Estendere Drupal creando moduli personalizzati

Drupal è un sistema estremamente flessibile e dispone di moltissimi moduli aggiuntivi, ma può capitare di doverne scrivere uno ad-hoc...

Andrea Franceschini
a.franceschini@oltrelinux.com

Laureato in Ingegneria Informatica a Padova, appassionato di informatica a tutto tondo, in particolare di musica, grafica ed elaborazione di immagini e suoni. Programma in C/C++, PHP, varie ed eventuali, fotografa e disegna a tempo perso

Nella scorsa puntata abbiamo approfondito lo sviluppo di un sito web basato su Drupal realizzando, un template personalizzato ed imparando ad usare alcuni moduli estremamente importanti, come **Views**.
In questa puntata vedremo come sviluppare un nostro modulo da zero: nonostante ne esistano centinaia, infatti, può capitare di non riuscire ad utilizzarne uno, per esempio perché richiede dipendenze esterne non disponibili sul nostro hosting, oppure di trovarsi di fronte ad un problema che non era mai stato affrontato. Può anche capitare, più comunemente, di dover modificare un modulo esistente che fa *quasi* quel che ci serve, ma non del tutto. In tal caso sarà utile sapere dove mettere le mani: potremmo contribuire allo sviluppo, come in ogni progetto Open Source che si rispetti!

Requisiti

Per il nostro esempio abbiamo deciso di usare un classico problema di integrazione: immaginiamo di avere un database di **impiegati** e **dipartimenti** già esistente e gestito, per quanto riguarda inserimento, modifica e cancellazione, dall'ufficio delle Risorse Umane, usando un software gestionale. La direzione vuole, però, che questi dati siano consultabili attraverso il sito web aziendale: dovremo fornire una lista dei dipartimenti, una degli impiegati, una scheda di ciascun dipartimento con la lista degli impiegati che vi lavorano ed una scheda personale per ciascun impiegato. Tutti questi dati sono già nel database, ma il gestionale non è in grado di fornirceli in qualche formato, come ad esempio XML, comodo da gestire per via programmatica quindi non ci rimane che realizzare un accesso diretto alla base dati.
Considerando che i dati mostrati dal sito dovranno essere sempre allineati con quelli nel database, decidiamo di non seguire la classica filosofia di Drupal, per cui le sorgenti dati esterne dovrebbero essere trasformate in *nodi*. Tale modo di procedere, però, implica l'importazione dei dati nel DB di Drupal, con la perdita di eventuali modifiche successive apportate al database originale, quindi nel nostro caso non andrebbe bene.
Lavorare senza seguire tale convenzione ci precluderà l'accesso a comodità come la possibilità sfruttare *Views*, ma ci garantirà di avere sempre i dati corretti evitando duplicazioni.

Database

Cominciamo a dare uno sguardo al database. Nel listato 1 troviamo lo schema sotto forma di due query SQL che creano le tabelle `employees` e `departments` contenenti i dati degli impiegati (nome, cognome, indirizzo e-mail e scheda personale) e dei dipartimenti (nome e descrizione). Nell'esempio abbiamo detto che il database sarebbe pre-esistente, ma abbiamo aggiunto alcune insert per darvi la possibilità di avere qualcosa su cui lavorare mentre seguite lo svolgimento dell'articolo.
Gli attributi `id` di entrambe le tabelle sono interi progressivi e li useremo come parametri negli URL, per consultare i dati via web. Infine i campi `description` contengono un frammento di HTML già formattato per cui, previo filtraggio per eliminare rischi di sicurezza, potremo usarlo direttamente nelle schede di impiegati e dipartimenti.

Dato che non abbiamo ancora alcun server che contiene quelle tabelle, prima di lanciarci nello sviluppo del modulo sarà bene creare il database in questione e configurare Drupal per accederci: per prima cosa colleghiamoci al nostro DB – può andare bene lo stesso su cui abbiamo installato Drupal – usando il comando:

```
$ mysql -u root -p
```

e creando un database diverso da quello di Drupal con il comando

```
mysql> CREATE DATABASE employees;
```

Di seguito ricopiamo le query del listato 1 e popoliamo le tabelle appena create con un po' di dati. Gli `id` non sono auto incrementanti, quindi dovremo inserirli a mano facendo attenzione a rispettare i relativi vincoli di chiave primaria di ciascuna tabella. C'è anche una chiave esterna per la tabella degli impiegati: per seguire l'articolo non è fondamentale, ma se volete mantenerla, andrà creata prima la tabella dei dipartimenti e, una volta popolata quella, si potrà procedere con quella degli impiegati.

