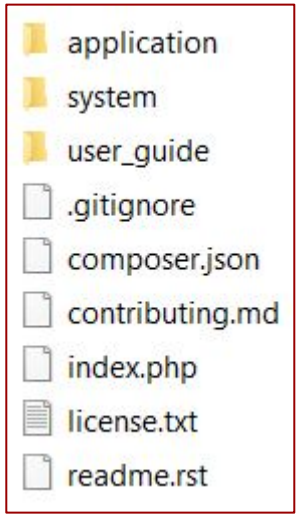
CodeIgniter básico

Realizado por A.Garay (dpto.de Informática)

Descarga e instalación

- Descargar, descomprimir y copiar en una carpeta accesible por apache (p.ej. CI)
  - https://codeigniter.com/
- Comprobar que funciona navegando a http://localhost/CI



Ajuste inicial: URL's y controlador

- Eliminación del "index.php" en las URL's
  - Incluir un fichero ".htaccess" en la raíz de la instalación de CodeIgniter

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

- Elección del controlador por defecto (p.ej. Home (en Home.php))
  - Editar el fichero "application/config/routes.php"
  - Añadir la línea $route['default_controller'] = 'home';
    - El nombre de las clases "controladoras" así como del archivo ".php" que las contiene ha de comenzar por mayúscula, pero en el fichero "routes.php", debe estar en minúscula
- Borrar la carpeta user_guide (si existiera)

Controladores

- Son clases que se crean en la carpeta “application/controllers”
- Deben heredar de “CI_Controller”
- Cada acción se representa con un método de la clase.
- Si se requiere de una acción por defecto, ésta debe llamarse index()
- Para invocar una acción “a1” de un controlador “MiCont”, basta escribir la url
  - http://localhost/CI/miCont/a1
- Para invocar la acción por defecto, basta con:
  - http://localhost/CI/miCont
- Todos las “rutas extra” de una url, serán parámetros para la acción asociada
  - localhost://CI/miCont/a1/p1/p2/p3
  - class MiCont extends CI_Controller {function a1($p1,$p2,$p3) {...} }
- Se pueden organizar los controladores en directorios
  - localhost://CI/producto/zapato/comprar/4
- Se pueden utilizar dentro de cada acción/función parámetros enviados vía GET o POST

Vistas (1/2)

- Son ficheros que se ubican bajo el directorio “application/views”, y contienen HTML “salpicado” de marcas <?php … ?> del estilo <?=...?>, <?php foreach …. : ?>, etc.
- Se cargan desde un controlador mediante $this -> load -> view(‘ficheroVista’)
  - Se cargaría el fichero “application/views/ficheroVista.php” (si no se indica la extensión utiliza .php)
- También se pueden organizar las vistas en carpetas: $this -> load ->view(‘c1/c2/ficheroVista’)
  - Cargaría el fichero “application/views/c1/c2/ficheroVista.php”
- Se le pueden pasar datos de la siguiente manera:
  - $this -> load -> view(‘ficheroVista’,$datos), donde $datos es un array asociativo del estilo:
    - $datos = [‘dato1’ => valor1, ‘dato2’ => valor2 , …..]
    - En la vista tendremos accesibles las variables $dato1, $dato2, etc...
- Los datos pasados de la forma anterior se pueden recuperar en las vistas así de sencillo.
  - Ej: <body> …. El valor del dato1 es: <?= $dato1 ?> </body>

Vistas (2/2)

- También se pueden pasar datos desde el controlador empaquetados en objetos. En ese caso, en lugar de aludir a los índices del array asociativo, aludiremos a los atributos del objeto.
- Se puede obtener el código HTML de la vista en lugar de desplegarla poniendo el último parámetro de view a true.
  - $this -> load -> view('miVista') transmitiría vía HTTP el código HTML al navegador.
  - $codigoVista = $this -> load -> view('miVista','',true) guardaría en $codigoVista el string que contiene el HTML de la vista.
- Se pueden utilizar helpers en el código de las vistas para facilitar su despliegue.
- Deberemos utilizar la función "base_url(..)" del helper "url" para concatenarla a nuestras rutas (en href de links, actions de forms, AJAX, css, etc.)

Helpers

- Son librerías que nos ayudan a realizar el código de manera más sencilla
- Se cargan así:
  - $this -> load -> helper (‘nombre_del_helper’)
  - Ej: $this -> load -> helper (‘url’)
  - O bien en el fichero application/config/autoload.php
- Una vez cargados se pueden utilizar sus funciones dentro del código.
  - Ej: **<?=** anchor('blog/comentarios’, 'Pincha aquí') **?>**
    - Devuelve el código HTML de un link
- Para ver una descripción de todos los helpers de CodeIgniter, pincha aquí
- Para escribir tu propio helper, crear un archivo de funciones en el directorio application/helpers llamado miHelper_helper.php. Se cargaría ejecutando $this->load->helper(‘miHelper’)

Array Helper
CAPTCHA Helper
Cookie Helper
Date Helper
Directory Helper
Download Helper
Email Helper
File Helper
Form Helper
HTML Helper
Inflector Helper
Language Helper
Number Helper
Path Helper
Security Helper
Smiley Helper
String Helper
Text Helper
Typography Helper
URL Helper
XML Helper

Modelos (1/2)

- Primero hay que ajustar los parámetros de la BD en el fichero “application/config/database.php”
- Son clases que se crean bajo la carpeta “application/models”.
- El archivo que lo contiene se debe nombrar <MiModelo>_model.php
- Deben heredar de “CI_Model”
- Cada clase modelo representa a una clase de un DCD. Sus atributos son los atributos de esa clase y sus métodos (funciones) las funciones de clase. También podremos añadir otros métodos para obtener datos de ese modelo que nos resulten útiles.
    - Si queremos mantener una relación objetual más acorde al diseño deberíamos utilizar un software de persistencia como RedBeanPHP, ya que si no, deberemos implementar nosotros todas las relaciones ORM “a mano”.

Modelos (2/2)

- Para crear un objeto de la clase MiModelo basta con invocar $this->load->model('miModelo'), desde el código de un controlador. A partir de entonces invocaremos los métodos del modelo (desde el controlador) mediante $this->miModelo->métodoDelModelo()
- Dentro del código de los métodos de un modelo podemos utilizar funciones como…
  - $this->load->database() Conecta con la Base de datos y la hace accesible a través de $this->db
    - Si cargamos el modelo con $this->load->model('miModelo','alias',**true**), la BD se conecta automáticamente, y hace innecesario lo anterior.
  - $this->db->get('miTabla')->result() Devuelve todas las filas de 'miTabla' como una lista de objetos.
  - $this->db->insert('miTabla',$this) Inserta una fila en miTabla con los valores de los atributos de este objeto (se asume que la tabla tiene columnas con los mismos nombres de los atributos, y que éstos son "public")
  - $this->db->query($miSQL)->result() Ejecuta la consulta $miSQL de forma genérica, y devuelve el resultado como una lista de objetos

Trabajando con información estática

- En un sitio web, el contenido dinámico lo generan los controladores desplegando vistas que tienen “huecos” de información que rellenan habitualmente con datos provenientes del modelo.
- SIn embargo, también hay muchos elementos estáticos (html, css, javascript, imágenes, video, audio, etc.), que escapan al “juego” de la construcción dinámica de páginas y deben ser tratados de forma diferente.
- Para empezar no deberían estar ubicados bajo la carpeta *application*, sino bajo otra carpeta que colgaría directamente del directorio raíz en muchas habitualmente se denomina “*assets*”. La razón de esto es porque la configuración por defecto de CodeIgniter impide (mediante .htaccess) la navegación por la carpeta application.
- En assets, habitualmente organizaremos nuestro contenido estático por familias (css, js, img, etc.), y dentro de cada familia, incluso podríamos mantener una estructura similar a la de las vistas para saber qué contenido estático se corresponde con cada vista, pero esto ya es decisión de cada diseñador.
- Incluso se podrían aplicar permisos particulares con .htaccess por usuario para impedir navegar en contenidos privados, imágenes películas, pdf’s de libros que sólo haya comprado un determinado usuario, etc.
- La forma de enlazar mediante “hrefs” en las vistas a este contenido estático conviene hacerlo trabajando con funciones helper como “*base_url*” o “*link_tag*”, y configurando la URL base de nuestra aplicación en el fichero “*application/config/**config.php***”

Referencias

- Tutorial de codeigniter
  - http://www.codeigniter.com/user_guide/general/index.html
- Manual de comandos de “model”
  - http://escodeigniter.com/guia_usuario/database/active_record.html