CodeIgniter... a modo mio

Stefano Bianchini
versione 1.1

11 giugno 2014

Indice

Capitolo 1

Prefazione

1.1 Perché questa guida

CodeIgniter è un ottimo framework, completo nonché affidabile, ed è nata in me la volontà di diffonderlo, in lingua italiana, attraverso un formato che fosse facilmente stampabile e consultabile. Ho sviluppato, per puro sfizio personale, il progetto Passeggiainbranco.com[1], completamente basato su CodeIgniter; inoltre ho rilasciato il codice sorgente di Crono[2], un Time Tracker web based, il quale possiede un frontend Bootstrap + jQuery e un backend API Php (ovviamente, su framework CodeIgniter).
Crono, essendo open source, può essere liberamente scaricato; ciò lo rende una buona base di partenza per i neofiti del framework.

Consigli e critiche sono ben accetti. Potete seguire gli aggiornamenti a questo libro collegandovi al mio sito personale[3] o al mio blog[4].

Una nota importante: per ora ho descritto solo gli aspetti più comuni ed importanti di CodeIgniter, sarà mia cura approfondire, nelle versioni successive di questa guida, le caratteristiche che non ho incluso.

1.2 Prerequisiti

Buona conoscenza del linguaggio Php e HTML, del web server Apache.

[1] http://www.passeggiainbranco.com
[2] https://github.com/bianchins/crono
[3] http://www.stefanobianchini.net
[4] http://stefanobianchini.blogspot.com

1.3 Qualche parola sull'autore

Mi chiamo Stefano Bianchini, Classe '82, ingegnere specialistico informatico, sono un appassionato sviluppatore Php e adoro il Web in generale. Altre tecnologie a me care sono jQuery, MySql e C#. Nel tempo libero adoro passeggiare assieme a mia moglie Romina, mio figlio Gabriele e la mia adorabile cagnetta meticcia - barboncino Laila.

1.4 Licenza

Capitolo 2

Introduzione a codeigniter

2.1 Perché CodeIgniter

Arriva un momento, nel percorso lavorativo di un programmatore Php, nel quale si avverte la mancanza di un sostegno che possa semplificare la creazione e la gestione di progetti complessi.

Questa sensazione porta alla ricerca dei cosiddetti framework, ossia particolari strumenti che permettono di velocizzare determinate operazioni di uno sviluppatore, aiutandolo anche (nei limiti del possibile) ad evitare gli errori più comuni durante la progettazione e la stesura del codice. I framework contengono metodi e funzioni in risposta ai problemi più comuni; permettono di interagire facilmente con vari tipi di database e supportano una caratteristica importante, la scalabilità.

Codeigniter è un framework, perfetto a mio modo di vedere, per chi ha il bisogno di avere il pieno controllo del proprio codice Php e necessita però di scorciatoie per le operazioni più comuni. Vediamo qualche caratteristica di Codeigniter così come vengono pubblicizzate sul sito del progetto:

- leggerezza - possiede senza dubbio un core di piccole dimensioni, ma in grado di fare grandi cose;
- ottime prestazioni;
- ampia compatibilità con i vari servizi di hosting, a prescindere dalle versioni (dalla 5.1.6 in avanti) e configurazioni del motore Php;
- già funzionante senza configurazioni - appena scaricato il framework non necessita di configurazioni particolari, a meno ovviamente delle impostazioni del database;

- non necessita di essere eseguito da linea di comando (al contrario di simphony);
- non necessita che lo sviluppatore impari un linguaggio di templating (ne supporta uno ma è assolutamente opzionale);
- ha una ottima documentazione - e migliaia di risposte già pronte in caso di problemi.

A queste caratteristiche, che dovrebbero convincere un programmatore php ad adottare Codeigniter, aggiungo le funzionalità che possiede, per me fondamentali per comprenderne appieno le potenzialità:

- pieno supporto di MySQL (4.1+), MySQLi, MS SQL, Postgres, Oracle, SQLite, and ODBC;
- interazione diretta con i database o mediante quello che viene chiamato *Active Record*;
- costringe a sviluppare seguendo il pattern MVC, il quale a mio avviso aiuta sia durante la progettazione che durante eventuali modifiche al software;
- genera URL facilmente leggibili dai motori di ricerca (SEO friendly), ad esempio
  http://www.example.com/news/view/12
- è facilmente personalizzabile ampliandone le potenzialità;
- supporta un sistema di internazionalizzazione (multilingua).

Questo libro tenterà di fare da guida all'interno delle innumerevoli caratteristiche di Codeigniter attraverso minuziose spiegazioni ed esempi diretti.

2.2 Cosa non fa

CodeIgniter non genera codice, al contrario di altri framework famosi ma particolarmente complessi e a linea di comando. Non ha un supporto nativo per la gestione degli utenti (login, registrazione eccetera) ma sono presenti librerie di terze parti che compiono questo lavoro in modo egregio. Nulla vieta, come ho fatto io, di scrivere una propria gestione degli utenti.

2.3 Un po' di storia

CodeIgniter è sviluppato e mantenuto da EllisLab, una società americana che ha sede vicino a Portland. La prima versione del framework ha visto la luce il 28 febbraio 2006.

Attualmente (giugno 2014) il codice sorgente di CodeIgniter ha subito un brusco rallentamento, motivato dalla volontà di EllisLab di, cito, *voler trovare un nuovo padre per CodeIgniter*.

Nonostante questo rallentamento, il framework possiede a mio avviso, nello stato in cui si trova, tutte le caratteristiche per essere uno degli strumenti più utili per uno sviluppatore Php.

2.4 Flusso di funzionamento

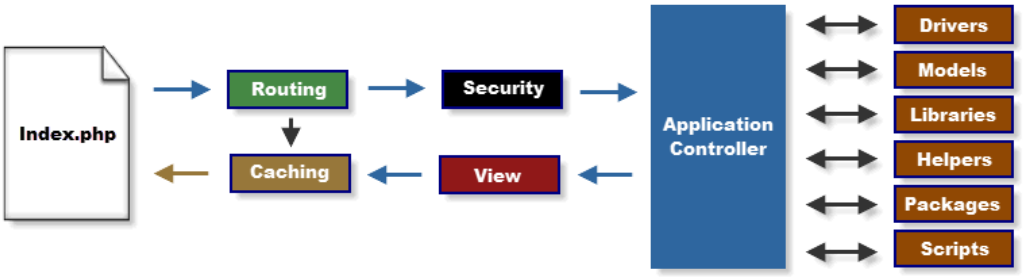

Figura 2.1: Flusso di funzionamento

Quello che uno sviluppatore, abituato a interagire con i files php atti ad una determinata funzione (ad esempio news.php, contact.php ecc) deve capire è la centralità del file index.php, vero e proprio "vigile urbano" di un progetto codeigniter. Non esistono altre pagine all'infuori di index.php nella root principale, ma come si vedrà successivamente tutte le pagine necessarie saranno "controllori" (application controller) seguendo il pattern MVC. Dopo questa precisazione risulta più comprensibile il flusso di funzionamento mostrato in figura 2.1

1. La pagina index.php è quella principale, fulcro del sistema, che inizializza le risorse utilizzate da CodeIgniter

2. Una funzionalità di routing (personalizzabile e modificabile) analizza la richiesta HTTP per determinare il comportamento che il sistema deve adottare.

3. Se un sistema di caching è abilitato e viene trovato il file richiesto in cache, allora si evita la normale esecuzione del file velocizzando il sistema.

4. La richiesta HTTP è filtrata per evitare problemi di sicurezza, prima che qualsiasi application controller sia eseguito.

5. Il Controller carica i modelli, le librerie del core principale, e tutte le risorse necessarie a processare la specifica richiesta. Nel controller risiede, per farla breve, la funzionalità che il programmatore vuole implementare.

6. Viene caricata (dal controller) la Vista, ossia il codice HTML che deve essere mostrato nel browser dell'utente finale. Se il caching è abilitato, la pagina viene salvata in cache per un eventuale richiesta futura.

