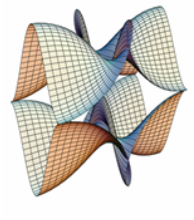


Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM II

Predavanje 07 - PHP - Model-View-Controller

14. travnja 2020.

Sastavio: Zvonimir Bujanović

Getteri i *setteri*

- U OOP je običaj da su varijable članice privatne ili protected.
- Pristup tip varijablama onda osiguravamo preko funkcija:
  - `function get_var() { return $this->var; }`
  - `function set_var($x) { $this->var = $x; }`
- U PHP-u postoje "magične metode" `__get()` i `__set()` koje ovo rade automatski.

- Neki smatraju da korištenje `__get()` i `__set()` osim u specijalnim situacijama nije dobra ideja.
- Pitanje: Što možemo postići ako stavimo da `get_var` vraća referencu? Isto pitanje za iduća 2 slajda: što se postiže ako `__get` vraća referencu?

Getteri i setteri

```php
class Test {
    private $x, $y;

    public function __get( $property ) {
        if( property_exists( $this, $property ) )
            return $this->$property;
    }

    public function __set( $property, $value ) {
        if( property_exists( $this, $property) )
            $this->$property = $value;
        return $this;
    }
};

$t = new Test();
$t->x = 8;  // Poziva $t->__set( 'x', 8 );
echo $t->y; // Poziva $t->__get( 'y' );
```

Getteri i *setteri* - Registry

```php
class Registry
{
    private $vars = array();

    public function __set( $index, $value )
    {
        $this->vars[$index] = $value;
    }

    public function __get( $index )
    {
        return $this->vars[$index];
    }
}

$reg = new Registry();
$reg->nesto = 8;  // Poziva $reg->__set( 'nesto', 8 );
echo $reg->nesto; // Poziva $reg->__get( 'nesto' );
```

`require` i `include`

- Ako se PHP aplikacija sastoji od više skripti, možemo (kao u C-u) "include-ati" jednu skriptu u drugu. Efekt je da se include-ana datoteka copy/paste-a na to mjesto.
- Postoje 4 vrste include-a u PHP-u:
  - `include 'a.php'`
    Ako 'a.php' ne postoji, javi warning i nastavi izvođenje.
  - `require 'a.php'`
    Ako 'a.php' ne postoji, javi grešku i prekine izvođenje.
  - `include_once 'a.php'`
    Kao **`include`**, ali se pazi da se 'a.php' ne include-a više od jednom.
  - `require_once 'a.php'`
    Kao **`require`**, ali se pazi da se 'a.php' ne include-a više od jednom.
- Najčešće koristimo **`require_once`**.

require i include

Oprez: Ako u browseru otvorimo skriptu `A.php`, donji kod neće raditi jer `C.php` nije u istom folderu kao `A.php`:

```php
// A.php
require_once 'folder/B.php'

// folder/B.php
require_once 'C.php'

// folder/C.php
...neki kod...
```

Rješenje: `__DIR__` daje folder skripte *čiji kod editiramo*:

```php
// A.php
require_once __DIR__ . '/folder/B.php'

// folder/B.php
require_once __DIR__ . '/C.php'
```

Autoload

- Ako nam je naporno include-ati sve datoteke u kojima su definirane klase, PHP to može automatski napraviti u trenutku kada stvaramo novi objekt nekog tipa.
- Potrebno je samo pozvati `spl_autoload_register`, te kao parametar proslijediti (anonimnu) funkciju koja objašnjava koju datoteku treba include-ati za određenu klasu.

```php
spl_autoload_register( function ( $class_name )
{
    // ako se klasa zove Nesto, include-a se nesto.class.php
    $fileName = strtolower($class_name) . '.class.php';
    if( file_exists($fileName) === false )
        return false;

    require_once ($file);
    return true;
} );

$x = new Test(); // automatski: require_once 'test.class.php'
```