A questo punto dobbiamo informare Drupal che desideriamo instaurare una connessione ad un secondo database. Apriamo il file `settings.php` del nostro sito (che sarà in `sites/default/` se abbiamo seguito le istruzioni delle due scorse puntate) e cerchiamo il punto in cui configuriamo il database. Dovremmo trovare una variabile chiamata `$db_url` a cui assegniamo la stringa di connessione al database. Modifichiamo l'assegnamento in modo simile a

```
$db_url = array(
  'default' => "mysqli://user1:pass1@localhost/drupal",
  'employees' => "mysqli://user2:pass2@localhost/employees"
);
```

così avremo la connessione di `default` per il normale funzionamento di Drupal ed `employees` per il nostro modulo.

Struttura del modulo

Un modulo di Drupal è normalmente racchiuso in una directory sotto `modules/` nel percorso generico (`sites/all/modules/`) o in quello di uno specifico sito (`sites/default/modules/`). Nel nostro caso chiameremo questa directory `employeedb` ed al suo interno creeremo due file, `employeedb.info` e `employeedb.module`. Troviamo il contenuto del primo nel listato 2: si specificano nome e descrizione del modulo, che appariranno nell'interfaccia di gestione dei moduli di Drupal e la versione del core con cui questo modulo è compatibile.

L'altro file, `employeedb.module`, come avrete ormai intuito, è quello più interessante. Lo trovate riportato nei listati che popolano il resto dell'articolo: al suo interno, per prima cosa incontriamo la funzione `employeedb_help()` (listato 3) che fornisce informazioni più estese circa le funzionalità del nostro modulo, quelle che ci vengono restituite nella pagina all'URL `/admin/help/employeedb/`. L'esempio è molto breve per motivi di spazio, ma fornire una documentazione "in linea" tramite questa funzione è molto importante se vogliamo rendere pubblico un nostro modulo.

L1 – Tabelle per il DB

```
use employees;
CREATE TABLE `departments` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `description` text NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `employees` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `email` varchar(50) NOT NULL,
  `description` text NOT NULL,
  `department_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_Employee_Department` (`department_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO departments VALUES(1, 'Amministrazione',
'Amministrazione ordinaria');
INSERT INTO departments VALUES(2, 'Commerciale',
'Operativi acquisti e vendite');
INSERT INTO departments VALUES(3, 'Assistenza clienti',
'Customer care');

INSERT INTO employees VALUES(1, 'Mario', 'Rossi',
'mario.rossi@example.org', 'CEO', 1);
INSERT INTO employees VALUES(2, 'Sara', 'Bianchi',
'sara.bianchi@example.org', 'COO', 1);
INSERT INTO employees VALUES(3, 'Elisa', 'Verdi',
'elisa.verdi@example.org', 'Assistenza clienti VIP', 3);
INSERT INTO employees VALUES(4, 'Sonia', 'Russo',
'sonia.russo@example.org', '', 3);
INSERT INTO employees VALUES(5, 'Marco', 'Rossi',
'marco.rossi@example.org', '', 2);
INSERT INTO employees VALUES(6, 'Davide', 'Bianchi',
'davide.bianchi@example.org', '', 2);
INSERT INTO employees VALUES(7, 'Simona', 'Verdi',
'simona.verdi@example.org', '', 2);
INSERT INTO employees VALUES(8, 'Luca', 'Russo',
'luca.russo@example.org', '', 2);
```

L2 - File di configurazione

```
name = External Employees DB
description = A module to retrieve employees' data from
an external database.
core = 6.x
```

L3 - funzioni help, init, perm

```
function employeedb_help($path, $arg) {
  $output = '';
  switch($path) {
    case 'admin/help#employeedb':
    $output = '<p>' . t('Retrieves employees\'s data from
                                  an external database') . '</p>';
    break;
  }
  return $output;
}

function employeedb_init() {
  drupal_add_css(drupal_get_path('module',
                                 'employeedb') . '/employeedb.css');
}

function employeedb_perm() {
  $permissions = array('view employees');
  return $permissions;
}
```

