Specificații ale unui sistem de management al conținutului utilizând MVC

Bogdan Dumitru CLIPA

Abstract—**The content in web pages tends to grow very fast, and we need a good, extendible way to manipulate the data and provide reports, all with a nice, fast user experience. A content management system built upon a Model View Controller design pattern offers the possibility of creating and integrating in the application new modules in a easy way. The main goal of this design pattern is to isolate as much as possible the business and logic part of the application from the user interface. This makes it very easy to change and improve the user experience without affecting the logic of the application. The paper presents the specifications of a web content management system implemented with MVC design pattern in PHP.**

Index Terms—**design patterns, MVC, CMS, web applications**

I. INTRODUCERE

CANTITATEA de informații stocată şi manipulată online creşte foarte repede, cerințele din partea utilizatorilor sunt din ce în ce mai numeroase, iar aplicațiile ce manipulează datele trebuie să satisfacă aceste cerințe.

Aplicațiile cel mai des utilizate pentru diverse categorii de pagini de internet (pagini de portofoliu, ziare online, magazine) sunt sistemele de management al conținutului (Content Management System CMS). Un sistem de management al conținutului trebuie să fie rapid, uşor extensibil prin plugin-uri sau noi module, iar interfața cu utilizatorul trebuie să poată fi uşor schimbată pentru a deservi scopului paginii. Acesta trebuie să simplifice publicarea de conținut pe internet pentru utilizatorii ce au cunoştiințe minime de programare. Sistemul trebuie să permită mai multor utilizatori să coopereze în adăugarea şi modificarea datelor de pe pagina de internet. Ei vor avea roluri şi drepturi diferite asupra datelor din baza de date cu care lucrează sistemul.

În momentul de față, există sisteme de management al conținutului comerciale şi necomerciale, fiecare îndeplinind într-o măsură mai mare sau mai mică cerințele unui CMS de calitate.

Sistemul de management al conținutului propus, are ca obiective:

- realizarea proceselor uzuale cu un minim de cod;
- utilizarea aceleiaşi baze pentru mai multe tipuri de pagini de internet *(portofoliu, blog, ştiri, magazin);*
- extindere uşoară prin adăugarea de noi module;
- crearea uşoară, rapidă de noi interfețe grafice pentru paginile de internet cu un minim de cunoştiințe de programare.

Deoarece majoritatea serviciilor de găzduire actuale oferă suport pentru PHP ca limbaj de scripting, şi MySQL ca SGBD, aplicația va fi construită, având la bază cele două tehnologii.

Sistemul propus în această lucrare, îşi doreşte să rezolve problemele de bază ale unui CMS printr-o implementare uşoară şi eficientă a şablonului de programare Model View Controller (MVC) [4]. Spre deosebire de Java Servlets sau Java Server Pages, limbajul PHP [1] nu oferă nativ o implementare a şablonului MVC, această sarcină fiind lăsată utilizatorului.

În continuare se va prezenta o posibilă implementare a acestui şablon, oferind aplicațiilor scrise flxibilitate ridicată, şi posibilitatea de a fi uşor extinse.

II. CONSIDERENTE TEORETICE

Model View Controller [4] este un şablon de progamare, folosit în proiectarea aplicațiilor. Acest şablon separă partea de logică a aplicațiilor de interfața cu utilizatorul, oferind autonomie modulelor ce alcătuiesc aplicația.

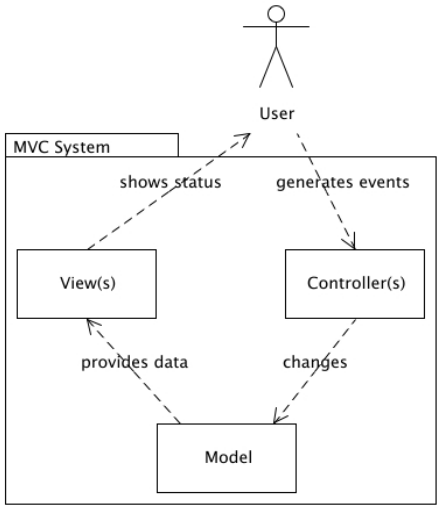

Fig. 1. Diagrama de bază a cazurilor de utilizare pentru şablonul de programare MVC.