"Čudno" pozivanje metoda, stvaranje objekata

- Donji kod je ispravan u PHP-u!

```php
$var = 0; $imeVarijable = 'var';
$$imeVarijable = 7; // Sad je $var = 7.

$imeKlase = 'Test';
$t = new $imeKlase; // isto kao $t = new Test;

// Pretp. da klasa Test ima člansku funkciju bar( $a, $b )
$metoda = 'bar';
$t->$metoda( 3, 4 ); // isto kao $t->bar( 3, 4 );

// Pretp. da klasa Test ima statičku člansku funkciju foo( $a )
$statickaMetoda = 'foo';
$imeKlase::$statickaMetoda( 7 ); // isto kao Test::foo( 7 );
```

Model-View-Controller

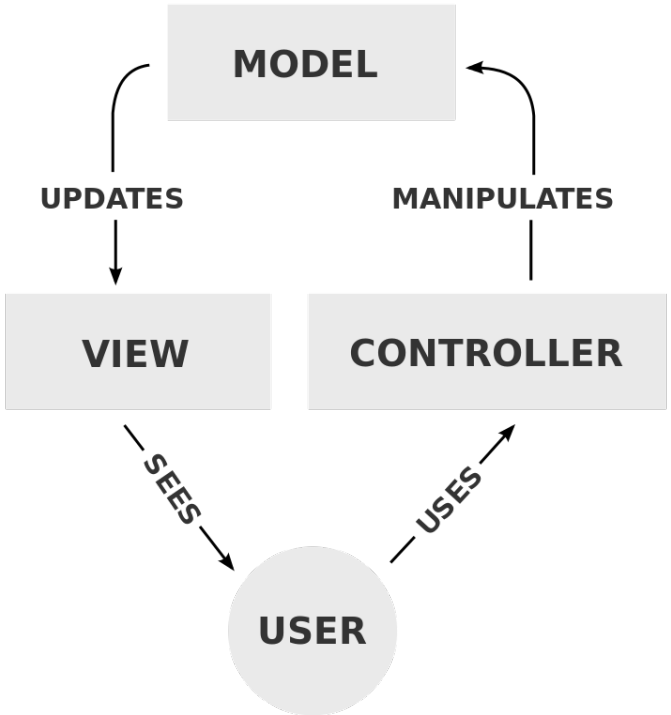
Problemi s dosadašnjim pristupom pisanju PHP aplikacija:

- HTML i PHP su ispremiješani.
- Obrađivanje korisničkog ulaza (`$_GET`, `$_POST`...) je ispremiješano s logikom aplikacije.
- Teško je snaći se u kodu.
- Teško je dodati novu funkcionalnost i održavati kod.
- Vrlo neprikladno za složenije aplikacije i više developera.

Ciljevi:

- *Separation of concerns*.
  - HTML (gotovo) potpuno odvojiti od PHP-a.
  - Obradu korisničkog ulaza potpuno odvojiti od logike.
- Postići da više developera može nezavisno raditi na aplikaciji.
  - Često web-dizajneri (koji rade HTML+CSS) ne znaju dobro PHP i obratno.
  - Omogućiti istovremen razvoj više developera na proširivanju funkcionalnosti aplikacije.

Model-View-Controller



- MVC je "arhitekturalni obrazac (*pattern*)" originalno namijenjen za implementaciju korisničkog sučelja (GUI).
- Prilagođen je i za protok podataka u web-aplikacijama.
- Brojni PHP okviri (Symfony, CakePHP, Laravel, ...) su bazirani na MVC, kao i web-okviri u drugim jezicima (npr. Django, Ruby on Rails).

Model-View-Controller

- Mi ćemo ovdje izvesti od nule jednu moguću varijantu MVC frameworka.
- Za svaku iduću aplikaciju sve ove korake neće trebati raditi od početka.
  $\rightsquigarrow$ jednostavno možemo uzeti gotovo konačno rješenje Zadatka 5 i samo prilagoditi sadržaje direktorija model/view/controller novoj aplikaciji.