L4 - employeedb_menu

```
function employeedb_menu() {
  $items = array();

  $items['employee'] = array(
    'title' => 'Employees',
    'page callback' => 'employeedb_employee_list',
    'access arguments' => array('view employees'),
    'type' => MENU_CALLBACK
  );
  $items['employee/%'] = array(
    'title callback'   => 'employeedb_employee_page_title',
    'title arguments'  => array(1),
    'page callback'    => 'employeedb_employee_page',
    'access arguments' => array('view employees'),
    'type' => MENU_CALLBACK,
    'page arguments'   => array(1)
  );
  $items['department'] = array(
    'title'  => 'Departments',
    'page callback' => 'employeedb_department_list',
    'access arguments' => array('view employees'),
    'type' => MENU_CALLBACK
  );
  $items['department/%'] = array(
    'title callback' => 'employeedb_department_page_title',
    'title arguments' => array(1),
    'page callback' => 'employeedb_department_page',
    'access arguments' => array('view employees'),
    'type' => MENU_CALLBACK,
    'page arguments' => array(1)
  );
  return $items;
}
```

Prima di continuare, un appunto: Drupal mette a disposizione un'enorme quantità di *hook* per lo sviluppo dei moduli e starà a noi effettuarne l'*override* per implementare le funzionalità che ci interessano. I nomi delle funzioni sono del tipo `hook_theme()` e quando le vogliamo sovrascrivere nei nostri moduli dovremo nominare le nostre sostituendo a *hook* il nome del nostro modulo, nel nostro caso `employeedb`. Ad esempio, abbiamo creato la funzione per specificare i template all'interno di `employeedb.module` chiamandola `employeedb_theme()` (listato 5). Ogni volta che aggiungiamo una funzione di questo tipo al nostro modulo (e in alcune altre occasioni, ad esempio quando modifichiamo lo schema degli URL come vedremo tra poco) dobbiamo cancellare la cache di Drupal. Come abbiamo visto nella scorsa puntata, il modo più veloce per farlo è usare Drush (chi non si ricorda cos'è può far riferimento, appunto, alla scorsa puntata o alla guida ufficiale reperibile all'URL http://drupal.org/project/drush) usando il comando `drush cc all` dall'interno della directory del nostro sito (per esempio `sites/default/`) o specificando l'URL a riga di comando con:

```
$ drush cc all -l http://localhost/drupal-6.20
```

Se ci troviamo impossibilitati ad usare la riga di comando, possiamo andare all'URL /admin/settings/performance e fare clic sul pulsante *Eliminare i dati della cache*.

Permessi del modulo

La prossima funzione interessante è `employeedb_perm()` (listato 3) con questa, dichiariamo la lista dei permessi relativi al modulo. Avremmo anche potuto evitarlo, data la semplicità dell'esempio, ma questo è un argomento molto importante ed è bene prenderci confidenza: Drupal si aspetta da ciascun modulo un array di stringhe che userà come chiavi per determinare quali *permessi* sono concessi a quali *ruoli*. Nel nostro caso inseriamo un solo livello di permessi, `view employees`.

Per vedere i permessi in azione dobbiamo prima di tutto attivare il nostro modulo. Andiamo all'URL /admin/build/modules, cerchiamolo nella lista ed attiviamolo selezionando la checkbox corrispondente e facendo clic su *Salva configurazione* a fondo pagina. A questo punto andiamo all'URL /admin/user/permissions e cerchiamo il nostro

modulo nella lista. L'unico permesso esposto è, `view employees`: assegnamolo a tutti i ruoli in modo da consentire a tutti i visitatori di consultare le informazioni su impiegati e dipartimenti.

URL di accesso al modulo

Proseguendo, incontriamo la funzione `employeedb_menu()` (listato 4) che serve a definire lo schema di URL a cui il nostro modulo deve rispondere e le eventuali voci del menu *Navigazione* o le operazioni della pagina corrente a cui questi possono essere associati.
Seguendo i requisiti, predisponiamo:

- `/employee` per ottenere la lista degli impiegati – notiamo la forma singolare, coerente con le convenzioni di Drupal, per esempio nel caso della lista dei nodi pubblicati accessibile all'URL `/node`;
- `/employee/%` per accedere alla scheda di un impiegato specifico, attraverso il passaggio di un parametro – anche in questo caso siamo coerenti con le convenzioni di Drupal, per esempio nel caso del singolo nodo accessibile con l'URL `/node/<id>`;
- `/department` presenta la lista dei dipartimenti;
- `/department/%` ci mostra la scheda di uno specifico dipartimento e la lista dei dipendenti che vi lavorano.

Per ciascun URL specifichiamo una funzione (*page callback*) che dovrà produrre il contenuto della pagina, la lista dei permessi che gli utenti devono avere per accedere a quel URL (*access arguments*) ed il tipo di elemento che stiamo descrivendo. Infatti, come abbiamo detto, con questa funzione possiamo anche creare voci del menu *Navigazione* (tipo *Profilo*) o le operazioni di pagina (tipo *Mostra* e *Modifica*), come vediamo in figura 1.
Inoltre specifichiamo un titolo della pagina per gli URL senza parametri. Quelli con parametri sfrutteranno una funzione che lo produrrà dinamicamente (*title callback*). Specifichiamo anche la lista degli argomenti che vogliamo fornire a questa callback ed a quella che produrrà il contenuto della pagina (*title arguments* e *page arguments*), che vedremo più avanti.
L'ultima tra le funzioni fondamentali per la buona riuscita del modulo è `employeedb_theme()` (listato 5) con cui diciamo a Drupal quali template di default forniamo (che di conseguenza saranno sovrascrivibili dai temi personalizzati) e quali variabili vogliamo rendergli disponibili. Nel nostro caso specifichiamo:

- *lista degli impiegati*, a cui passeremo un array associativo contenente nome, cognome, nome del dipartimento e URL della scheda personale di ciascun impiegato;
- *scheda impiegato*, a cui passeremo un array associativo contenente i dati (nome, cognome, etc) del singolo impiegato;
- *lista dei dipartimenti*, a cui passeremo un array associativo con il nome del dipartimento e l'URL per accedere alla sua descrizione;
- *scheda dipartimento*, a cui passeremo un array associativo contenente i dati del dipartimento ed uno contenente la lista degli impiegati che vi lavorano, in un formato analogo a quello della lista di tutti gli impiegati.

Va notato che le chiavi dell'array restituito da `employeedb_theme()`

L5 – employeedb_theme

```
function employeedb_theme() {
  return array(
    'employee_list' => array(
      'template'  => 'employee-list',
      'arguments' => array('employees' => array())
    ),
    'employee_page' => array(
      'template'  => 'employee-page',
      'arguments' => array('employee' => array())
    ),
    'department_list' => array(
      'template'  => 'department-list',
      'arguments' => array('departments' => array())
    ),
    'department_page' => array(
      'template'  => 'department-page',
      'arguments' => array('department' => array(),
                           'employees' => array())
    )
  );
}
```

Mostra e Modifica, due delle operazioni di default che è possibile effettuare su una pagina Drupal

1

(employee_list, employee_page, etc) non hanno alcuna correlazione con quelle della chiave template (employee-list, employee-page, etc): la chiave (per esempio employee_list) sarà usata da noi internamente al modulo quando dovremo invocare la funzione theme() per dire a Drupal quale configurazione vogliamo usare, mentre il *valore* della chiave all'interno di template specifica il nome del file con estensione .tpl.php che vogliamo usare. Avremo, ad esempio, un file chiamato employee-list.tpl.php per mostrare la lista degli impiegati recuperata dalla callback employeedb_employee_list(), ma questo lo vedremo tra poco. L'importante è ricordare che la convenzione di Drupal vorrebbe che usassimo trattini (-) e non underscore (_) per i nomi dei file, mentre non dice alcunché per quanto riguarda le chiavi – e infatti usiamo gli underscore, per non creare confusione tra le due.

