# Lesdoelen

* Bouwen van een controler samen met views.
* Ombouwen van HTML docs naar templates voor xTemplate
* Toepassing templates
* Koppeling Model en views in controler

# OOP en MVC vervolg Views en controler

In deze les gaan we een aantal dingen aanelkaar knopen. De controler samen met de view (van het huiswerk) en de model.

### Vorige les: blog\_model class

Belangrijkste methodes de die we de vorige keer hebben behandeld en voor deze les van belang zijn:

select\_all ();

select\_post ();

en alle getters voor verschillende properties. Deze zullen we nodig hebben voor de controler.

## Views en xTemplate

We hebben 3 paginas om 4 verschillende pagina’s weer te geven.

1: Overzichtspagina

2: Gedetailleerde pagina

3: Edit formulier pagina (en dezelfde maar dan leeg) 4: Nieuwe formulier pagina

Dit zijn de verschillende views die we moeten ombouwen naar templates. Dit is niet heel ingewikkeld en je hebt niet alle methodes nodig die de template class in zich heeft.

Xtemplate is een class die files (tekst gebaseerd) inleest en verwerkt tot een output die we nodige hebben. In ons geval is dat XHTML maar het kan ook .XML zijn .CVS of een ander textbased format. Xtemplate gaat eigenlijk van twee dingen uit.

1. Het **vullen van variabelen** (vanuit de controler php).
2. Het verwerken van **blokken** (of te wel parsen).

Ad1: Aangezien je niet weet welke inhoud getoond moet worden dit is natuurlijk afhankelijk van de data in de db en van de model class. De voorbeeld inhoud wordt vervangen door variabelen of arrays in de template.

Ad2: Herhaling en de mogelijk tot weglaten van bepaalde delen kan worden gedaan door middel van blokken die ‘parsed’. Je kan blokken in blokken zetten die eerst meerdere keren vult en aan het einde parsed zodat je eenvoudig elementen kan maken waarvan je het aantal niet weet.

## Aanpassen van HTML naar een Template

### Het aan maken van blokken

Het aan maken van blokken kan doormiddel van het plaatsen van een stuk code in de te gebruiken template. Een blok heeft een begin en een einde en alles daar binnen kan worden geparsed of niet.

<!-- BEGIN:blok -->

inhoud die getoond kan worden. Hoera!

<!-- END:blok -->

Zorg er voor dat een blok altijd geëindigd wordt. De naam die later wordt geparsed is “blok”.

### Variabelen en verzamelde variabelen plaatsen de template

Een variabele wordt aangeduid door accolades. Tussen de accolades staat de naam van de variabele die wilt plaatsen in de template.

{voornaam}

Je kan ook meerder variable verzamelen in een template. Op deze manier kan je dus variabele namen meerdere malen gebruiken maar ook in een keer een sloot variabele in de template verwerken. De verzamelings naam en de variabele naam wordt gekoppeld door een “.”. Ik zelf schrijf de verzamelings naam meestal in capitalen. Je krijgt dan iets als volgt:

{NAW.voornaam}

{NAW.achternaam}

## PHP code voor het verwerken van de Template

### Aanmaken

Voordat we de template gaan verwerken moeten we eerste de template class koppelen en aangeven welk document we gaan gebruiken voor de output. Dit gaat als volgt:

require\_once(dirname(realpath(\_\_FILE\_\_)). “/xTemplate.class.php”);

//nu gaan we een template verwerking beginnen

//door een object aan te maken.

$template = new xTemplate(dirname(realpath(\_\_FILE\_\_)).“/views/overview.tpl”);

### Variabelen en arrays verwerken in de template

Door binnen het object de methode “assign” aan te roepen kunnen we variabelen en arrays direct in de template zetten (overal waar deze voorkomen – zonder uitzondering). Assign heeft twee argumenten:

1. Variabele naam zoals bekend in de template
2. Variabele die er in wilt zetten.

Dit ziet er als volgt uit:

$template->assign(‘voornaam’, “michiel”);

of

$template->assign(‘voornaam’, $voornaam);

en bij een array met de associatieve key als volgt:

$NAW = array (

‘voornaam’ => ‘kees’,

‘achternaam’ => ‘jansen’,

);

$template->assign(‘NAW, $NAW);

### Het parsen van blokken

Nadat je de variabele in de template heb gezet moet je parsen om ze te kunnen zien. Dit moet dus ook tussentijds als je een deel blok parsed. Parsen gaat met de parse methode. Deze heeft 1 argument te weten de naam van het te parsen blok.