Model

- Predstavlja logiku (web-)aplikacije i pripadne strukture podataka.
- Ovaj sloj je potpuno odvojiv i smislen i bez web-a.
- Isti model obično koristi više raznih pod-stranica iste web-aplikacije.
- Model obično implementira 3 aspekta:
  - Strukture podataka koje predstavljaju objekte s kojima se manipulira u aplikaciji (npr. `User`, `NewsArticle`, ...).
  - Načine za dohvaćanje tih podataka ("data mapper" – kako dohvatiti podatke iz baze, cache-a, diska...).
  - Sučelje (servis) za sve koji te podatke trebaju ("business logic").
- Stoga, model:
  - Nema nikakvu interakciju s korisnikom aplikacije.
  - Ne koristi `$_GET`, `$_POST` i slične.
  - Ne sadrži HTML niti ikakav drugi prezentacijski kod.

Zadatak 1

Rješavat ćemo postupno Zadatak 4 iz Zadataka za vježbu.

- Napravite direktorij `library`.
- U tom direktoriju napravite poddirektorije `model`, `controller`, `view`, `app`.
- U poddirektoriju `app/database`:
  - Pripremite datoteku `db.class.php` – singleton klasu za spajanje na bazu. Prilagodite podatke za spajanje.
  - Spremite datoteke `create_tables.php` i `seed_tables.php` (skinite ih s web-stranice), te ih pokrenite iz browsera. Ovo će napraviti tablice u bazi i popuniti ih nekim probnim podacima.
- U poddirektoriju `model` napravite datoteke:
  - `user.class.php` – klasa `User` koja ima članove `id`, `name`, `surname`, `password`, te konstruktor i pripadne gettere i settere.
  - `libraryservice.class.php` – klasa `LibraryService` koja ima samo funkciju `getAllUsers()` koja vraća polje svih korisnika iz baze podataka.

View

- Predstavlja sloj za vizualizaciju modela u nekom pogodnom formatu. U web-aplikacijama taj format je najčešće HTML, ali može biti i neki drugi (npr. JSON – kasnije).
- Tipično će svaka podstranica u web-aplikaciji biti predstavljena jednim view-om.
- Moguće je više puta koristiti iste view-ove za neke tipične radnje: ispis headera, menija, footera, formi koje se pojavljuju više puta i slično.
- Stoga, view:
  - Treba sadržavati skoro isključivo HTML.
  - Koristi PHP na vrlo trivijalnoj razini – samo za ispis varijabli, ili iteraciju po PHP polju za ispis varijabli.
  - Ne pristupa bazi podataka, datotekama na disku i slično.
  - Ako uopće pristupa modelu (što se može u potpunosti izbjeći), onda koristi isključivo read-only pristup – ne izmjenjuje model.
  - Služi samo za dohvaćanje i prikazivanje gotovih podataka. Ne radi nikakvu obradu.
  - Ne koristi `$_GET`, `$_POST` i slične.

Zadatak 2

U poddirektoriju `view` napravite sljedeće datoteke:

- `_header.php`
  - Sadrži html deklaraciju, do uključivo `<body>`.
  - Ispisuje naslov web-stranice, pretpostavite da je dostupan u varijabli `$title`.
- `_footer.php`
  - Sadrži samo `</body></html>`.
- `users_index.php`
  - Include-a `_header.php`.
  - Ispisuje u tablici sva imena i prezimena svih korisnika.
  - Pretpostavite da podaci o korisnicima dani u polju `$userList` objekata tipa `User`.
  - Include-a `_footer.php`.

Controller

- Obrađuje podatke poslane od strane korisnika, na temelju njih update-a model.
- Radi upit na model, na temelju odgovora priprema podatke koje će prikazati view.
- Dakle, controller je "ljepilo" koje povezuje model (aplikacijsku logiku) i view (prezentaciju podatka).

- Stoga, controller:
  - Pristupa `$_GET`, `$_POST` i ostalim PHP varijablama koje reprezentiraju podatke koje je poslao korisnik.
  - Stvara i koristi servise iz modela – pomoću njih može i dohvaćati podatke iz modela i mijenjati ih.
  - Ne radi direktno upite na bazu, nego koristi servise/data mappere iz modela.
  - Ne sadrži HTML niti ikakav drugi prezentacijski kod.
  - Na temelju podataka dobivenih od korisnika, odabire prikladni view i prikazuje ga.
- Vrlo često postoji 1-1 korespondencija između controllera i viewa.

Zadatak 3

U poddirektoriju `controller` napravite datoteku `usersController.php` u kojoj:

- Definirajte klasu `UsersController`.
- Klasa zasad ima jednu javno dostupnu funkciju `index` koja:
  - Stvara novi `LibraryService`.
  - Popunjava varijable `$title` (sami odaberite što) i `$userList` (popis svih korisnika iz baze) za view.
  - Include-a view `users_index.php`.

Router

- Sada je potrebno nekako dati korisniku pristup aplikaciji.
- Bez obzira koliko modela, controllera i view-ova imali, korisnik će pristupati samo skripti `index.php`.
- Ovisno o željenoj "ruti" poslanoj kroz parametar `$_GET['rt']`, korisnika ćemo preusmjeriti odgovarajućem kontroleru.
- Ako korisnik pristupi adresi `library/index.php?rt=con/action`, onda ćemo pozvati funkciju `action` iz kontrolera `conController.php`.
- Ako korisnik pristupi adresi `library/index.php?rt=con`, onda ćemo pozvati funkciju `index` iz kontrolera `conController.php`.
- Dakle, ako se pristupa `library/index.php?rt=users/index`, ili `library/index.php?rt=users`, pozivamo funkciju `index` iz kontrolera `usersController`.

Zadatak 4

Implementirajte `index.php` tako da radi preusmjeravanje kao na prethodnoj stranici.

Provjerite radi li ispravno prikaz svih korisnika u web-browseru.

Zadatak 5

- Dodajte controller `BooksController` i pripadne akcije:
  - `index` – za prikaz svih knjiga u knjižnici.
  - `search` – za pretraživanje knjiga po imenu autora.
  - `searchResults` – za prikaz rezultata pretrage.

  Prilagodite, tj. proširite model.
- Dodajte controller `_404Controller` i pripadni view za prikaz nepostojećih stranica (tj. parova (controller, action)).
- U `view/_header.php` dodajte prikaz menija sa ponuđenim opcijama ispisa svih korisnika, knjiga, te pretraživanja knjiga.

Route i .htaccess

- Linkovi tipa

  `library/index.php?rt=con/action`

  nisu "lijepi".
- Koristeći skrivenu datoteku `.htaccess` u glavnom direktoriju (gdje je `index.php`) možemo napraviti da se pristup "lijepoj" adresi

  `library/con/action`

  automatski preusmjeri na ne-lijepi link.
- Tako korisnik nikad neće vidjeti ne-lijepe linkove.
- Na Apache web-serveru mora biti omogućen `mod-rewrite`: `sudo a2enmod rewrite`.
- Vidi i ovaj link.

```
# sadrzaj datoteke .htaccess
RewriteEngine On
RewriteBase /~zvonimir/library/

RewriteRule "^css/(.*)$" "css/$1" [L]
RewriteRule "^app/boot/(.*)" "app/boot/$1" [L]
RewriteRule "^(.*)$" "index.php?rt=$1" [L,QSA]
```

Konačna implementacija MVC frameworka

- Dodavanje novih dijelova u naš framework nije komplicirano – samo se dodaju novi dijelovi modela, te novi controlleri i viewovi. Ostatak aplikacije ne treba više modificirati.

- Neki problemi:
  - Nespretno je i naporno raditi include-ove.
  - Controlleri nisu povezani, svaki je svoja klasa, ne možemo npr. raditi nasljeđivanje.
  - Prikaz view-a je nespretno izveden, direktnim include-anjem u controller.
  - Dijeljenje varijabli između controllera i view-a je izvedeno nespretno.