2.5 Il pattern Model View Controller

Il pattern MVC viene utilizzato sempre più frequentemente e il suo principale vantaggio, applicato al php, è quello di separare l'interazione coi i dati e l'interfaccia grafica. Vediamo nel dettaglio:

- il model fornisce i metodi per interagire con i dati utili all'applicazione;
- il view visualizza i dati contenuti nel model: normalmente è una pagina web in html;
- il controller è l'intermediario tra le due entità precedenti e racchiude l'elaborazione dei dati (ossia una volta ottenuti i dati con il Model, li elabora e li passa al View).

2.6 Installazione e configurazione

Come menzionato in precedenza, l'installazione e la configurazione di questo framework sono operazioni molto semplici e veloci. Dopo averne scaricato l'ultima versione disponibile dal sito codeigniter[1], si scompatta l'archivio nella cartella del web server preposta allo scopo, ad esempio la directory root, htdocs per Apache.

Per la configurazione è sufficiente aprire il file

```
application/config/config.php
```

[1] http://www.codeigniter.com

ed impostare la variabile di configurazione `base_url` con l'Url base della vostra applicazione web, ad esempio:

```
$config['base_url']='http://localhost/';
```

Da notare il fatto che se viene lasciata vuota, CodeIgniter tenterà di comprendere questo parametro da solo (spesso funziona anche se lasciato vuoto).

Basterà aprire a questo punto tramite un browser l'indirizzo specificato come Url base e il sistema vi darà il benvenuto(2.2).

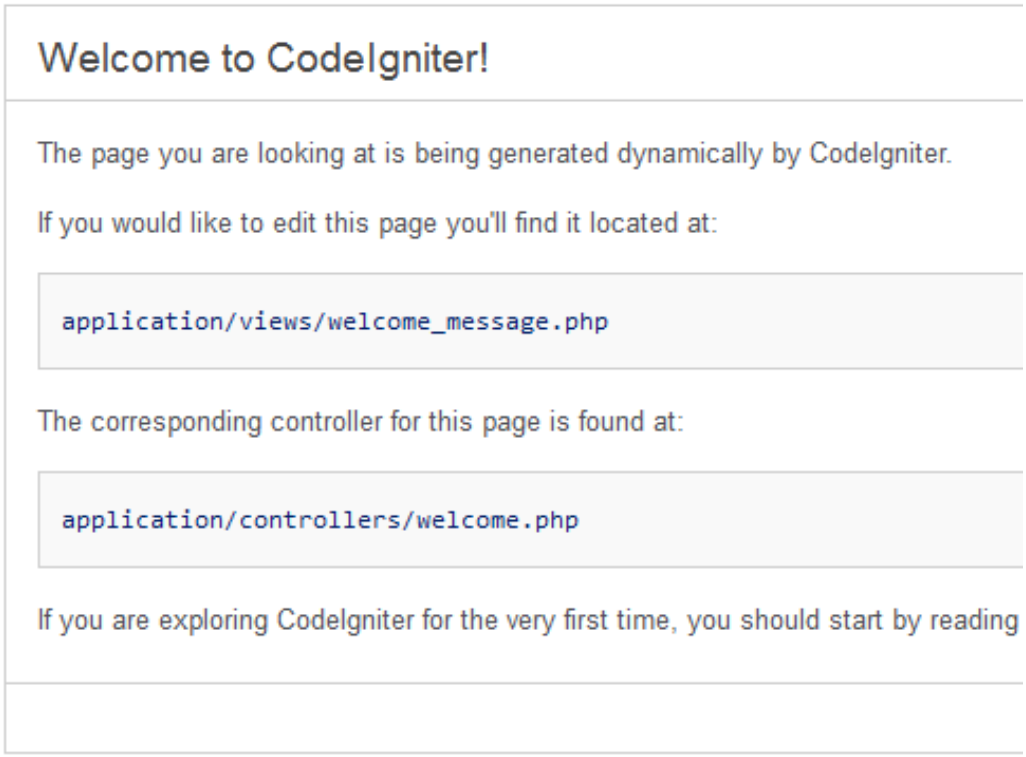

Figura 2.2: La schermata di benvenuto

2.7 Un'occhiata alla struttura

Di base, CodeIgniter ha tre cartelle principali:

application che conterrà tutto ciò che lo sviluppatore deve implementare (controllori, modelli, viste, librerie di terze parti eccetera); di tutte le sue sottocartelle, per il momento suggerisco di notare le cartelle fondamentali, ossia config, controllers, models e views.

system che contiene il cuore del framework.

user_guide che può essere eliminata poiché contiene la versione offline della documentazione.

Consiglio di creare una terza cartella, chiamata `assets`, che conterrà gli script Javascript, i fogli di stile CSS e le immagini di cui l'applicazione avrà bisogno.

2.8 Stile di programmazione

La documentazione ufficiale consiglia un insieme di regole e best-pratices per semplificare la lettura del codice e migliorare l'ordine strutturale di una applicazione web. Di seguito riporto quelli che secondo me sono i consigli più importanti. Se siete curiosi, potete dare una occhiata alla Style Guide ufficiale[2].

2.8.1 Tag di chiusura Php

Per evitare spiacevoli spazi bianchi, che potrebbero comportare errori (header already sent) viene consigliato di non usare il tag di chiusura Php ?¿ alla fine di un documento, ma di inserire due linee di commento che visivamente termineranno il file:

```
/* End of file myfile.php */
/* Location: ./application/controllers/myfile.php */
```

2.8.2 TRUE, FALSE e NULL

I valori TRUE, FALSE e NULL devono essere sempre scritti in carattere MAIUSCOLO.

```
//SBAGLIATO:
if ($foo == true)
$bar = false;
function foo($bar = null)
//CORRETTO:
if ($foo == TRUE)
$bar = FALSE;
function foo($bar = NULL)
```

[2] http://codeigniter.com/user_guide/general/styleguide.html

2.8.3 Operatori logici

È scoraggiato l'uso dell'operatore || (usare OR) e viene richiesto uno spazio prima e dopo l'operatore di negazione !

```
//SBAGLIATO:
if ($foo || $bar)
if (!$foo)
if (! is_array($foo))
//CORRETTO:
if ($foo OR $bar)
if ( ! $foo)
if ( ! is_array($foo))
```

2.8.4 Commenti

Usare lo standard DocBlock[3] per descrivere il comportamento delle funzioni dichiarate:

```
/**
* Encodes string for use in XML
*
* @access public
* @param string
* @return string
*/
function xml_encode($str)
```

2.8.5 Indenting

Viene consigliato l'uso dello stile Allman[4] per la stesura del codice.

2.8.6 Query Sql

Tutte le parole chiave devono essere espresse in maiuscolo (SELECT, FROM, WHERE...) e viene consigliato di separare in più righe le query più complesse.

```
//SBAGLIATO:
// le parole chiave sono minuscole e la query è troppo lunga
// per una singola riga
$query = $this->db->query("select foo, bar, baz, foofoo, foobar↩
    as raboof, foobaz from exp_pre_email_addresses where foo ↩
   != 'oof' and baz != 'zab' order by foobaz limit 5, 100");
```

[3] http://bit.ly/LNCGvF
[4] http://en.wikipedia.org/wiki/Indent_style#Allman_style

```
\\CORRETTO:
$query = $this->db->query("SELECT foo, bar, baz, foofoo, foobar↩
    AS raboof, foobaz
        FROM exp_pre_email_addresses
        WHERE foo != 'oof'
        AND baz != 'zab'
        ORDER BY foobaz
        LIMIT 5, 100");
```

2.9 Nomi riservati

I seguenti nomi sono riservati da CodeIgniter e non possono essere usati dallo sviluppatore rispettivamente per:

