

codeigniter

php

set up

- descomprimir .zip actual
- mover carpetas config y application donde corresponda, fuera del WEBROOT si es posible
- copiar el .htaccess
- index.php
 - dejarlo en el WEBROOT
 - cambiar las variables \$system y \$application con la nueva ubicación de esas carpetas
- application/config/config.php
 - eliminar el contenido de la entrada index_page (para soportar el .htaccess)
 - cambiar el base_url
 - activar HOOKS en enable_hooks
 - añadir clave encriptación (OJO; guardar en lugar seguro)
 - se usa para cookies en Session
 - establecer el tipo de entorno en el que se está trabajando (production, development) para mostrar mensajes de error, etc...
 - crear carpetas para cada entorno dentro de application/config y copiar los archivos desde esa raíz de los ficheros que se tengan que modificar a causa del entorno
 - habilitar protección de formularios CSRF
- /application/config/database.php
 - poner los datos de conexión. tener en cuenta que se pueden establecer diferentes archivos en función del entorno (desarrollo, test, producción)
- application/config/routes.php
 - establecer propias reglas de ruteo:
 - \$route['about/:any'] = «estaticas/about» ← por ejemplo, para que una URL «bonita» se redirija a un controlador de páginas estáticas
 - \$route['contactus/:any'] = «estaticas/contactus» ← por ejemplo, para que una URL «bonita» se redirija a un controlador de páginas estáticas
 - \$route['where/:any'] = «estaticas/\$1» ← al estilo de lo de arriba, más mejor... :) incluso mejorable y potencialmente peligroso
 - \$route['yoquese/:any'] = «estaticas/index/\$1» ← todo a una misma función pasando la página como parámetro de la misma. Habría que comprobar existencia de la vista
- controlador base de la aplicación
 - clase User_agent
 - clase Session
- formularios
 - csrf_verify()
 - clase input
 - crear en /application/config/form_validation.php las reglas de validación para que se autoejecuten (controlador/funcion) aparte de las genéricas (registro, etc...)
- vistas & sesiones
 - para mensajes temporales con redirecciones, usar
\$this->session->set_flashdata(nombre, valor); ← se destruyen al usar ¿como se recuperan en la vista? ¿\$this->session->get_flashdata(nombre);?
- everywhere
 - clase Lang?

models

- la clase ha de extender de `CI_Model`
- se carga con `$this->load->model(' <modelo> ' [, ' <otro_nombre_instancia> '] [, ' TRUE ']) ;`
 - el tercer parámetro obliga a la conectar al cargar el modelo (si está a TRUE)
 - en lugar de TRUE se le pueden pasar los parámetros de conexión si no se van a utilizar los definidos en `/config/database.php`
 - http://codeigniter.com/user_guide/general/models.html

controllers

- la clase ha de extender de `CI_Controller`
- cargar helper → `$this->load->helper(' <helper> ')`
- crear archivo PHP con el mismo nombre que la clase (en minúsculas)
- la función `index` se ejecuta «automáticamente» al invocar a la clase
- podemos cargar una vista con `$this->load->view()`
- ~~añadir en el constructor `$this->load->scaffolding($tabla)`~~
- añadir en `index()` la carga de la tabla → `$data['query'] = $this->db->get($tabla)`

views

- para cargar una vista → `$this->load->view(' <vista> ') ;`
- crear archivo con el nombre de la clase y «_view» (no obligatorio, por normativa)
- utilizar sintaxis PHP para mostrar datos del controller en la vista
- recorrer los registros de la tabla on `$query->result()` y `$row->campo`
- añadir función del helper **URL** `<?anchor('blog/comments/' . $row->id, 'comentarios');?>` que se encargará de llamar a la función `comments` de la clase `Blog`

helper

- para cargar helper(s)
 - `$this->load->helper(' <helper> ') ;`
 - `$this->load->helper(array(' <helper> ' , ' <helper> ')) ;`
- disponibles
 - url (poner echo delante)
 - `anchor(' <controlador[/método]> ')` → enlace al controlador[/método]
 - `$this->uri->segment(n)` ← recoge el parámetro posición n de la URL
 - form (poner echo delante)
 - `form_open(' <controlador> ')` ← controlador[/método] es quien recoge los datos
 - `form_label(' <nombre_campo> ' , ' <nombre_id> ')`
 - `form_input(array('name' => 'nombre', 'id' => 'nombre', 'size' => '50', 'value' => set_value('nombre')))` ← `set_value` sirve para la recarga de datos en el formulario
 - `form_password(array('name' => 'password', 'id' => 'password', 'size' => '50'));`
 - `form_submit(' <nombre_campo> ' , ' <texto> ') ;`
 - `form_close()`