- Malo ćemo prilagoditi naš framework tako da slijedimo koncept opisan na stranici PHPRO, autora Kevina Watersona.
  - `__autoload` svih klasa iz modela.
  - Bazna klasa za controllere.
  - Izvojene klase za router i template (apstrakcija view-a).
  - Registry objekt kojeg dijele controlleri i viewovi.

Zadatak 6

- Proučite implementaciju Zadatka 5 u MVC frameworku.
- U controlleru **`Users`**:
  - Modificirajte akciju **`index`** tako da kraj svakog korisnika u tablici možemo kliknuti na gumb "Prikaži posuđene knjige".
  - Klik na bilo koji od tih gumba vodi na **`library/users/showLoans`** koji prikazuje u tablici sve knjige koje je kliknuti korisnik posudio, zajedno s datumom isteka posudbe.
- U header dodajte meni s linkovima na **`library/users`** i **`library/books`**.
- Dovršite sve podzadatke Zadatka 4 iz Zadataka za vježbu. Implementirajte novi servis **`AuthenticationService`** koji će tražiti šifru korisnika za pristup knjigama koje je posudio:

```
1 $as = new AuthenticationService();
2 if( $as->validateUser($user, $pass) )
3     $this->registry->template->show( 'users_showLoans' );
4 else
5     $this->registry->template->show( 'users_loginForm' );
```

Object-relationship mapping (ORM)

Tipično, podatke o svakoj pojedinoj klasi iz modela čuvamo u njezinoj pripadnoj tablici u bazi.

- Operacije poput "dohvati sve objekte neke klase" (knjige, korisnike), "dohvati objekt sa zadanim id-om", "dohvati sve objekte čiji zadani atribut ima zadanu vrijednost" želimo provoditi nad svim klasama iz modela.
- Moguće je implementirati automatsku konverziju između objekata neke klase i redaka u pripadnoj tablici iz baze podataka $\rightsquigarrow$ objektno-relacijsko preslikavanje (ORM).
- Standardno se to radi tako da su klase iz modela izvedene iz bazne, apstraktne klase `Model` u kojoj su implementirane sve funkcije za dohvaćanje i spremanje objekata u bazu.
- Dobra implementacija klase `Model` nije trivijalna.
- Na sljedećim slide-ovima pokazujemo jedan mogući način korištenja takve bazne klase, sličan ORM-u Eloquent iz razvojnog okvira Laravel.

Object-relationship mapping (ORM) - Primjer

Funkcije `all` i `where` su iz bazne klase `Model` i ne treba ih ponovno implementirati u klasama `User` i `Book`.

- SQL upiti se nalaze isključivo u klasi `Model`.

```php
class User extends Model {
    // povezujemo klasu User s tablicom 'users' u bazi podataka
    protected static $table = 'users';
}

class Book extends Model {
    // povezujemo klasu Book s tablicom 'books' u bazi podataka
    protected static $table = 'books';
}

// Dohvati sve korisnike iz baze u polje objekata tipa User
$userList = User::all();

// Dohvati sve knjige autora "Asimov, Isaac"
$bookList = Book::where( 'author', 'Asimov, Isaac' );
```

Object-relationship mapping (ORM) - Primjer

Moguće je jednostavno opisati i odnose među klasama. Na primjer:

- `belongsTo($className, $foreignKey)` je funkcija iz bazne klase `Model` koja opisuje 1-1 preslikavanje. Objekt klase koji ju pozove (`Loan`) ima strani ključ (`id_user`) koji pripada jednom objektu proslijeđenog tipa (`User`), te ona vraća taj objekt.

```php
class Loan extends Model {
    protected static $table = 'loans';

    public function user() {
        return $this->belongsTo( 'User', 'id_user' );
    }
}

// Dohvati posudbu s id-om 1. (id je ključ u svakoj tablici)
$loan = Loan::find( 1 );

// Ispiši ime korisnika koji je napravio tu posudbu.
echo $loan->user()->name;
```