L6 - employee_page_title

```
function employeedb_employee_page_title($eid = -1) {
  db_set_active('employees');
  $result = db_query('SELECT employees.name AS name,
                      employees.surname AS surname FROM employees
                                    WHERE employees.id = %d', $eid);
  db_set_active('default');
  $title = '';
  if($result != false) {
    $row = db_fetch_array($result);
    $title = filter_xss($row['name'],
           array()) . ' ' . filter_xss($row['surname'], array());
  }
  return $title;
}
```

L7 - employeedb_department_list

```
function employeedb_department_list() {
  db_set_active('employees');
  $result = db_query('SELECT departments.id AS id,
                          departments.name AS name FROM departments
                                    ORDER BY departments.name ASC');
  db_set_active('default');
  $departments = array();
  if($result == false) {
    drupal_set_message('Error in query', 'error');
  } else {
    if($result->num_rows == 0) {
      drupal_set_message('No departments found in the
                          external database.', 'employeedb_error');
    } else {
      while($row = db_fetch_array($result)) {
        $department = array(
          'id' => (int)$row['id'],
          'name' => filter_xss($row['name'], array()),
          'url' => url('department/' . $row['id'],
                                    array('absolute' => true)),
        );
        $departments[] = $department;
      }
    }
  }
  return theme('department_list', $departments);;
}
```

Recuperare i dati

A questo punto possiamo passare ad esaminare le funzioni che verranno invocate ogni volta che accederemo agli URL del nostro modulo. Prendiamo ad esempio employeesdb_department_list(), (listato 7) che inizia selezionando la nostra connessione al database con la funzione db_set_active() dell'API di Drupal. Nel caso specifico, selezioniamo il database degli impiegati, per cui tutte le funzioni della DB API che chiameremo da quel punto in poi, faranno riferimento a tale connessione. Dopo aver effettuato la query che recupera la lista dei dipartimenti, ripristiniamo la connessione di default per permettere a Drupal di proseguire con le sue operazioni. È buona norma mantenere attive le connessioni personalizzate solo per il tempo necessario e compattare il più possibile le query da effettuare su ciascuna, in modo da minimizzare i cambi di contesto, come abbiamo fatto all'inizio di employeedb_department_page(), che vedremo tra poco.

Seguono le verifiche sul risultato della query: il primo verifica che non sia false. In tal caso, infatti, significherebbe che c'è stato un errore nell'esecuzione della query e dovremmo segnalarlo usando i messaggi di Drupal. Abbiamo già incontrato qualcuno di questi messaggi, per esempio quelli *verdi* che indicano che un'operazione ha avuto successo. In questo caso, però, vogliamo emettere un messaggio di errore: possiamo farlo accodando il parametro error.

Se la query non ha restituito errori, dobbiamo verificare che abbia restituito almeno un record e, in caso contrario, informare il visitatore usando un altro messaggio. Questa volta passeremo un valore personalizzato, employeedb_error, che verrà aggiunto come classe CSS al contenitore del messaggio, così potremo personalizzarne l'aspetto tramite i fogli di stile allegati al modulo. Le linee guida, infatti, prevedono che i moduli possano fornire dei CSS (che possono essere a loro volta sovrascritti dagli autori dei temi), quindi nella funzione di inizializzazione del modulo employeedb_init() (listato 3), eseguita ogni volta che il modulo entra in azione, richiamiamo drupal_add_css() per includere il file employeedb.css. Se vogliamo includere un CSS specifico solo per una certa pagina dovremo farlo nella callback relativa ad essa.

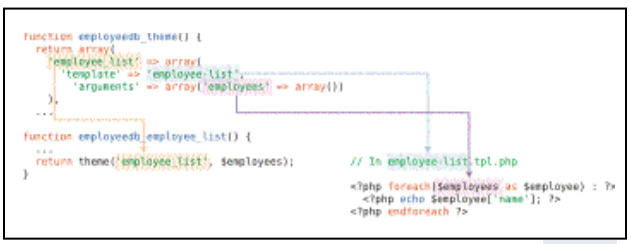
È importante fare attenzione a come si includono i CSS perché una politica poco accorta potrebbe influre sull'eventuale raggruppamen-

L8 - department_page

```
function employeedb_department_page($did = -1) {
  db_set_active('employees');
  $departments_result = db_query('SELECT departments.id AS id,
                                  departments.name AS name,
                                  departments.description AS description
                           FROM departments WHERE departments.id = %d', $did);
  $employees_result = db_query('SELECT employees.id AS id,
          employees.name AS name, employees.surname FROM employees
                                  WHERE employees.department_id = %d
                           ORDER BY employees.surname ASC', $did);
  db_set_active('default');
  $department = array();
  $employees = array();

  if($departments_result == false) {
    drupal_set_message('Error in query for department.', 'error');
  } else {
    if($departments_result->num_rows == 0) {
      drupal_not_found();
      return;
    } else {
      $row = db_fetch_array($departments_result);
      $department = array(
        'id' => $row['id'],
        'name' => filter_xss($row['name'], array()),
        'description' => filter_xss($row['description']),
      );

      if($employees_result == false) {
        drupal_set_message('Error in query
                                        for employees.', 'error');
      } else {
        while($row = db_fetch_array($employees_result)) {
          $employee = array(
            'id' => $row['id'],
            'name' => filter_xss($row['name'], array()),
            'surname' => filter_xss($row['surname'], array()),
            'url' => url('employee/' . $row['id'],
                                        array('absolute' => true))
          );
          $employees[] = $employee;
        }
      }
    }
  }
  drupal_set_breadcrumb(array(
    l('Home', '<front>'),
    l('Departments', 'department')
  ));
  return theme('department_page', $department, $employees);
}
```

L9 - department_page_title

```
function employeedb_department_page_title($did = -1) {
  db_set_active('employees');
  $result = db_query('SELECT departments.name AS name FROM
departments WHERE departments.id = %d', $did);
  db_set_active('default');

  $title = '';
  if($result != false) {
    $row = db_fetch_array($result);
    $title = filter_xss($row['name'], array());
  }

  return $title;
}
```



Relazioni tra nomi di variabili, template e funzioni

2

to e caching di Drupal. Infatti, per ridurre il numero di chiamate HTTP e quindi ridurre i tempi di caricamento delle pagine, Drupal cerca di compattare in un unico file tutti i rimandi esterni esterni inclusi dalle pagine: tutti i file Javascript finiranno concatenati in un unico file e la stessa sorte toccherà ai CSS. Idealmente, Drupal dovrebbe poter generare un solo file JS ed un solo CSS per tutte le pagine del sito, ma questo vorrebbe dire che in molti casi staremmo includendo porzioni di JS o CSS non necessarie, incrementando la dimensione del file e quindi diminuendo l'efficacia del meccanismo. Se includessimo un CSS diverso per ogni pagina del nostro modulo, Drupal sarebbe costretto a creare ogni volta un file diverso da mettere in cache, col risultato di replicare la maggior parte del CSS (tutto, tranne le regole specifiche per quella pagina) ed aumentare la dimensione della cache e, di conseguenza, dei download per gli utenti del sito. Di conseguenza, è consigliabile ridurre al minimo le differenze tra le varie pagine del modulo e raggruppare tutte le regole CSS comuni in un singolo file.

Tornando alla nostra funzione, se tutto è andato per il verso giusto, a questo punto possiamo produrre l'array che passeremo al template engine. In questo caso ci interessa passare l'`id`, il nome del dipartimento e l'URL per accedere alla sua pagina. Le prime due informazioni le recuperiamo dal database: effettuiamo un cast a `int` per quanto riguarda l'`id` (di default in uscita dal database sarebbe una stringa, ma PHP ci consente di trasformarlo in intero semplicemente forzandone il tipo) e filtriamo il nome del dipartimento usando la funzione di Drupal `filter_xss()` passandogli come parametro la stringa da filtrare ed un array vuoto, che indica che non desideriamo alcun tag HTML in uscita. Per una panoramica sugli attacchi di

tipo *Cross-Site Scripting* (XSS) vi rimandiamo al riquadro 1.

Componiamo l'URL della scheda usando la funzione `url()` di Drupal, a cui passeremo l'indirizzo interno relativo (`department/<id>`) ed un array di opzioni con la chiave `absolute` valorizzata a `true` per ottenere l'URL assoluto pronto per essere usato in un tag `<a>`.

Infine chiamiamo la funzione `theme()` passandogli come parametro il nome del template (la chiave dell'array che abbiamo restituito in `employeedb_theme()`) e l'array con la lista dei dipartimenti. Questa funzione accetta un numero variabile di parametri, oltre al template da usare ed è venuto il momento di chiarire il contenuto dell'array che abbiamo restituito da `employeedb_theme()`. In particolare, le chiavi `arguments` contengono un array che associa i nomi delle variabili che avremo a disposizione all'interno del template con i default da utilizzare in caso non fossero specificate (figura 2).