library

- `$this->load->library('<libreria>');`
 - `$this->load->library(array('<libreria>', '<libreria>'));`
- disponibles
 - form_validation
 - `$this->form_validation->set_rules('<nombre_campo>', '<texto_referencia_e rrores>', 'validador|validador|funcion_php')`
 - `$array = array(«name» => «trim|required», «email» => «trim|required|valid_email»); $this->form_validation->set_rules($array)`
 - se pueden encadenar validadores y funciones PHP en cualquier orden
 - los cambios que efectuen las funciones PHP se mantienen al recuperar el campo
 - validadores: required, valid_email, matches[<otro_campo_formulario>], min_length[n], etc...
 - http://codeigniter.com/user_guide/libraries/form_validation.html#rulereference
 - se puede crear una función PHP para validar, al margen de las vistas con `callback_funcion_check`. Solo hay que crear la `funcion_check` en el controlador
 - http://codeigniter.com/user_guide/libraries/form_validation.html#callbacks
 - http://codeigniter.com/user_guide/libraries/form_validation.html#validationrules
 - se pueden crear grupo de validadores en `application/config/form_validation.php` para su uso global y automatizado
 - ese fichero ha de contener un array `$config` y arrays contenidos, con etiqueta `<grupo>` y dentro las reglas a validar
 - se puede llamar automaticamente al hacer un `$this->form_validation->run(<grupo>)`
 - se ejecuta automaticamente al hacer un `$this->form_validation->run()` si existe una etiqueta de grupo que coincida con la clase/método
 - http://codeigniter.com/user_guide/libraries/form_validation.html#savingtoconfig
 - funciones «evidentes»: trim, md5, sha1
 - `$this->form_validation->set_message('<validador>', '<mensaje de error cuando el validador no se cumple>')`
 - `$this->form_validation->run()`
 - `$this->input->post('<campo>')` → recupera los campos, con las modificaciones que se hayan podido efectuar en `set_rules`, con protección XSS si está activo en `config/config.php`
 - pagination
 - `$this->load->library('pagination');`
 - `$config = array(«base_url», «total_rows», «per_page», «uri_segment»]; ← rellenar`
 - `$this->pagination->initialize($config);`
 - `$data['pagination'] = $this->pagination->create_links();`
 - unit_test
 - `$this->unit->run(<test>, <resultado>, <texto>);`
 - test = array de pruebas
 - resultado = array (o no) de resultados
 - se ejecuta un foreach del array test ejecutando el `$this->unit->run`
 - `$this->unit->report();`
 - table
 - session

- `$this->session->sess_destroy();`
- `$this->session->sess_create();`
- `$this->session->set_userdata(array(...));`
- `$this->session->userdata(<campo>);`
- email
 - http://codeigniter.com/user_guide/libraries/email.html
 - [Day 3 : Sending Emails](#)
- creación de una librería
 - en el constructor, hacer una instancia: `$this->CI = & get_instance();` para poder acceder a todas las funciones de CodeIgniter (CI puede ser cualquier nombre, solo hay que referenciarse a él)
 - por ejemplo, `$this->CI->db->getwhere(...)`

core

- db
 - http://codeigniter.com/user_guide/database/active_record.html
 - se pueden concatenar: `$query = $this->db->order_by(<campo>)->get(<tabla>)`
 - `$this->db->select('<campo>,<campo>,<campo>');`
 - `$this->db->select_max('<campo>','<alias>');`
 - `$this->db->select_min('<campo>','<alias>');`
 - `$this->db->select_avg('<campo>','<alias>');`
 - `$this->db->select_sum('<campo>','<alias>');`
 - `$this->db->distinct()`
 - `$this->db->from('<tabla>');`
 - `$this->db->join('<tabla>','<condicion_join>');`
 - `$this->db->count_all('<tabla>')`
 - `$this->db->order_by('<campo>','asc|desc')`
 - `$this->db->group_by('<campo>','asc|desc')`
 - `$this->db->where('<campo>','valor')`
 - `$this->db->where_in('<campo>','array_valores')`
 - `$this->db->where_not_in('<campo>','array_valores')`
 - `$this->db->or_where_in('<campo>','array_valores')`
 - `$this->db->or_where_not_in('<campo>','array_valores')`
 - `$this->db->or_where('<campo>','valor')`
 - `$this->db->like('<campo>','valor')`
 - `$query = $this->db->get('<tabla>','limite,offset')`
 - `$query->result()`
 - `$query->num_rows`
 - `$query =`
`$this->db->getwhere('<tabla>',array('<campo>'=<variable>,<campo>'=>variable)`
`;` ← auna el where y el get en una instrucción
 - `$this->db->insert('<tabla>,<datos>');`
 - `$this->db->insert_id()`
 - `$this->db->update('<tabla>,<datos>');` ← utilizar antes `$this->db->where()`
 - `$this->db->delete('<tabla>');` ← utilizar antes `$this->db->where()`
- output
 - `$this->output->enable_profiler(TRUE);`
- benchmark
 - `$this->benchmark->mark('<marca>');`
 - `$this->benchmark->elapsed_time('<marca_inicio>','<marca_final>');`
- router
 - `$this->router->method` → último método llamado

config

- config.php
 - base_url : la URL que se añade delante de todo
 - index_page : la página que se abre/busca por defecto, dejar en blanco si se usa .htaccess
 - global_xss_filtering : para evitar ataques XSS, usando `$this->input->`
 - enable_hooks
- routes.php
 - para establecer el controlador «por defecto» de la aplicación
- database.php
 - introducir los valores de conexión de la BBDD (parece que permite varios)
- autoload.php
 - añadir a 'core' 'database' para poder soportar esa librería (también se puede cargar a través de `$this->load->database()`);
- hooks.php
 - indicar el momento de ejecución de los scripts que consideremos necesarios
 - `$hook['post_controller_constructor'] = array(«class»=>, «function»=>, «filename»=>, «filepath»=>);`
 - crear archivo (en directorio hooks)
 - es necesario instanciar como en una librería creada
- routes.php → añadir en 'scaffolding_trigger' 'scaffolding' ← palabra secreta para activar la feature 'scaffolding' en el modelo
 - activa la opción de ejecutar un editor de datos de tablas simple

codeigniter from de scratch

<http://net.tutsplus.com/sessions/codeigniter-from-scratch/>

- [day 2 : Database Selecting Methods](#)
- [Day 3 : Sending Emails](#)
- [day 4 : newsletter signup](#)
- [day 5 : CRUD operations](#)

primer video ejemplo

[blog.php](#)

```
<?php

class Blog extends CI_Controller {

# si reescribimos el constructor de la clase, invocar al constructor de la
clase padre
    public function __construct() {
        parent::__construct();
    }

# se invoca directamente con http://www.ejemplo.com/index.php/blog
    public function index() {

# se crea un array que se pasará a la vista para que pueda acceder a esas
variables por el nombre
```

```
$data["titulo"] = "Mi Titulo";
$data["cabecera"] = "Mi Cabecera";
$data["cosas"] = arra("limpiar","comprar","llamar a mama");
$this->load->view("blog_view",$data);

}

# se invoca con http://www.ejemplo.com/index.php/blog/mate
public function mate() {
    echo "mate world";
}
}
?>
```

blog_view.php

```
<html>
  <head>
    <title><?=$titulo?></title>
  </head>
  <body>
    <h1><?=$cabecera?></h1>
    <ol>
      <?php foreach($cosas as $item): ?>
        <li><?=$item?></li>
      <?php endforeach; ?>
    </ol>
  </body>
</html>
```

segundo video ejemplo

controllers/blog.php

```
<?php

class Blog extends CI_Controller {

# si reescribimos el constructor de la clase, invocar al constructor de la
clase padre
    public function __construct() {
        parent::__construct();
    }

# DEPRECATED > nombre de la tabla. Se invoca con
http://www.ejemplo.com/index.php/blog/scaffolding <- es la palabra secreta
definida en config/routes.php
    $this->load->scaffolding("entradas");
}
}
```

```
# se invoca directamente con http://www.ejemplo.com/index.php/blog
public function index() {

# se crea un array que se pasará a la vista para que pueda acceder a esas
variables por el nombre
$data["titulo"] = "Mi Titulo";
$data["cabecera"] = "Mi Cabecera";
$data["cosas"] = array("limpiar", "comprar", "llamar a mama");

$data['query'] = $this->db->get("CI_Entradas");

$this->load->view("blog_view", $data);

}

# se invoca con http://www.ejemplo.com/index.php/blog/mate
public function mate() {
    echo "mate world";
}
}

?>
```

[views/blog_view.php](#)

```
<html>
  <head>
    <title><?=$titulo?></title>
  </head>
  <body>
    <h1><?=$cabecera?></h1>
    <ol>
      <?php foreach($cosas as $item): ?>
        <li><?=$item?></li>
      <?php endforeach; ?>
    </ol>
    <?php foreach($query->result() as $row): ?>
      <h3><?=$row->titulo?></h3>
      <p><?=$row->cuerpo?></p>
    <?php endforeach; ?>
  </ol>

  </body>
</html>
```

From:
<https://miguelangel.torresegea.es/wiki/> - miguel angel torres egea

Permanent link:
<https://miguelangel.torresegea.es/wiki/development:php:codeigniter:start?rev=1573818407>

Last update: 15/11/2019 03:46