2.9.1 Controller

- Controller
- CI_Base
- _ci_initialize
- Default
- index

2.9.2 Funzioni

- is_really_writable()
- load_class()
- get_config()
- config_item()
- show_error()
- show_404()
- log_message()
- _exception_handler()
- get_instance()

2.9.3 Variabili

- $config
- $mimes
- $lang

2.9.4 Costanti

- ENVIRONMENT
- EXT
- FCPATH
- SELF
- BASEPATH
- APPPATH
- CI_VERSION
- FILE_READ_MODE
- FILE_WRITE_MODE
- DIR_READ_MODE
- DIR_WRITE_MODE
- FOPEN_READ
- FOPEN_READ_WRITE
- FOPEN_WRITE_CREATE_DESTRUCTIVE
- FOPEN_READ_WRITE_CREATE_DESTRUCTIVE
- FOPEN_WRITE_CREATE
- FOPEN_READ_WRITE_CREATE
- FOPEN_WRITE_CREATE_STRICT
- FOPEN_READ_WRITE_CREATE_STRICT

Capitolo 3

Primi passi con Codeigniter

3.1 Controller

I Controllers, come già detto, sono la parte fondamentale del software strutturato con il framework CodeIgniter. Il tutorial della documentazione ufficiale li definisce *la colla* della web application. Cerchiamo di comprendere meglio la logica di funzionamento: quando viene fatta una richiesta del tipo

```
http://localhost/index.php/news/view/12
```

il framework elabora la richiesta in questo modo:

```
http://localhost/index.php/<controller>/<metodo>/<parametri>
```

richiedendo l'esecuzione del metodo 'view' con parametro '12' al controller denominato 'news'. Questo significa che, per far funzionare la richiesta GET specificata, deve essere implementato un controller news.php sotto la cartella application/controllers, con il seguente contenuto:

```
<?php
class News extends CI_Controller {
    public function view($id)
    {
        //qui codice per fare qualcosa
    }
    public function index()
    {
        //qui codice per fare qualcosa
    }
}
```

Capendo questo meccanismo, si è a buon punto nella comprensione di CodeIgniter; bisogna smettere di ragionare in termini di files php e iniziare a spostare l'attenzione su controllers e metodi. Anche se inizialmente sembra

complicato, in realtà questo modo di strutturare l'applicazione porta enormi vantaggi.

Non preoccupiamoci, per il momento, del poco estetico 'index.php' all'interno dell'Url che stiamo considerando: con una semplice configurazione può essere eliminato, risultando molto più facile da leggere sia per i motori di ricerca sia per gli esseri umani. Cerchiamo ora di analizzare questo codice, comprendendo la logica base del framework. Per prima cosa un controller deve essere definito come classe Php che estende CI_Controller e deve essere chiamato con lo stesso nome del file (senza l'estensione .php, ma con la prima lettera maiuscola, *news.php* diventa *News*). All'interno della classe notiamo un metodo pubblico (view) che accetta un parametro ($id). Come già scritto, il sistema andrà a richiamare il metodo view con il parametro passato (12) ricercandolo all'interno del controller news. In caso di più parametri, basta separarli con il carattere '/' slash.

Il metodo 'index' non è obbligatorio, tuttavia ha il ruolo di gestire le richieste generiche al controller, cioé non specificando metodo né parametri:

```
http://localhost/index.php/news
```

Ricordiamo è compito del controller fare da tramite tra l'elaborazione dei dati e la visualizzazione degli stessi; il che significa che, una volta eseguite determinate operazioni sui dati, deve caricare un altro mattone fondamentale: la vista (view). Per questo motivo il codice visto in precedenza viene così modificato:

```
public function view($id)
{
    $data['id'] = $id;
    $this->load->view('news/view', $data);
}
public function index()
{
    $this->load->view('news/index');
}
```

La prima funzione ha due istruzioni, che servono per inserire all'interno di un array il contenuto della variabile $id e per caricare in memoria una vista posizionata sotto /application/views/ news/view.php passando il suddetto array. Traspare già il modo per passare dei dati alla vista specificata, in modo che la vista li possa mostrare (vedremo poi in che modo). Da notare che tutte le viste vengono ricercate nella directory /application/views al cui interno possono esistere sottocartelle (news ne è un esempio) che contengono le viste di categorie simili, in questo caso tutte le viste utilizzate dal controller news. La vista deve essere specificata senza l'estensione .php tramite il comando

```
$this->load->view('nomeDellaVista',$eventualeArrayDaPassare);
```

3.2 View

Nel paragrafo precedente abbiamo visto come caricare una vista da un controller, funzione fondamentale per il corretto di una web application con il framework CodeIgniter. Ma come deve essere strutturata una vista? Rispolverando l'esempio precedente, nella directory che contiene le view (application/views) deve essere creata una sottocartella news, che conterra due viste:

- view.php
- index.php

che corrispondono alle viste caricate in precedenza nel codice del controller. Analizziamo ora il contenuto della vista view.php, che e semplicemente HTML con l'aggiunta di codice Php dove necessita

```
<html>
<head>
<title>CodeIgniter a modo mio - Esempio di Vista</title>
</head>
<body>
<h1>Hai specificato il parametro id impostandolo al valore <?↩
   php echo $id; ?></h1>
</body>
</html>
```

Il concetto chiave per il passaggio dei dati da un controller ad una view è il seguente: passando un array del tipo $data['qualcosa'], accederò ai dati dalla view interagendo con la variabile $qualcosa. Nel caso del nostro esempio, $data['id'] diventa $id all'interno della vista. La vista mostra i dati elaborati dal controller; il controller a sua volta accede ai dati attraverso un modello, anche se non è espressamente richiesto dal framework, ma fortemente consigliato dal pattern MVC.

3.2.1 Caricamento di più viste

È buona prassi in una applicazione web complessa suddividere le viste creando un header ed un footer uguali per tutte le pagine. Come è possibile in CodeIgniter fare una cosa del genere, come si possono caricare più viste da un controllore? Semplicemente chiedendo di caricarle una di seguito all'altra:

```
$this->load->view('header');
$this->load->view('example_view', $data);
$this->load->view('footer');
```

3.3 Model

Il Model si occupa dell'interazione con i dati, offre sostanzialmente al controller un approccio ad un livello più astratto delle semplici interrogazioni su database. Ad esempio, per gestire un motore di news, un modello potrebbe avere il seguente scheletro:

```
//news_model.php
class News_model extends CI_Model {
    function __construct()
    {
        // Richiamo del costruttore parent
        parent::__construct();
        $this->load->database();
    }
    function get_latest_news()
    {
        // qui codice
    }
    function insert($data)
    {
        // qui codice
    }
    function update($news_id, $data)
    {
        // qui codice
    }
}
```

Un modello è una classe che estende CI_Model (base Model class), il cui nome deve essere lo stesso del file posizionato sotto application/models/ e deve avere la prima lettera maiuscola. Il modello deve quindi venire caricato dal controller, il quale ne eseguirà le funzioni per accedere ai dati (nell'esempio, get_latest_news, insert o update). Nel costruttore del modello si può notare il caricamento del database

```
$this->load->database()
```

fondamentale per accedere ai dati.

3.3.1 Caricare un modello

Il controller, come già scritto, deve caricare il modello: questo può essere fatto attraverso il comando

```
$this->load->model('News_model');
```

che permette di caricare in una variabile con lo stesso nome del modello (nell'esempio, $News_model) e quindi di poter interagire con essa:

```
$this->News_model->get_latest_news();
```

3.4 Classi native

CodeIgniter possiede alcune classi native che permettono di compiere facilmente le operazioni quali interagire con un database, accedere alle variabili di input (get, post, cookies ecc.), validare form, gestire il multilingua all'interno del progetto, manipolare immagini e tanto altro ancora. Una spiegazione del funzionamento delle classi più utili verrà fatta più avanti, affrontando esempio pratici.