Le altre callback sono molto simili a questa, ma ci sono alcune piccole differenze in `employeedb_department_page()` (listato 3). Per cominciare, questa funzione ha un parametro `$did` che viene estratto dall'URL `department/%` che abbiamo dichiarato in `employeedb_menu()` (listato 4). In questa pagina abbiamo deciso di mostrare sia le informazioni sul dipartimento in questione che la lista degli impiegati che vi lavorano, perciò dovremo fare *due* query. Ricordando che è meglio raggrupparle, all'inizio della funzione attiviamo il database esterno ed effettuiamo le due query, dopodiché riattiviamo quello di default. A questo punto verifichiamo se ci sono stati errori. La prima differenza che notiamo è che, se non ci è stata restituita alcuna riga, chiamiamo la funzione `drupal_not_found()` (che genera un errore 404) ed usciamo immediatamente dalla funzione. Questo ha senso, perché la pagina del dipartimento è da considerare un documento e, nel caso in cui non sia presente, il protocollo HTTP prevede la restituzione di un errore *404 Not Found*.

Proseguendo, notiamo un'altra piccola differenza: il campo `description` che recuperiamo dal database è già formattato in HTML e procediamo a filtrarlo, tuttavia in questo caso non passiamo un array vuoto, accettando implicitamente il default (già piuttosto restrittivo) predefinito di Drupal, come spiegato nel riquadro 1. Per quanto riguarda la lista degli impiegati, controlliamo solo se c'è stato un errore nella query, dato che se non ci sono dipendenti semplicemente restituiremo l'array vuoto e sarà cura di chi creerà il template gestire la situazione.

L'ultima differenza notevole viene prima della `return` ed è la chiamata alla funzione `drupal_set_breadcrumb()` che ci permette di generare un percorso ideale che porta dalla home page al documento corrente, utile soprattutto per costruire un senso di *struttura del sito*. In questo caso abbiamo l'elemento *Home page* e quello *Dipartimenti*, che riconduce alla lista dei dipartimenti. Va da sé che i percorsi per giungere ad una pagina siano in realtà molteplici, ad esempio possiamo esaminare il file `employee-page.tpl.php`, presente nei sorgenti a disposizione su http://www.oltrelinux.com/risorse/LC75/

R1 - Cross-Site Scripting

Stando al rapporto 2007 di Symantec, gli attacchi XSS, o Cross-Site Scripting si attestano attorno al 70-80% del totale delle violazioni verso siti web, ma di cosa si tratta, esattamente?

Si tratta di una tecnica di attacco diretta ai visitatori di un sito web per cui l'attaccante riesce, in modo permanente o in modo temporaneo, ad inserire nella pagina che si sta visitando del codice arbitrario non previsto dal gestore del sito, con lo scopo di sfruttare vulnerabilità sia del browser che delle stesse persone per ottenere un'enorme varietà di effetti: dal crash del browser al dirottamento della navigazione verso un suo sito fasullo, dall'intercettare dati che vengono inseriti nelle form da utenti ignari, tipo password e carte di credito, allo sfruttamento di vulnerabilità in certi browser per arrivare addirittura a prendere il controllo del sistema operativo dell'utente per eventualmente installare malware di vario tipo.

Detta così può sembrare una situazione catastrofica e, all'apparenza, la varietà di effetti può farci pensare che le contromisure da adottare siano tantissime, ma la base dell'attacco è una sola: modificare o inserire codice HTML malevolo, magari per includere Javascript ospitati da altri siti (da cui il nome Cross-Site Scripting). A questo punto è chiaro che il rimedio, oltre eventualmente a tappare le vulnerabilità che hanno consentito all'attaccante di mettere in atto la prima fase, è eliminare il codice malevolo prima che giunga al browser del visitatore. La funzione filter_xss() di Drupal fa esattamente questo: prende in input un frammento di HTML e lo ripulisce eliminando tutti i tag che possono causare danni. Il motivo per cui è bene effettuare questa operazione il prima possibile è che, col passare dei livelli, potremmo distruggere tag che invece servono al buon funzionamento del sito, per esempio rimuovendo l'inclusione di jQuery, un framework Javascript che Drupal usa per rendere più reattive certe pagine, specialmente dell'interfaccia di amministrazione.

Nel nostro caso, cosa succederebbe se un attaccante avesse accesso al gestionale delle Risorse Umane e inserisse codice malevolo nelle descrizioni di dipartimenti o impiegati, o addirittura nei nomi e cognomi? Chiaramente un disastro, ma noi che siamo attenti e scrupolosi filtriamo alcuni elementi (ad esempio nomi e cognomi) in modo estremamente restrittivo (passando come parametro a filter_xss() un array vuoto come "tag consentiti" di fatto vietiamo ogni tipo di tag) e in modo più rilassato, ma sempre estremamente efficace, chiamando la funzione col solo frammento di HTML da filtrare e fidandoci del secondo parametro passato di default da Drupal che, per chi è curioso, corrisponde a lasciar passare i tag a, em, strong, cite, code, ul, ol, li, dl, dt e dd.

L10 - template department

department-list.tpl.php

```
<ul>
  <?php foreach($departments as $department) : ?>
    <li><a href="<?php echo $department['url'] ?>">
      <?php echo $department['name'] ?></a></li>
  <?php endforeach ?>
</ul>
```

department-page.tpl.php

```
<div class="description">
  <?php echo $department['description'] ?>
</div>

<?php if(!empty($employees)) : ?>
<div class="employees">
  <h2><?php echo t('Employees') ?></h2>
  <ul>
    <?php foreach($employees as $employee) : ?>
      <li><a href="<?php echo $employee['url'] ?>">
        <?php echo $employee['surname'] . ', '
        . $employee['name'] ?></a></li>
    <?php endforeach ?>
  </ul>
</div>
<?php endif ?>
```

L11 - template employee

employee-list.tpl.php

```
<ul>
  <?php foreach($employees as $employee) : ?>
    <li><a href="<?php echo $employee['url'] ?>">
      <?php echo $employee['surname'] . ', '
      . $employee['name'] ?></a> (<?php echo
      $employee['department_name'] ?>)</li>
  <?php endforeach ?>
</ul>
```

employee-page.tpl.php

```
<p><strong>Department: </strong>
<a href="<?php echo $employee['department_url'] ?>">
  <?php echo $employee['department_name'] ?></a></p>

<div class="description">
  <?php echo $employee['description'] ?>
</div>
```

Webografia

- http://www.
  Lor...
- http://www.
  Lor...
- http://www.
  Lor...

che rappresenta la scheda degli impiegati, da cui è possibile accedere a quella del dipartimento d'appartenenza tramite un link.

Ultima nota sulla chiamata alla funzione `theme()`: in questo caso viene chiamata con due parametri aggiuntivi: se torniamo alla specifica in `employeedb_theme()` notiamo che infatti abbiamo indicato che questo template avrà due variabili, `$department` e `$employees`, entrambi array, vuoti per default.

Concludendo

In questo articolo abbiamo dato una rapida panoramica sullo sviluppo di moduli per Drupal, descrivendo la struttura base di un modulo e qualche funzione di accesso diretto ai database. Non abbiamo visto molte delle operazioni più comuni, come l'interazione con altri moduli, la gestione dei nodi o altre cose di questo genere, ma tali temi sono trattati ampiamente nella documentazione ufficiale, mentre c'è carenza di risposte a situazioni più complesse, come quella che abbiamo presentato: spesso nella documentazione ufficiale e nei forum si tende a suggerire di importare tutto in Drupal e gestirlo con i moduli già esistenti, ma non sempre, come abbiamo visto, questa è la soluzione migliore.

Ovviamente non vale neanche l'opposto: è inutile scrivere nuovi moduli, quando se ne può usare o integrare qualcuno già pronto, ma è importante sottolineare che, a volte, non seguire *"la Via di Drupal"* è una cosa perfettamente accettabile o, come in questo caso, consigliabile. In tutti gli altri casi, si può fare riferimento ai moduli ed alla documentazione esistenti.

Come "compiti per le vacanze" potremmo immaginare di dover dare seguito alla "richiesta della direzione" che vorrebbe rendere gestibile il database anche via Drupal, con l'obiettivo di sostituire gradualmente l'ormai vetusto gestionale in uso alle Risorse Umane. Si tratterebbe di creare una serie di pagine accessibili da URL del tipo admin/employee/%/edit o admin/department/new ed implementare tramite il generatore di form di Drupal un piccolo gestionale che permetta tutte le operazioni di creazione, modifica e cancellazione di impiegati e dipartimenti. A questo punto, però, l'idea di importare i dati in Drupal e gestirli come nodi veri e propri diventa più sensata, quindi anche quella potrebbe essere una soluzione papabile... Buon divertimento con l'analisi di questo problema!