puede ser un MODELO del bean cuya acción
    // estamos definiendo, o podría ser un modelo de otro bean. En este ejemplo, asumimos que es de
    // este mismo bean
    $this -> load -> model ( 'bean_model' );
    $parametros = ... ; // Algunos parámetros que necesitará nuestra acción del modelo, extraídos
    // habitualmente de $_POST o $_SESSION
    $respuesta = $this -> bean_model -> accion_del_modelo ( $parametros );

    // 2. EMPAQUETAMOS, y/o transformamos, si es que no lo hemos hecho ya, los datos
    // provenientes del modelo
    $datos [ 'respuesta' ] = $respuesta ;

    // 3. DESPLEGAMOS, enmarcada o no enmarcada, la vista asociada a esta acción de este bean.
    // Normalmente habrá dos acciones y dos vistas asociadas a cada acción: “accionGET” y
    // “accionPOST”. Yo suelo llamar a la primera “accion” a secas, y por tanto utilizo la misma
    // nomenclatura para las vistas. Para que funcione el siguiente ejemplo debe existir el fichero
    // 'ci/application/views/bean/accion1.php'
    frame ( $this, 'bean/accion1', $datos );
}
```

5. **MODELO.** Creamos los modelos (si no lo hubiéramos hecho ya), y las acciones de modelo correspondientes. En el ejemplo anterior, crearíamos el fichero “**ci / application / models / bean_model.php**”, que contendría algo parecido a esto:

```
class Bean_model extends CI_Model {
    public function accion_de_modelo ( $parametros ) {
        // Aquí escribiremos el código asociado a la accion_de_modelo necesaria. Es aquí y sólo aquí
        // donde escribiremos SQL o código de RedBeanPHP o nuestro framework de persistencia
        // Nunca aludiremos desde aquí a $_GET, $_POST o $_SESSION. Si necesitamos algún dato de esos
        // nos lo debería haber pasado el controlador en los $parametros
        ...
        return $datos_para_el_controlador
    }
}
```

6. Ver apéndice “[Resumen de comandos RedBeanPHP](#)”

7. **VISTA.** Crear una vista para esta acción bajo la carpeta “ **ci / application / views** “. Es útil organizar las vistas por beans (creando una carpeta para todas las vistas de un bean), y dentro de ella creando un fichero **accion.php** para cada acción asociada a ese bean. Es muy posible que para una determinada acción, exista un fichero **accion.php** (para el despliegue GET) y otro **accionPost.php**. En este ejemplo crearemos un fichero llamado “ **ci / application / views / bean / accion1.php** “, que contendrá algo parecido a esto.

```
<div class="container">
  <p>
    Aquí habrá casi un 90% de HTML puro, con algunas “pinceladas” de php, por ejemplo
    si queremos mostrar el valor de un dato que nos empaquetó el controlador previamente.
  </p>
  <p>
    Por ejemplo, lo que empaquetó en ‘respuesta’ en el punto 4, valdría <?= $body [ 'respuesta' ] ?>
  </p>
  <p>
    También podría aparecer código javascript, o código de control PHP del estilo
  </p>
  <?php foreach ($body [ 'algunaColeccion' ] as $dato) : ?>
    Esta zona contendrá HTML que se escribirá en bucle en la página generada, y podremos aludir
    en cada vuelta, al valor que va tomando el dato, en este caso <?= $dato ?>
  <?php endforeach; ?>
</div>
```

8. Si se utiliza AJAX, recuérdese que:
- El controlador debe verificar las cabeceras, para comprobar que la petición es AJAX, y en caso contrario desplegar alguna vista de error.
 - El controlador invocará al modelo para hacer la acción AJAX correspondiente.
 - La vista desplegada contendrá datos estructurados: XML ó JSON, texto plano o HTML si queremos introducirlo dentro de un div.
 - En muchas ocasiones la vista tan sólo envía un código de status (o por ejemplo, la clase de un “span”) y un mensaje, para que el javascript correspondiente ponga el mensaje en su sitio apropiado con un color apropiado en función del status recibido
9. --

Resumen de comandos básicos de RedBeanPHP

Comando	Descripción
CONEXIÓN y PERSISTENCIA BÁSICA	
<code>\$bd = R::setup('mysql:host=localhost;dbname=bd', 'user', 'pwd')</code>	Conecta con la BD y devuelve la conexión. No es necesario con la config. de CodeIgniter establecida anteriormente
<code>\$bean = R::dispense('bean')</code>	Hace un "new" de un objeto persistente de la clase "bean"
<code>\$id = R::store(\$bean)</code>	Persiste un bean (persistente) en la BD
<code>R::trash(\$bean)</code>	Borra un bean (¡OJO! puede que en cascada)
CARGAR	
<code>\$bean = R::load('bean', \$id)</code>	Carga un bean conociendo su id. Devuelve \$bean->id = 0 , si no lo encuentra
<code>\$bean = R::findOne('bean', 'nom=?', ['Pepe'])</code>	Carga un bean con el criterio WHERE indicado. Devuelve null , si no encuentra
<code>\$beans = R::find('bean', 'edad > ? and nom like ?', ['18', 'Pep%'])</code>	Carga una colección de beans con el criterio WHERE indicado. Devuelve [] , si no encuentra
<code>\$beans = R::findAll('bean', 'order by num desc')</code>	Carga una lista con todos los beans de un tipo. Opcionalmente se pueden ordenar y/o limitar
<code>R::exec("...sentencia CUD SQL...")</code> <code>\$filas = R::getAll("...sentencia SELECT SQL...")</code> <code>\$fila = R::getRow("...sentencia SELECT SQL...")</code> <code>\$columna = R::getCol("...sentencia SELECT SQL...")</code> <code>\$dato = R::getCell("...sentencia SELECT SQL...")</code> <code>\$arr_asoc = R::getAssoc("SELECT id,c1,c2 FROM...")</code> <code>\$arr_asoc = R::getAssocRow("SELECT id,c1,c2 FROM...")</code>	Ejecuta una sentencia genérica SQL, y en el caso de ser una sentencia SELECT, la empaqueta de forma distinta en función del método utilizado
RELACIONES 1-N ó N-1	
<code>\$elemento -> caja = \$caja</code>	Asignación desde el lado N
<code>\$elemento -> caja = null</code>	Eliminar la pertenencia a un contenedor
<code>\$caja -> ownElementoList [] = \$elemento</code>	Añade un elemento a la caja
<code>\$caja -> ownElementoList = []</code>	Elimina todos los elementos de una caja
<code>\$caja -> countOwn('elemento')</code>	Cuenta los elementos de la caja
<code>unset (\$caja -> ownElementoList [3])</code>	Elimina el elemento de id=3 de una caja

foreach (\$caja -> own Elemento List as \$e)	Recorre los elementos de una caja
Usar xown (en lugar de own)	Establece dependencia de existencia (borrado en cascada)
ALIASING en 1-N ó N-1	
\$pelicula1 -> director = \$cineasta1; \$pelicula1 -> productor = \$cineasta2;	Añade dos elementos distintos desde el lado N, Utilizamos un alias, no el nombre del bean. Importante: hacer el R::store(...) desde el “lado N” siempre.
\$cineasta1 -> alias ('director') -> own Pelicula List	Recupera la lista de elementos en las que yo tengo un alias determinado
\$peli -> fetchAs ('cineasta') -> director -> nombre \$bean -> fetchAs ('otroBean') -> campo -> atrib	Consulta el bean cineasta al que yo (película), añadí como “director”. Luego consulto uno de sus atributos (nombre)
\$peli -> director -> nombre	Idéntico al anterior. Sólo funciona para RedBeanPHP >= 4.2 y si previamente se ha ejecutado R::setAutoResolve (true) No es necesario con la config. de Codelgniter establecida al principio de este documento
RELACIONES N-N	
\$persona -> shared Aficion List [] = \$aficion	Añadir un elemento desde cualquier lado
\$persona -> shared Aficion List	Recupera la lista de aficiones de una persona
ALIASING en N-N	
	<i>Se implementa con beans ad-hoc de relaciones 1-N y N-1</i>
\$gusta -> persona = \$persona \$gusta -> cancion = \$cancion1 \$aborrece1 -> persona = \$persona \$aborrece1 -> cancion = \$cancion2 \$aborrece2 -> persona = \$persona \$aborrece2 -> cancion = \$cancion3	Asignaciones Se harán siempre desde los beans ad-hoc <i>p.ej. A \$persona le gusta la canción \$cancion1 y aborrece la \$cancion2 y la \$cancion3</i>
\$cancionesQueGustanAUnaPersona = \$persona -> aggr (' own Gusta List ', 'cancion') \$personasQueAborrecenUnaCancion = \$cancion -> aggr (' own Aborrece List ', 'persona')	Recupera una lista de agregados. <i>Para que este ejemplo funcione, el nombre del tipo de los beans ad-hoc ha de ser “gusta” y “aborrece” respectivamente</i>

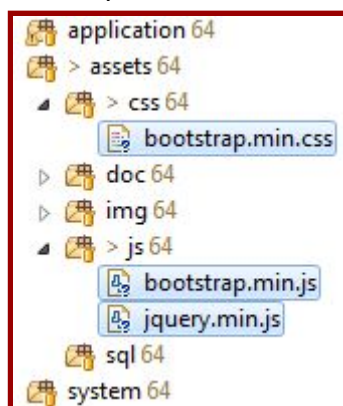
Incluyendo librerías bootstrap dentro del proyecto

Las cabeceras típicas de bootstrap son las siguientes:

```
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js">
</script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js">
</script>
```

Esas URI's están en términos de "maxcdn.bootstrapcdn.com" y "ajax.googleapis.com". Si por cualquier razón no se puede acceder a esos servidores, nos quedaremos sin formato de nuestras páginas, para ello es recomendable descargarse el contenido de estos archivos a nuestro proyecto de la siguiente manera:

1. Descargar cada uno de los tres archivos indicados a nuestro disco duro (p.ej. pinchando con el botón derecho sobre el link de arriba, al ver el código fuente de una página HTML con cabeceras de bootstrap y especificando, "Guardar como")
2. Copiar dichos ficheros a nuestra carpeta "**assets**" de la siguiente manera:



3. Cambiar el fichero "**application / views / templates / head.php**" para que aluda a la ruta local de nuestra aplicación, para esos tres ficheros, con algo del siguiente estilo

```
<link href="php echo base_url() ?assets/css/bootstrap.min.css" rel="stylesheet">
<script src="php echo base_url() ?assets/js/jquery.min.js">
</script>
<script src="php echo base_url() ?assets/js/bootstrap.min.js">
</script>
```

4. (Sólo en Bootstrap < 4.X) El problema de esta configuración es que funcionará el "core" de nuestros diseños bootstrap, pero si añadimos glyphs o fuentes, quizá no funcionen porque se intentarán cargar utilizando una ruta relativa.
 - a. Para arreglar esto se pueden bajar independientemente los glyphs desde aquí
 - b. <http://getbootstrap.com/getting-started/#download>



- c. Descargar el zip, pinchando en "Download source", descomprimirlo y copiar las carpetas "**fonts**" y "**dist / fonts**" bajo "assets"