MVC implică următoarele:

- utilizatorul interacționează cu interfața utilizator;
- controllerul preia evenimentele generate de utilizator

Bogdan Clipa este student în anul II la programul de masterat Ştiința şi ingineria calculatoarelor, Facultatea de Inginerie Electrică şi Ştiința Calculatoarelor, Universitatea "Ştefan cel Mare" Suceava.

şi invocă modelul necesar pentru a prelucra datele cerute; după aceste prelucrări iniţializează răspunsul către utilizator;
- modelul prelucrează datele şi oferă utilizatorului rezultatul dorit.

Scopul şablonului este de a decupla modelele de interfaţa utilizator, pentru a simplifica menţinerea şi modificarea aplicaţiilor.

III. SPECIFICAŢII DE IMPLEMENTARE

Şablonul MVC oferă ideea dar nu oferă un model de implementare.

Modelul de implementare propus, foloseşte 4 clase: Model, Controller, Registry, Router [3]. Acestea vor fi clasele de bază care constituie scheletul aplicaţiei, şi pe care se va construi în continuare.

În această configuraţie, aplicaţia poate fi considerată una modulară, fiecare pereche de fişiere Model, Controller, View, care extind clasele de bază, fiind independentă de restul aplicaţiei şi putând evolua în mod autonom.

Clasa **Model** este o clasă abstractă folosită drept clasă de bază pentru toate clasele model ale modulelor aplicaţiei. Noile module care vor avea nevoie de clase de tip model, vor extinde clasa de bază. Această clasă nu conţine decât un obiect de tip registry pentru a avea acces la diferite obiecte si variabile globale ale aplicaţiei. Restul metodelor ce vor fi implementate in fiecare clasă copil ţin de necesităţile modulului din care clasa model face parte.

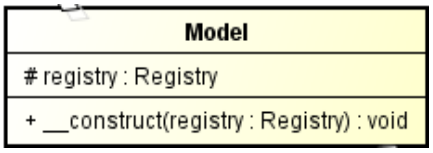

Fig. 2. Reprezentarea clasei de bază Model în diagrama de clase.

```
abstract class Model {
  protected $registry;

  public function __construct($registry) {
    $this->registry = $registry;
  }
}
```

Clasa **Controller** ca şi clasa Model este o clasă abstractă. Prin clasele de tip controller se face interfaţarea dintre fişierele de tip *view,* fişiere ce reprezintă interfaţa cu utilizatorii şi fişierele de tip model, unde se prelucrează cererile acestora.

Clasa de bază Controller oferă o metodă de a încărca *modelele* pentru prelucrările necesare *loadModel(model).* Aceasta întoarce un obiect de tipul clasei model dorite. Pentru a afişa utilizatorului rezultatele dorite, se apelează metoda *renderTpl(template).* Metoda returnează un string ce conţine codul html al paginii ce trebuie afişate utilizatorului.

Vectorul *TplData* este o proprietate a clasei Controller, ce conţine datele prelucrate ce vor popula fişierul *view*. Metoda renderTpl citeşte fişierul de tip view şi adaugă în locurile predefinite datele din vectorul menţionat. Ca şi clasa Model, pentru a avea acces la obiecte şi proprietăţi globale, clasa de bază Controller are ca dată membră un obiect de tip Registry.

Pentru a avea un punct comun în care începe prelucrarea datelor, clasa defineşte metoda index. Această metodă trebuie implementată în fiecare clasă copil care extinde clasa de bază. Dacă nici o altă metodă nu e specificată explicit, aceasta va fi cea apelată la invocarea controllerului.

