

MVC y otros...

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(...)`

From:
<https://matebcn.eu/> - **miguel angel torres egea**

Permanent link:
<https://matebcn.eu/development:php:codeigniter:mvc>

Last update: **28/01/2020 21:50**