$template->parse(‘blok’);

Als je meerde blokken in elkaar hebt staan moet je deze aanwijzen door middel van een pad van blokken. Dit zijn de blok namen met daar tussen punten. Stel heb een template als volgt:

<!-- BEGIN:adres -->

<ul>

<!-- BEGIN:naam -->

<li>{NAW.voornaam} {NAW.achternaam}</li>

<!-- END:naam -->

</ul>

<!-- END:adres -->

Het kan voor komen dat je achter elkaar meerdere keren parsen. Als je het blok binnen in wil tonen. Doe je het volgende:

$template->parse(‘adres.naam’);

$template->parse(‘adres’);

Dit zal de volgende uiteindelijk output (bijna) geven:

<ul>

<li>kees jansen<li>

</ul>

Let op! Alles wat buiten de blokken staat wordt niet weer gegeven. Er bestaat altijd een soort “root” blok. Hier moet je dus ook alle HTML gerelateerde elementen in kwijt.

### Genereren van output

Als je klaar bent met parsen en vullen moeten out put generen. Dit kan op twee manieren. Je kan text uitvoeren (voorals je het verder wilt verwerken) en gewoon output kan dat ook :

$tempvar = $template->text (‘adres’);

// deze is ook te gebruiken voor niet html

$template->output (‘adres’);

// geeft html terug

## Controler: putting it al together

Na het opzetten van blog\_model kunnen we nu een controler gaan maken. De controler heeft een aantal properties stellen. We beginnen de class met de properties en natuurlijk de construct. In de laatste koppelen we direct de blog\_model. Aangezien de xTemplate wordt gebruikt per verschillende view en de daaraan gekoppelde templates

require\_once(dirname(realpath(\_\_FILE\_\_)). “/xTemplate.class.php”);

require\_once(dirname(realpath(\_\_FILE\_\_)). “/blog\_model.class.php”);

class blog\_controler {

private $BLOG\_MODEL;

public function \_\_construct (){

$this->BLOG\_MODEL = new blog\_model;

}

…

}

Als we dus de blog\_controler openen wordt ook de blog\_model geopend. Dit betekend dat de logica die in de model zit ook uitgevoerd wordt. Dit betekent delete update etc. Van belang is wel dat er dus sommige variabelen wordt gezet voor de verwerking van de model.

Nu kunnen we een aantal methodes maken voor het verwerken van de views. We kunnen een eenvoudig overzicht maken van de gebruikte methodes van blog\_model en de templates.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Template** | **Methodes van blog\_model** | **Action** |
| **1: Over view** | Overview.tpl | Select\_all | default |
| **2: Detail view** | Detail.tpl | Select\_blog | show |
| **3: Edit view** | Form.tpl | Select\_blog | edit |
| **4: New view** | Form.tpl |  | new |

We gaan 4 methodes maken:

private function do\_Overview (){

$blogs = $this->BLOG\_MODEL->select\_all();

// returns a array with array of all datafields

// we need to loop them in the used template

$template = new xTemplate (dirname(realpath(\_\_FILE\_\_)).”/overview.tpl”);

$blog[‘message’] = $this->BLOG\_MODEL->get\_message();

foreach ($blogs as $blog){

$template->assign(‘BLOG, $blog);

$template->parse (‘html.blogs.blog);

}

$template->parse (‘html.blogs);

$template->parse (‘html’);

$template->output (‘html’);

}

private function do\_Detail (){

$blog = $this->BLOG\_MODEL->select\_blog();

$template = new xTemplate (dirname(realpath(\_\_FILE\_\_)).”/detail.tpl”);

$template->assign(‘BLOG, $blog);

$template->parse (‘html.blogs);

$template->parse (‘html’);

$template->output (‘html’);

}

private function do\_Edit (){

$blog = $this->BLOG\_MODEL->select\_blog();

$blog[‘action’] = “update”;

$blog[‘message’] = $this->BLOG\_MODEL->get\_message();

$template = new xTemplate (dirname(realpath(\_\_FILE\_\_)).”/edit.tpl”);

$template->assign(‘BLOG, $blog);

$template->parse (‘html.blogs);

$template->parse (‘html’);

$template->output (‘html’);

}

private function do\_New (){

$blog[‘action’] = “insert”;

$blog[‘message’] = $this->BLOG\_MODEL->get\_message();

$template = new xTemplate (dirname(realpath(\_\_FILE\_\_)).”/new.tpl”);

$template->assign(‘BLOG, $blog);

$template->parse (‘html.blogs);

$template->parse (‘html’);

$template->output (‘html’);

}

Afhankelijk van de actie gedefinieerd in de pagina’s en links binnen de pagina’s. Gaan we in de construct een oproepdoen op een methode die bepaalt wat er getoond gaat worden.

public function do\_Blog (){

//and this time a switch

switch ($\_REQUEST[‘action’]){

case “detail”;

$this->do\_Detail();

break;

case “edit”;

$this->do\_Edit();

break;

case “new”;

$this->do\_New();

break;

default;

$this->do\_Overview ();

break;

}

}

en oproepen in de construct.

public function \_\_construct (){

$this->BLOG\_MODEL = new blog\_model;

$this->do\_Blog();

}

En van buiten de boel oproepen in de index.

require\_once (dirname(realpath(\_\_FILE\_\_)).”/blog\_controler.class.php”);

$blog = new blog\_controler;

En klaar is kees.