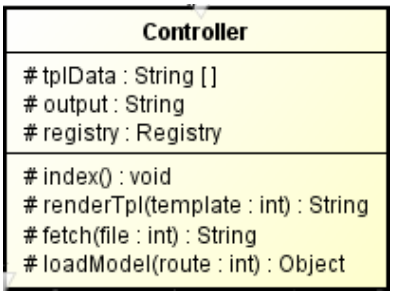

Fig. 3. Reprezentarea clasei de bază Controller în diagrama de clase.

Registry este o clasă care nu este tipică pentru şablonul MVC. Aici, aceasta este folosită pentru a memora diferite obiecte şi proprietăţi globale. Clasele de bază Model, Controller şi Router, au ca dată membră un obiect de tip Registry pentru a avea acces la aceste proprietăţi. Clasa are o singură proprietate, un vector de obiecte în care vor fi ţinute datele necesare. Aceaste date vor fi accesate prin intermediul metodelor de tip getter şi setter.

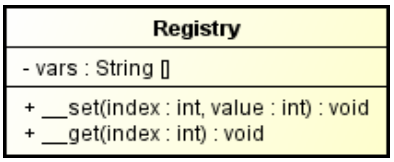

Fig. 4. Reprezentarea clasei Registry în diagrama de clase.

Clasa **Router** oferă punctul de intrare în aplicaţie. Acestă clasă va şti fişierul controller, clasa controller, metoda ce va fi apelată şi parametrii acesteia dacă sunt necesari.

Constructorul clasei va primi o cale de forma *„numeDirector/Controller/metodă”* . Pentru a găsi fişierele necesare şi clasa dorită, se recurge la o convenţie de nume. Fiecare fişier controller va fi cel precizat în cale, plus exstensia *.php,* iar numele clasei va fi cel al fisierului plus subfixul *Controller.* De exemplu la calea *„catalog/categorii”* se caută fişierul controller *categorii.php* în directorul *catalog*, de unde se iniţializează un obiect al clasei *categoriiController*. Dacă în cale nu este precizată o metodă ce trebuie apelată, se va apela metoda *index* care trebuie definită de toate sub clasele Controller.

De obicei, calea către controllerul apelat este o varibilă definită în query string. Dacă aceasta este inexistentă, se va apela un controller implicit. Metoda *action()* a clasei este cea care apelează controllerul dorit.

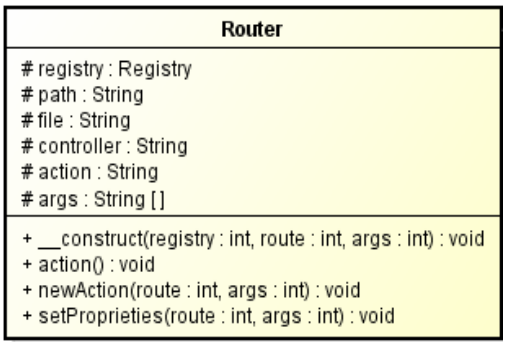

Fig. 5. Reprezentarea clasei Router în diagrama de clase.

Fişiere de tip **view** sunt fişierele ce vor fi afişate utilizatorilor. Reprezintă fişiere PHP care vor da formă conţinutului creat de model. În mare parte sunt alcătuite din cod html. Sigurul cod PHP care apare în ele este cod inline, unde se doreşte afişarea conţinutului dinamic. În aceste fişiere sunt afişate diferite variabile, a căror valoare va fi luată din vectorul *tplData* din controllerul asociat. Astfel, fişierele de tip view pot fi uşor modificate pentru a exprima identitatea fiecărei pagini în parte.

```
<table class="gridView">
          <thead>
           <tr>
             <?php echo $tableView; ?>
           </tr>
         </thead> …
```

Codul HTML ce înconjoară tagurile *<?php ?>* poate fi modificat pentru a schimba modul de afişare a informaţiilor, fără a modifica partea de logică a aplicaţiei.