3.5 Helpers

I programmatori nostalgici del periodo precedente il passaggio del Php ad una struttura Object Oriented non potranno che essere entusiasti degli Helpers. Gli Helpers, come suggerisce il nome stesso, sono degli aiutanti che offrono delle funzioni per le operazioni più comuni. Avete capito bene, funzioni, non classi. Semplici funzioni Php. Un Helper va caricato con la seguente sintassi:

```
$this->load->helper('nome_helper');
```

3.6 Interazione con i database

CodeIgniter può interagire con i database attraverso un model (ma non è obbligatorio) o attraverso un controller. Per prima cosa è necessario configurare le impostazioni per il collegamento al database, modificando appositamente il file database.php collocato sotto il percorso /application/configs/. Nell'esempio si fa riferimento ad un dbms di tipo MySql in locale, con username root e password vuota (ambiente di testing).

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "";
```

```
$db['default']['database'] = "database_name";
$db['default']['dbdriver'] = "mysql";
$db['default']['dbprefix'] = "";
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = FALSE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = "";
$db['default']['char_set'] = "utf8";
$db['default']['dbcollat'] = "utf8_general_ci";
$db['default']['swap_pre'] = "";
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

L'impostazione db_debug è settata a FALSE, come se fosse in ambiente di produzione. Inizialmente consiglio di impostarla a TRUE per visualizzare eventuali errori di query (che andranno evitati o gestiti in maniera appropriata!). Uno dei problemi più frequenti è quello di gestire la dualità ambiente di produzione - ambiente di test, spesso divisi come due database server diversi. CodeIgniter permette di gestire una molteplicità di configurazioni di database: ad esempio possiamo inventarci un ambiente chiamato 'produzione':

```
$db['produzione']['hostname'] = "10.0.0.1";
$db['produzione']['username'] = "user1234";
$db['produzione']['password'] = "pwd1234";
$db['produzione']['database'] = "database_name";
$db['produzione']['dbdriver'] = "mysql";
$db['produzione']['dbprefix'] = "";
$db['produzione']['pconnect'] = TRUE;
$db['produzione']['db_debug'] = FALSE;
$db['produzione']['cache_on'] = FALSE;
$db['produzione']['cachedir'] = "";
$db['produzione']['char_set'] = "utf8";
$db['produzione']['dbcollat'] = "utf8_general_ci";
$db['produzione']['swap_pre'] = "";
$db['produzione']['autoinit'] = TRUE;
$db['produzione']['stricton'] = FALSE;
```

A questo punto è sufficiente impostare all'interno del file config.php principale (application/ config) l'ambiente database da utilizzare:

```
$active_group = "produzione";
```

La connessione al database viene eseguita caricando l'oggetto in memoria, come già scritto, con il metodo:

```
$this->load->database();
```

3.6.1 Eseguire query

Per eseguire query arbitrarie è sufficiente richiamare il metodo query sull'oggetto db:

```
$query = $this->db->query('SELECT title, name, body FROM ↩
   messages');
```

Il risultato della funzione sarà un oggetto 'risultato' facilmente consultabile, vediamo come:

```
if ($query->num_rows() > 0)
{
    foreach ($query->result() as $row)
    {
        echo $row->title;
        echo $row->name;
        echo $row->body;
    }
}
```

Nel caso di risultato a singola riga, o anche per ottenere solo la prima riga di un risultato, è disponibile la funzione row():

```
$row = $query->row();
echo $row->title;
```

Per conoscere il numero di righe ottenute come risultato di una interrogazione, solitamente molto utile, è possibile usare num_rows():

```
echo $query->num_rows();
```

3.6.2 Funzione di escaping

Può essere eseguita una funzione di escaping su una stringa che farà parte della query in tre modi diversi:

`$this->db->escape()` questa funzione determina automaticamente il tipo di dato MySql che si sta inserendo nella query ed esegue l'escaping di conseguenza. Aggiunge direttamente anche gli apici singoli prima e dopo il valore richiesto.

`$this->db->escape_str()` esegue direttamente l'escaping, non considerando il tipo di dato Mysql

`$this->db->escape_like_str()` questa funzione dovrebbe essere usata per stringhe destinate a comparazioni di tipo LIKE

```
WHERE campo LIKE '%qualcosa%'
```

3.6.3 Active Record

CodeIgniter possiede una comoda modalità di interazione con i database chiamata Active Record e basata su funzioni immediate e facili da ricordare. Il principio base è quello dell'Active Record Pattern ma la documentazione ufficiale ci tiene a precisare che si tratta di una versione appositamente modificata. La modalità che stiamo considerando permette di eseguire con pochissimo codice operazioni quali inserimento, modifica, eliminazione, selezione di record da un database. Per informazioni su tutte le funzioni disponibili, visitate il sito ufficiale[1].

$$\$this->db->get()$$

Esegue una interrogazione sulla tabella specificata e restituisce i record ottenuti. Opzionalmente è possibile limitare la query tramite LIMIT. È l'operazione finale dopo le varie selezioni e condizioni che vedremo tra poco.

```
$query = $this->db->get('mytable');
$query = $this->db->get('mytable', 10, 20);
```

$$\$this->db->select()$$

Permette di specificare una selezione dei campi all'interno di una query (operazione di SELECT)

```
$this->db->select('title, content, date');
$query = $this->db->get('mytable');
```

$$\$this->db->from()$$

Esegue una operazione FROM sulla query

```
$this->db->select('title, content, date');
$this->db->from('mytable');
$query = $this->db->get();
```

$$\$this->db->join()$$

Il Join è una operazione fondamentale quando si tratta di interagire con un database. Questa funzione permette di farlo in maniera immediata, anche specificando facoltativamente il tipo di join:

```
$this->db->select('*');
$this->db->from('blogs');
$this->db->join('comments', 'comments.id = blogs.id');
```

[1] http://codeigniter.com/user_guide/database/active_record.html

```
// oppure specifico che si tratta di un left join
// $this->db->join('comments', 'comments.id = blogs.id', 'left↩
   ');
$query = $this->db->get();
```

$$\$this->db->where()$$

Permette di specificare tutte le condizioni da applicare all'interrogazione. Ha a disposizione quattro modalità di utilizzo:

- metodo chiave / valore:

  ```
  $this->db->where('name', $name);
  // WHERE name = 'Joe'
  ```

  Il segno uguale è automaticamente inserito dal framework. Se usata più volte, applica tutte le condizioni con l'operatore AND

  ```
  $this->db->where('name', $name);
  $this->db->where('title', $title);
  $this->db->where('status', $status);
  // WHERE name = 'Joe' AND title = 'boss' AND status = '↩
     active'
  ```

- metodo chiave/valore avanzato: È possibile includere un operatore nel primo parametro per avere un miglior controllo della condizione sql specificata

  ```
  $this->db->where('name !=', $name);
  $this->db->where('id <', $id);
  // WHERE name != 'Joe' AND id < 45
  ```

- array associativo:

  ```
  $array = array('name' => $name, 'title' => $title, 'status↩
     ' => $status);
  $this->db->where($array);
  // WHERE name = 'Joe' AND title = 'boss' AND status = '↩
     active'
  ```

  Anche in questo caso si può includere un operatore specifico

  ```
  $array = array('name !=' => $name, 'id <' => $id, 'date >'↩
      => $date);
  $this->db->where($array);
  ```

- Stringa contenente condizioni sql: Viene specificata la stringa where da applicare:

```
$where = "name='Joe' AND status='boss' OR status='active'"↩
    ;
$this->db->where($where);
```

$$\$this->db->group\_by()$$

Come suggerisce il nome, esegue una group by.

$$\$this->db->insert()$$

Genera un comando insert basato sui parametri passati ed esegue la query. Di seguito un esempio con un array passato alla funzione:

```
$data = array(
  'title' => 'My title' ,
  'name' => 'My Name' ,
  'date' => 'My date'
);
$this->db->insert('mytable', $data);
// INSERT INTO mytable (title, name, date) VALUES ('My title', ↩
    'My name', 'My date')
```

Il primo parametro deve contenere il nome della tabella su cui agire, il secondo l'array di valori da inserire.Viene eseguito l'escaping automatico di tutti i valori.

$$\$this->db->update()$$

Come dice il nome stesso, esegue una query di aggiornamento su una determinata tabella.

```
$data = array(
              'title' => $title,
              'name' => $name,
              'date' => $date
           );
$this->db->where('id', $id);
$this->db->update('mytable', $data);
// UPDATE mytable
// SET title = '{$title}', name = '{$name}', date = '{$date}'
// WHERE id = $id
```

$$\$this->db->delete()$$

Genera una stringa Sql di eliminazione:

```
$this->db->delete('mytable', array('id' => $id));
// DELETE FROM mytable
// WHERE id = $id
```

Il primo parametro è il nome della tabella su cui agire, il secondo la condizione where dove applicarla. Quest'ultimo parametro è opzionale, al suo posto è possibile usare la funzione where vista in precedenza:

```
$this->db->where('id', $id);
$this->db->delete('mytable');
// DELETE FROM mytable
// WHERE id = $id
```

3.7 Parametri di input

Per gestire al meglio i parametri GET, POST o COOKIE CodeIgniter mette a disposizione una comoda classe che si preoccupa di mettere in sicurezza le variabili al posto nostro e offre un insieme di funzioni quali capire lo user agent utilizzato, scoprire se e una richiesta ajax, catturare l'indirizzo ip dell'utente (per informazioni [2]). Di seguito sono mostrate le funzioni fondamentali della classe input:

`$this->input->post('some_data')` se $_POST['some_data'] è impostato ritorna il valore, FALSE altrimenti

`$this->input->get('some_data')` se $_GET['some_data'] è impostato ritorna il valore, FALSE altrimenti

`$this->input->cookie('some_data')` se $_COOKIE['some_data'] è impostato ritorna il valore, FALSE altrimenti

`$this->input->server()` ricerca nell'array globale $_SERVER

`$this->input->get_post('some_data')` ricerca sia nelle variabili post che in quelle get

`$this->input->set_cookie()` indispensabile per impostare un cookie

```
$cookie = array(
'name' => 'The Cookie Name',
'value' => 'The Value',
'expire' => '86500',
'domain' => '.some-domain.com',
'path' => '/',
'prefix' => 'myprefix_',
'secure' => TRUE
);
$this->input->set_cookie($cookie);
```

[2] http://codeigniter.com/user_guide/libraries/ input.html

Capitolo 4

Uso avanzato

In questo capitolo verranno approfondite le caratteristiche più tecniche del framework che stiamo analizzando.

4.1 Rimuovere index.php dagli Url

Nell'installazione standard, la stringa 'index.php' compare negli url di Codeingniter, ad esempio:

`example.com/index.php/news/article/my_article`

È possibile rimuovere quell'antiestetico `index.php` usando, ad esempio un file `.htaccess` se si sta lavorando su web server Apache, usando il modulo chiamato `mod_rewrite` e una logica chiamata 'negativa': a parte determinate eccezioni (cartella `assets`, file `robots.txt`, il file `index.php` stesso) considera tutto ciò che viene richiesto come concatenato ad `index.php`.

```
RewriteEngine on
RewriteCond $1 !^(index\.php|assets|robots\.txt)
RewriteRule ^(.*)$ /index.php/$1 [L]
```

Non dimentichiamoci inoltre di modificare il file di configurazione impostando una determinata variabile vuota

```
$config['index_page'] = '';
```

Questa soluzione può non sempre funzionare. A seconda delle configurazioni dell'hosting / web server, può non dare il risultato sperato. In questi casi viene consigliato di tentare tutte le possibili configurazioni per la variabile che determina la stringa URI

```
/*
| Se AUTO (auto-detect) non funziona, prova uno dei seguenti
|
| 'PATH_INFO'       Uses the PATH_INFO
| 'QUERY_STRING'    Uses the QUERY_STRING
| 'REQUEST_URI'     Uses the REQUEST_URI
| 'ORIG_PATH_INFO'  Uses the ORIG_PATH_INFO
*/
$config['uri_protocol'] = 'AUTO'
```

Ad esempio, per il diffuso hosting aruba[1], è necessario il seguente `.htaccess`

```
RewriteEngine on
RewriteCond %{REQUEST_URI} ^/system.*
RewriteCond $1 !^(index\.php|assets|robots\.txt)
RewriteRule ^(.*)$ index.php?/$1 [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.+)$ index.php?/$1 [L]
```

4.2 Sessioni

Codeigniter offre un comodo strumento per mantenere informazioni relative ad un determinato utente, la classe Session. Bisogna subito chiarire che non fa uso delle classiche sessioni Php, ma anzi ha un meccanismo diverso, basato sui cookies oppure, fortemente consigliato, su un database. La visione classica di una sessione, in questo framework, è un cookies che contiene informazioni. Queste informazioni sono strutturate in modo da poter leggere e scrivere variabili di sessione in modo agile e semplice. Per rendere univoca la sessione e fare in modo che i dati siano relativi ad un solo utente, una sessione in CI è così strutturata:

- dati di sessione (variabili)
- id di sessione (univoco)
- indirizzo ip dell'utente
- user agent del browser dell'utente
- timestamp dell'ultima attività

[1] http://www.aruba.it

Di base queste informazioni vengono salvate su un cookie, il quale è notoriamente quanto di più insicuro ci sia in ambito web. C'è ovviamente la possibilità di cifrare il contenuto del cookie di sessione, impostando una determinata variabile all'interno del file di configurazione principale[2].

```
$config['sess_encrypt_cookie']=TRUE;
```

Una limitazione da non poco conto è relativa alla dimensione dei dati salvabili in una sessione basata su cookies, ossia 4 KB. In caso di cifratura del cookie, questo limite si abbassa poiché gli algoritmi spesso aumentano la dimensione della stringa. Poiché le sessioni vengono usate soprattutto in caso di login da parte di un utente, ossia per contenere le informazioni di accesso di quest'ultimo e permettergli di eseguire determinate azioni, la sicurezza ricopre un ruolo fondamentale. Il consiglio è quindi quello di salvare le sessioni su database, creando una semplice tabella[3]:

```
CREATE TABLE IF NOT EXISTS  `ci_sessions` ( session_id varchar↩
    (40) DEFAULT '0' NOT NULL, ip_address varchar(16) DEFAULT '↩
    0' NOT NULL, user_agent varchar(120) NOT NULL, ↩
    last_activity int(10) unsigned DEFAULT 0 NOT NULL, ↩
    user_data text NOT NULL, PRIMARY KEY (session_id) );
```

Comunichiamo ora a Codeigniter che le sessioni dovranno basarsi sulla tabella appena creata:

```
$config['sess_use_database'] = TRUE;
$config['sess_table_name'] ='ci_sessions';
```

Bisogna tenere bene a mente il funzionamento della sessione: dopo un tempo definito il sistema ricreerà il session_id e aggiornerà data e ora dell'ultima attività (il timestamp visto in precedenza). L'intervallo predefinito per questo aggiornamento è 5 minuti, ma può essere modificato.

4.2.1 Inizializzare e distruggere una sessione

Possiamo caricare manualmente la libreria di sessione usando il comando:

```
$this->load->library('session');
```

Per distruggere una sessione, usiamo:

```
$this->session->sess_destroy();
```

[2] /application/config/config.php
[3] nell'esempio una struttura per Mysql

4.2.2 Interagire con le sessioni

Scrivere una variabile di sessione è possibile mediante una specifica funzione della classe Session:

```
//Singola variabile
$this->session->set_userdata('some_name', 'some_value');

//Più variabili in una volta sola con array associativo
$newdata = array(
                    'username'  => 'johndoe',
                    'email'     => 'johndoe@some-site.com',
                    'logged_in' => TRUE
                );

$this->session->set_userdata($newdata);
```

Per ottenere una delle variabili di sessione impostate in precedenza:

```
$username = $this->session->userdata('username');
```

Infine, per cancellare una variabile di sessione (ad esempio `username`):

```
$this->session->unset_userdata('username');
```

4.2.3 Flashdata: un tipo particolare di sessione

Una variabile *flashdata* può essere letta solo una volta, poiché alla lettura la variabile viene eliminata. Molto spesso viene usata per i vari messaggi di errore o di feedback positivo (ad esempio 'Operazione completata con successo'). Una variabile di questo tipo si imposta con la funzione:

```
$this->session->set_flashdata('item', 'value');
```

E si legge (ricordate bene: una sola volta, dopodiché la variabile sarà cancellata) con il comando

```
$this->session->flashdata('item');
```

4.3 Routing

Un'altra caratteristica molto interessante è la possibilità di gestire il routing delle richieste Uri a proprio piacimento. In precedenza abbiamo già appurato che un Uri standard in CI è

`example.com/controller/metodo/parametro/`

Tramite semplici regole, è possibile personalizzare gli Uri in modo da semplificarne il ranking SEO o semplicemente renderli più memorizzabili per l'utente, agendo su un file chiamato routes.php[4]. Ci sono due route predefinite e riservate:

```
//Imposta quale controller caricare in caso di url non ↩
   specificata
//(ossia la pagina iniziale)
$route['default_controller'] = 'welcome';
//sostituisce la pagina 404
$route['404_override'] = '';
```

Queste due route devono essere definite prima di tutte le altre. Ricordatevi che le route vanno in ordine di importanza, quindi più in alto sono, più hanno la precedenza sulle altre.

Tre sono i metodi utilizzati: **routing diretto**, **caratteri jolly** ed **espressioni regolari**.

4.3.1 Routing diretto

Attraverso questo metodo si obbliga CI a sostituire una certa Uri con un'altra. Ad esempio, se volessimo che

`example.com/blog/view/12`

fosse in realtà

`example.com/title-here`

basta specificarlo nel modo seguente

```
$route['title-here'] = "blog/view/12";
```

4.3.2 Caratteri jolly

Abbiamo a disposizione due 'caratteri' jolly per il routing:

(:num) - solo caratteri numerici

(:any) - qualsiasi carattere

Sono molto utili in caso di route semplici. Analizziamo il seguente esempio: desidero accorciare una determinata Uri poiché il sistema mi constringerebbe a passare per il controller `catalog`, il metodo `view` e infine il parametro `id`; al contrario, un utente deve poter ottenere informazioni su un determinato prodotto (identificato dall'`id`, nell'esempio 12) usando semplicemente un Uri come il seguente

[4] application/config/routes.php

`example.com/product/12`

e non quello standard

`example.com/catalog/view/12`

Tutto ciò è possibile con questa semplice route

```
$route['product/(:num)'] = "catalog/view/$1";
```

Da notare che il match del parametro (in questo caso, richiediamo che sia un numerico) viene identificato con $1 dal sistema, cioé il primo parametro che soddisfa la nostra richiesta.

4.3.3 Espressioni regolari

Le espressioni regolari[5] sono un altro metodo per gestire il routing. Ad esempio:

```
$route['products/([a-z]+)/(\d+)'] = "$1/id_$2";
// Trasforma products/shirts/12 in /shirts/id_12
```

4.4 Validazione di form

Una delle caratteristiche più comode e potenti di CI è la cosiddetta *Form Validation*, ovvero quell'insieme di procedure che permettono di controllare i campi dei moduli web (form) basandosi su regole prestabilite. In codeigniter il controller si occupa di fare ciò, mentre la vista ha il compito di mostrare il modulo html e gli eventuali errori di compilazione. Vediamo di comprendere meglio il tutto con un esempio pratico, una semplice procedura di registrazione. Iniziamo con il codice della vista

```
<!-- Vista register.php-->
<html>
<head>
<title>My Form</title>
</head>
<body>

<?php echo validation_errors(); ?>
```

[5]Le ho sempre trovate parecchio noiose. Sapete quelle cose che non avete mai voglia di studiare perché siete sicuri che non vi entreranno mai in testa? Ecco, per me sono le espressioni regolari del Php. Per maggiori informazioni:
http://php.net/manual/en/reference.pcre.pattern.syntax.php

```
<form method="post" action="<?php echo site_url('register');?>"↩
    >

<h5>Username</h5>
<input type="text" name="username" value="" size="50" />

<h5>Password</h5>
<input type="text" name="password" value="" size="50" />

<h5>Password Confirm</h5>
<input type="text" name="passconf" value="" size="50" />

<h5>Email Address</h5>
<input type="text" name="email" value="" size="50" />

<div><input type="submit" value="Registrati" /></div>

</form>

</body>
</html>
```

Niente di trascendentale: viene creato un normale form di registrazione i cui dati verranno elaborati dal controller `register` (l'*action* del form). Il comando `site_url` fa parte dell'helper `url`[6] e genera un url completo partendo dal controller specificato[7].
La funzione `validation_errors` fa intuire che verranno mostrati, se ci sono, eventuali errori di compilazione. Le regole e la logica di funzionamento, come in un MVC che si rispetti, sono implementate nel controller:

```
class Register extends CI_Controller {

    function index()
    {
        //Carico l'helper
        $this->load->helper('url');
        //Carico la libreria per la validazione del form
        $this->load->library('form_validation');

        //Imposto le regole per la validazione
        $this->form_validation->set_rules('username', 'Username↩
            ', 'required|is_unique[users.username]');
        $this->form_validation->set_rules('password', 'Password↩
            ', 'required');
        $this->form_validation->set_rules('passconf', 'Password↩
             Confirmation', 'required|matches[passconf]');
```

[6] l'helper dovrà essere quindi caricato dal controller stesso
[7] come già visto in precedenza, cit. funzioni per url

```
        $this->form_validation->set_rules('email', 'Email', '↩
            required|valid_email');

        //Eseguo il controllo del form
        if ($this->form_validation->run() == FALSE)
        {
            //Questo viene eseguito anche quando il form
            //non è compilato, cioé la prima volta che si ↩
                carica
            //la pagina con il form html
            $this->load->view('register');
        }
        else
        {
            //Se i controlli sono positivi, redireziono l'↩
                utente
            redirect('register/success');
        }
    }

    function success()
    {
        //Ipotizziamo l'esistenza di una view chiamata 'success↩
            '
        //che mostri semplicemente un messaggio di avvenuta ↩
            registrazione
        $this->load->view('success');
    }
}
```

Focalizziamoci sui due punti più importanti:

```
$this->form_validation->set_rules
```

questo comando si occupa di impostare le regole su cui basarsi per il controllo, mentre

```
if ($this->form_validation->run() == FALSE)
```

esegue il controllo del form ed in questo caso verifica che non sia andato a buon fine. In caso di esito negativo infatti ricarica la vista che mostra il modulo html con i relativi messaggi di errore di `validation_errors`, in caso positivo redireziona l'utente ad una pagina che mostrerà un messaggio di avvenuta registrazione.
Soffermiamoci sulla creazione delle regole: il comando accetta tre parametri:

- il nome del campo;
- la relativa label (descrizione) usata negli eventuali messaggi di errore;

- le regole.

Le regole possono essere scelte tra le seguenti che potete trovare a questo indirizzo[8].

4.5 Download assistito di file

Codeigniter offre un comodo strumento per forzare il download di un determinato file da parte di un utente. Per prima cosa, bisogna caricare l'helper adatto:

```
$this->load->helper('download');
```

Ed utilizzare la funzione apposita, con il seguente formato

```
force_download('filename', 'data')
```

Questa funzione genera automaticamente gli header http necessari per forzare un download all'utente. Il primo parametro è il nome che si vuol dare al file da salvare, il secondo parametro sono i dati che si vogliono far scaricare. Quest'ultimo parametro deve essere sempre una stringa; bisogna quindi caricare il contenuto di un file come stringa per effettuarne il download. Vediamo qualche esempio:

```
$data = 'Here is some text!';
$name = 'mytext.txt';

force_download($name, $data);
```

Il seguente codice è necessario nel caso di un file esistente:

```
// Leggo tutto il contenuto del file
$data = file_get_contents("/path/to/photo.jpg");
$name = 'myphoto.jpg';

force_download($name, $data);
```

4.6 Gestione del multilingua

L'internazionalizzazione di un sito di questi tempi è un mattone fondamentale per la diffusione dello stesso. Per questo motivo, CodeIgniter ne offre nativamente il supporto. Di base, le traduzioni della nostra applicazione devono risiedere in una determinata cartella, nominata a seconda della lingua, ad esempio:

[8] http://codeigniter.com/user_guide/libraries/form_validation.html

```
application/language/english
application/language/italian
```

I file di traduzione devono terminare con il suffisso _lang.php e devono essere così strutturati:

```
/**
 * File application/language/italian/prova_lang.php
 */
$lang['chiave_1'] = "Messaggio tradotto 1";
$lang['chiave_2'] = "Messaggio tradotto 2";
//esempio pratico
$lang['welcome_title'] = "Benvenuti!";
```

Per poter essere disponibile a controller o viste, il file di traduzione deve essere caricato:

```
$this->lang->load('filename', 'language');
//seguendo l'esempio precedente
$this->lang->load('prova', 'italian');
```

Il primo parametro della funzione è il nome del file da caricare, senza estensione e senza suffisso; il secondo è la lingua desiderata[9], che coincide con il nome della sottocartella, ad esempio `italian`. Ora carichiamo una delle stringhe tradotte definite in precedenza:

```
echo $this->lang->line('welcome_title');
```

Per una internazionalizzazione completa, è necessario tradurre nelle lingue desiderate, con relative cartelle, anche i messaggi inglesi dei vari componenti di CodeIgniter, che si trovano in

```
system/language/english
```

4.7 File di configurazione

Tutti i file di configurazione base sono collocati nella seguente cartella:

```
application/config/
```

Per creare un proprio file di configurazione (per qualsiasi uso se ne debba fare) basta generare un file all'interno del suddetto percorso, ad esempio `mio_config.php` con un contenuto simile al seguente:

[9]Se omesso, CI utilizzerà la lingua predefinita, impostata in application/config/config.php

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script ↩
    access allowed');

$config['webmaster_email'] = 'admin@example.com';
$config['website_name'] = 'Testing site';

/* End of file mio_config.php */
/* Location: ./application/config/mio_config.php */
```

Per poter accedere alle configurazioni appena impostate è necessario prima caricare il file di configurazione in CodeIgniter, successivamente richiamare la configurazione che ci interessa. Vediamo come:

```
//carico il file senza l'estensione php
$this->config->load('filename');
//stampo ad esempio la voce "website_name"
echo $this->config->item('website_name');
```

4.8 Caricamento automatico di helper, librerie, risorse

Come abbiamo visto in precedenza, librerie, helper, modelli, traduzioni e in generale tutti i tipi di risorse devono essere caricati manualmente in caso di bisogno. Se però una risorsa viene usata spesso è possibile chiedere di caricarla automaticamente e renderla quindi disponibile mediante la modifica del file

`application/config/autoload.php`

Ad esempio, se volessimo caricare automaticamente le librerie di sessione e database, nonché l'helper per il download:

```
$autoload['libraries'] = array('database', 'session');
$autoload['helper'] = array('download');
```

4.9 Upload di files

Il framework mette a disposizione una comoda libreria[10] per la gestione del caricamento di files. Ipotizziamo un form con il seguente codice html:

```
<form method="post" action="<?php echo site_url('upload');?>" ↩
    enctype="multipart/form-data">
```

[10] http://ellislab.com/codeigniter/user-guide/libraries/file_uploading.html

```
<input type="file" name="file_caricato"/>
<input type="submit" value="Carica" />
</form>
```

La libreria predefinita per l'upload deve essere caricata, mediante l'apposita funzione di load, all'interno del controller, con un parametro di configurazione (array) come ad esempio il seguente:

```
$config['upload_path'] = './tmp/';
$config['allowed_types'] = 'gif|jpg|png';
$config['file_name']    = uniqid().'.jpg';
//Caricamento libreria
$this->load->library('upload', $config);
```

In questo parametro strutturato ad array stiamo specificando il path di caricamento, le estensioni di file permesse, il nome del file (unico usando la funzione `uniqid`).

A questo punto non resta che controllare che l'upload sia eseguito, magari mostrando a video il percorso completo del file caricato. Il check upload si esegue con la funzione `do_upload` della libreria, che tenta il caricamento, con parametro il nome del campo di tipo file del form precedente, nel nostro caso "file_caricato". In caso di errore può esserne mostrata la motivazione con la funzione `display_errors`.

```
if ($this->upload->do_upload('immagine'))
{
    //All'interno di $upload_data avrò tutte le informazioni ↩
       relative al caricamento
    $upload_data = $this->upload->data();
    echo $upload_data['full_path'];
}
else
{
    echo $this->upload->display_errors();
}
```

4.10 Manipolazione di immagini

CodeIgniter permette di manipolare le immagini con una apposita libreria[11], eseguendo le seguenti operazioni:

- ridimensionamento
- cropping

[11] http://ellislab.com/codeigniter/user-guide/libraries/image_lib.html

- rotazione
- aggiunta di watermark

In modo simile alla libreria upload, anche la image library viene caricata dal framework mediante l'apposita funzione, con parametro un array di configurazione. A seconda della configurazione utilizzata può essere richiamata una funzione di manipolazione immagini. Ad esempio, per eseguire un ridimensionamento devono essere specificati i parametri width e height:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/mypic.jpg';
$config['maintain_ratio'] = TRUE;
$config['width'] = 75;
$config['height'] = 50;
//Carico la libreria con la configurazione
$this->load->library('image_lib', $config);
//Ridimensiono l'immagine /percorso/mypic.jpg
$this->image_lib->resize();
```

Il listato precedente *ridimensiona* l'immagine sovrascrivendola. Se si preferisce mantenere l'immagine originale, creando una nuova immagine ridimensionata, è possibile utilizzare il parametro di configurazione new_image:

```
$config['new_image'] = '/percorso/nuovaimmaginediversa.jpg';
```

Una *rotazione* di una immagine può essere eseguita con 5 opzioni diverse:

- 90 gradi
- 180 gradi
- 270 gradi
- "hor" (inversione orizzontale)
- "vrt" (inversione verticale)

Di seguito un esempio di codice per l'inversione orizzontale dell'immagine:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/mypic.jpg';
$config['rotation_angle'] = 'hor';

//Carico la libreria con la configurazione
$this->load->library('image_lib', $config);
$this->image_lib->rotate();
```

Per ritagliare una immagine si può utilizzare la funzione textttcrop specificando gli assi X e Y lungo i quali eseguire il cropping:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/mypic.jpg';
$config['x_axis'] = '100';
$config['y_axis'] = '60';
//Carico la libreria con la configurazione
$this->load->library('image_lib', $config);
//Eseguo il cropping
$this->image_lib->crop();
```

L'aggiunta di un watermark (filigrana) può avvenire attraverso due modalità, *text* (aggiunta di un testo all'immagine) e *overlay* (aggiunta di una piccola immagine in overlay all'immagine originale). Vediamo un esempio dell'ultimo caso, con una piccola immagine da inserire in basso a destra, con un padding di 5 pixel dai vertici:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/mypic.jpg';
$config['wm_vrt_alignment'] = 'bottom';
$config['wm_hor_alignment'] = 'right';
$config['wm_padding'] = '5';
$config['wm_overlay_path'] = '/percorso/watermark.png';
//Carico la libreria con la configurazione
$this->load->library('image_lib', $config);
//Eseguo l'aggiunta del watermark
$this->image_lib->watermark();
```

Ora la risposta ad un problema che mi ha fatto sbattere un po' la testa prima di comprenderne la (ovvia) soluzione. Se voglio manipolare due immagini di fila, ma con una configurazione diversa? Notiamo nei listati precedenti come la libreria venga caricata dal metodo `load` con l'array di configurazione, ma questo deve avvenire solo una volta, dopodiché la libreria è già stata caricata all'interno dell'esecuzione di codeigniter. La risposta è nella funzione `clear` ed `initialize`, vediamone un esempio concreto:

```
//Impostazioni cropping
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/mypic.jpg';
$config['x_axis'] = '100';
$config['y_axis'] = '60';
//Carico la libreria con la configurazione
$this->load->library('image_lib', $config);
//Eseguo il cropping
$this->image_lib->crop();

//Cancello le impostazioni della libreria
$this->image_lib->clear();
//Impostazioni per rotazione orizzontale
$config['image_library'] = 'gd2';
$config['source_image'] = '/percorso/mypic.jpg';
```

```
$config['rotation_angle'] = 'hor';

//Carico la libreria con la configurazione
$this->image_lib->initialize($config);
//Eseguo la rotazione
$this->image_lib->rotate();
```

Capitolo 5

Un sistema API in Codeigniter

Ultimamente sempre più progetti web sono strutturati con un frontend generico (sia esso web o applicazione mobile) che va ad interagire con il sistema attraverso delle API[1] richiamate con metodologia AJAX[2], spesso in formato JSON[3]. Questo permette una netta separazione tra vista (lato client) e controllo (lato server), spostando una parte del carico computazionale lato server (per i frontend completamente client-side, ad esempio in javascript, c'è un alleggerimento anche di consumo di banda del server).

In questo capitolo verranno approfondite due modalità di implementazione di API in Codeigniter. La prima, la più semplice, è relativa alla creazione di API che usando esclusivamente il metodo HTTP GET. La seconda, più complessa e completa, parlerà di come sviluppare un sistema di API Rest mediante una libreria di terze parti molto comoda e adatta allo scopo.

5.1 Semplici API GET con risposta JSON

Il nostro framework preferito gestisce in modo nativo le richieste con metodo HTTP GET, quindi creare un semplice sistema di API a sola lettura è molto semplice. Ipotizziamo di creare una sottocartella chiamata `api` nella cartella `controller`, e di posizionarvi dentro i vari controller che rispecchieranno le risorse delle API

```
application
|- controllers
    |- api
      |- news.php
```

[1] http://it.wikipedia.org/wiki/Application_programming_interface
[2] http://it.wikipedia.org/wiki/AJAX
[3] http://www.json.org/

A questo punto sappiamo perfettamente che richieste del tipo

```
http://www.example.com/api/news/getAll
http://www.example.com/api/news/getSingle/12
```

Richiederà l'esecuzione del controller `news.php` che compare nell'albero di esempio, con i metodi `getAll` senza parametri e `getSingle` con un parametro GET valorizzato a "12". Vediamo quindi il codice del controller

```
<?php

class News extends CI_Controller {

    public function getAll()
    {
        $query = $this->db->get('news');
        $elenco_news = array();
        foreach ($query->result() as $row)
        {
             $news = new stdClass();
             $news->id = $row->id;
             $news->titolo = $row->titolo;
             $news->contenuto = $row->contenuto;
                 array_push($elenco_news, $news);
        }
        echo json_encode($elenco_news);
    }

    public function getSingle(id)
    {
        $this->db->from('news')->where('id',$id);
        $query = $this->db->get();
        $row = $query->row();
        $news = new stdClass();
        $news->id = $row->id;
        $news->titolo = $row->titolo;
        $news->contenuto = $row->contenuto;
        echo json_encode($news);
    }

}
```

Sarà sufficiente quindi creare un controller per ogni tipo di risorsa (ad esempio news, eventi, banner) ed i relativi metodi, ricordandosi di strutturare le risposte in formato JSON come da esempio.

5.2 Un sistema API complesso con codeigniter-restserver

Nella sezione precedente abbiamo compreso come creare un semplice sistema API con il solo metodo GET. Ma se volessimo un server API che risponda secondo lo standard RESTful[4], dovremmo gestire anche i metodi POST, PUT e DELETE. In particolare, lo standard consiglia di utilizzare:

- GET per richieste di lettura di una risorsa
- POST per richieste di creazione di una risorsa
- PUT per richieste di modifica di una risorsa
- DELETE per richieste di eliminazione di una risorsa

Per fare questo velocemente è possibile appoggiarsi ad una libreria di terze parti, codeigniter-restserver[5].

5.2.1 Installazione

Il primo passo è ovviamente scaricare il pacchetto dalla pagina github della libreria. Il pacchetto includerebbe anche CodeIgniter, ma è molto probabile che si voglia integrare *restserver* in un progetto codeigniter già esistente. In questo caso è sufficiente copiare i files

```
application/libraries/Format.php
application/libraries/REST_Controller.php
application/config/rest.php
```

nella propria directory `application`, e ricordarsi di caricare la classe REST_Controller. Al contrario di quanto troverete nella guida di installazione di restserver, non è possibile caricare automaticamente la classe inserendola nel file autoload.php di codeigniter, ma si dovrà includere in ogni controller che la andrà ad utilizzare tramite la funzione php *include* oppure, per semplicità, modificare il file config.php di CodeIgniter per supportare il prefisso *REST*

```
//application/config/config.php
$config['subclass_prefix'] = 'REST_';
```

Per personalizzare le opzioni di *restserver*, basta modificare l'apposito file precedentemente copiato:

```
application/config/rest.php
```

[4] http://en.wikipedia.org/wiki/Representational_state_transfer
[5] https://github.com/philsturgeon/codeigniter-restserver

5.2.2 Funzionamento di base

Per il buon funzionamento del sistema è necessario creare controller che estendono la classe base REST_Controller. La libreria elabora le richieste sulla base del metodo HTTP utilizzato e della risorsa (controller), eseguendo la funzione corrispondente all'insieme dei due elementi, sempre seguendo la logica CodeIgniter. Per spiegarmi meglio, ipotizziamo di avere il seguente controller:

```
class News extends REST_Controller
{
    public function index_get()
    {
        // Lettura delle news
    }

    public function index_post()
    {
        // Crea una news
    }

    public function index_put()
    {
        // Modifica una news
    }
    public function index_delete()
    {
        // Elimina una news
    }
}
```

Una richiesta del tipo

```
GET http://www.example.com/news
```

comporterà l'esecuzione della funzione `index_get`, index perché l'url è senza una funzione specificata (news è direttamente il controller) e get perché il metodo HTTP utilizzato è, appunto, GET.

L'accesso ai parametri è garantito dalle funzioni get, post e put.

```
$this->get('id'); //mappatura di $this->input->get()
$this->post('id'); //mappatura di $this->input->post()
$this->put('id');
```

Per i parametri del metodo DELETE, poiché lo standard non li prevede, è sufficiente gestirli direttamente dalla funzione richiamata dal controllore:

```
public function index_delete($id)
    {
        $this->response(array(
```

```
            'returned from delete:' => $id,
        ));
    }
```

E' possibile inviare in output una struttura dati con la funzione *response*, gestendo anche la risposta HTTP direttamente (opzionale). In questo modo la libreria si preoccuperà di formattare la risposta a seconda dello standard impostato nella configurazione (io consiglio JSON). Inoltre, gestendo le risposte HTTP, è possibile utilizzare codici appropriati come il 201 per *HTTP 201 Created* e 404 per *HTTP 404 Not Found* (in quest'ultimo caso il primo parametro sarà una risorsa vuota, come ad esempio un array senza elementi).

```
public function index_post()
{
    // ...crea una news

    $this->response($news, 201); // Manda la risposta HTTP 201
}
```