Punând în practică această implementare a şablonului de programare MVC, sistemul de management al conţinutului devine modular, uşor de întreţinut şi de depanat în caz de eroare, toate erorile fiind uşor de localizat.

Pentru a crea un nou modul este necesar un minim de 2 fişiere, unul de tip *controller*, şi unul de tip *view*. Fişierul *model* nu este absolut necesar, dacă partea de logică este implementată în alt fişier. Fişierele model şi controller ale noilor module trebuie să extindă clasele de bază Model şi Controller.

Diagrama de clase a accestei implementări poate fi reprezentată astfel:

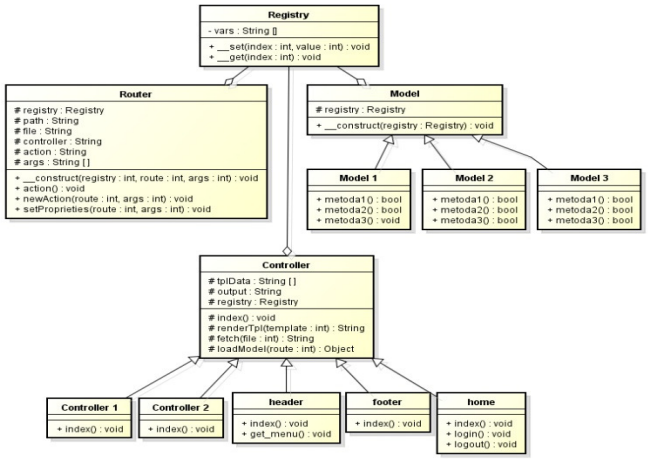

Fig. 6. Reprezentarea diagramei de clase pentru această implementare a şablonului MVC.

Într-o astfel de configuraţie, fişierul *index* este singurul punct de intrare în aplicaţie. Aici se declară proprietăţile globale, se instanţiază obiectele necesare, iar routerul va apela controllerul şi metoda dorită din acesta. De aici tot controlul este deţinut de către modulul cerut de utilizator.

De exemplu, pentru a crea un nou modul ce se va ocupa de managementul categoriilor în cadrul sistemului de management al conţinutului, vom crea 3 noi fişiere *categoriiModel.php, categoriiController.php* şi *categoriiView.php* care vor avea associate clasele *categoriiModel* şi *categoriiController*.

Clasa categoriiModel

```
class categoriiModel extends Model{
  function add($name, $description, $activeState){}

  function delete($id){}

  function update($id, $name, $description,
                  $activeState){ }

  function get($fields, $filter, $order_by){}
}
```

Clasa categoriiController

```
class CategoriiController extends Controller {
  public function index() {
    $this->tplData['title'] = "Categories";

    //handle user requests

    //render header
    //render content
    //render footer
  }
}
```

Fisierul view *CategoriiView.php* va afişa utilizatorilor conţinutul pe care-l dorim, în acest caz doar titlul paginii.

```
<div class='head'>
  <h2><?php echo $title; ?></h2>
</div>
```

IV. Concluzii

Implementarea unui şablon de tip Model View Controller într-un sistem de management al conţinutului rezolvă două mari probleme ale aplicaţiilor web:

- separarea părţii de logică de cea de afişare
- organizarea modulară a aplicaţiei.

Aplicaţia poate fi uşor extinsă, orice nou modul putând fi scris în minim 2 fişiere, unul de tip controller, şi unul view.

Se pot crea uşor noi interfeţe grafice, partea de php din fişierele de tip *view* fiind minimă.

Pe viitor în nucleul sistemului se poate adăuga posibilitatea de *caching* pentru a servi utilizatorilor pagina dorită fără latenţă dată de o nouă prelucrare a datelor.

Referinţe

[1] Harry Fuecks “The PHP Antholgy”
[2] http://www.phppatterns.com/docs/design/archive/model_view_controller_pattern
[3] http://phpro.org/tutorials/Model-View-Controller-MVC.html
[4] http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller