# Lesdoelen

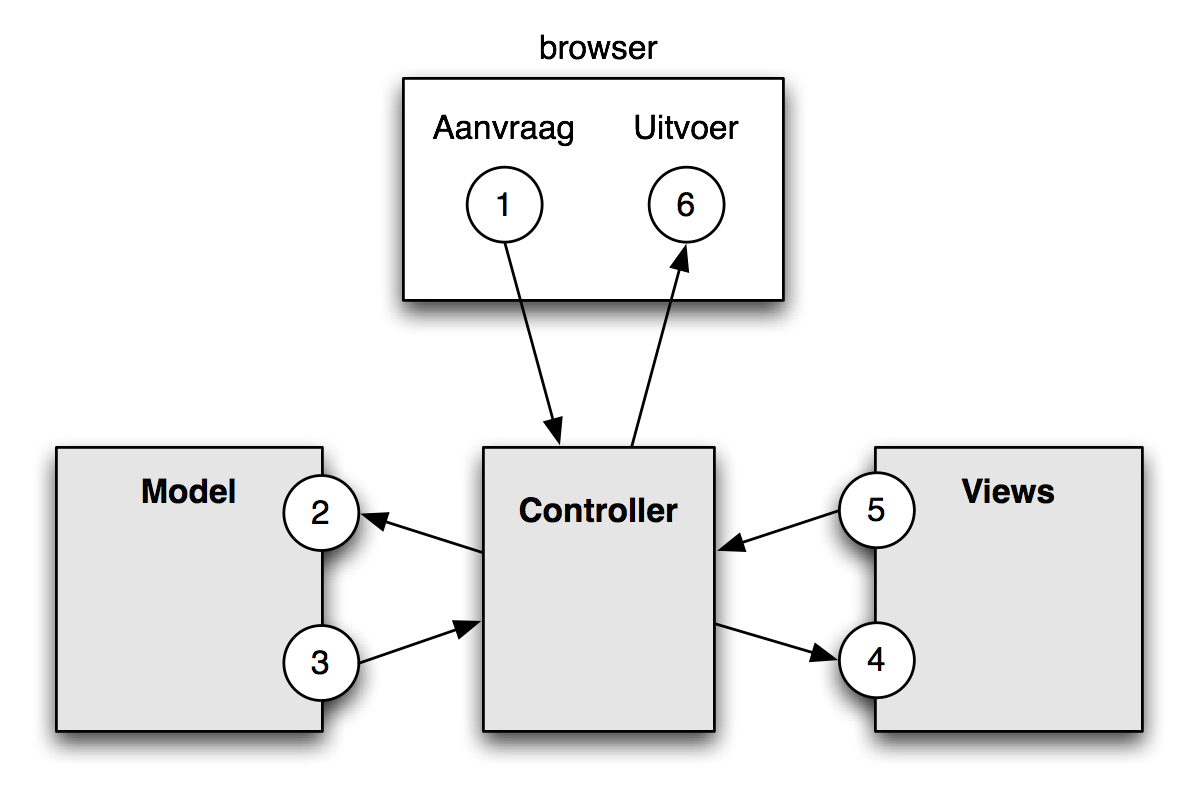
* Idee krijgen van wat MVC is
* Weten wat een framework is.
* Wat de verschillende onderdelen van de model, view controler doen
* Waarom mvc word toegepast en wat de voordelen zijn
* Basis van de model

# OOP en MVC (Frameworks) met php

## Inleiding

MVC staat voor “Model View Controler” en is een architectuur patroon uit de informatica. Dit model is ontstaan op de fameuze XEROX parc (zoals de oude mac) en al in 1978 beschreven. Uitgangspunt is dat de vormgeving en de business logica worden gescheiden, zodat dat ze eenvoudiger los van elkaar te wijzigen zijn. De MODEL representeert de data (dus ook de connectie naar de database) en de manipulatie van deze data; de VIEW representeert de user interface kenmerken en alles wat door de gebruiker wordt getoond en bedient kan worden; De CONTROLER handelt de communicatie af tussen MODEL en VIEW.

In een browser versie kun je het volgende overzicht maken.



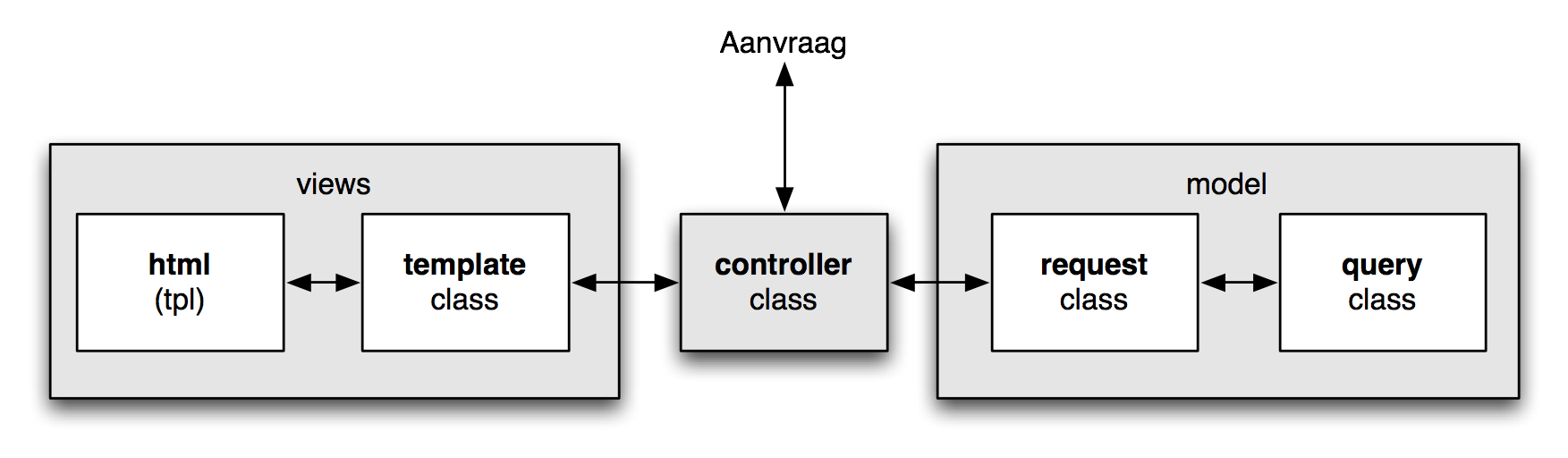
Een Framework is een omgeving waar al veel onderdelen die nodig zijn voor het schrijven van script of programmas aanwezig zijn. Er zijn veel frameworks die MVC gebruiken als manier van ontwikkelen in veel verschillende talen. De frameworks bestaan in de talen: php, ruby, java, .net(asp). Etc.

Aangezien veel mensen eerst de oude manier van procedureel programmeren leren en daarna gaan kijken is de leer curve bij dit soort frameworks nogal steil. Aangezien je weinig kapot kan maken moet je gewoon trial en error toepassen. En goed kunnen zoeken. Als je aan een framework begint kijkt dan goed naar eventuele documentatie. Je kunt een overzicht vinden van verschillende frameworks in verschillende talen op WIKIpedia: <http://en.wikipedia.org/wiki/Model-view-controller>

# MVC blog

## Te gebruiken MVC - framework

Wij gaan niet met een framework werken maar leggen in twee lessen de basis uit van de MVC doormiddel van een het bouwen van een eenvoudig blog. Het model wat we gaan gebruiken (objecten in MCV-“stijl”) is als volgt:



Wat we zien is dat er een aantal dingen benodigd zijn voor het maken van het software:

De model bestaat bij deze uiteen tweetal dingen een Query-class welk vragen aan de database afhandelt (dus de connectie legt naar de db en de daadwerkelijke queries doet). Deze query-class handelt naast de queries ook wat formatting taken af van de te retourneren data. Dit was het huiswerk van vorige week. (als je deze niet hebt gemaakt als nog doen) Daarnaast heb we een ander onderdeel binnen de model te weten de request-class. Deze handelt de vragen af welke zijn gedefineert in de predefined-vars af maar ook de verschillend het opstellen van de queries afhankelijk zijn van de predifined-vars. (bijvoorbeeld post-1 weggooien)

De controller levert de koppeling tussen de view en de model. De controler in het klein wat de model ook doet afhankelijk wat de gebruiker opvraagt: het ophalen van data (model) en het tonen (view) daarvan.

De **view** bestaat uit twee delen. Aangezien we te maken hebben met html output dienen we daar rekening met te houden. Templating is daarvoor een goed hulpmiddel aangezien een template uit html is opgebouwd en deze wordt verwerkt door een template engine. Er zijn er meerdere van en de keuze is aan jou om te gebruiken welke je wil. Er zijn veel voorstanders van **Smarty**. We gebruiker hier echter XTemplate aangezien deze totaal geen logica bezit alleen maar platte tekst verwerkt. De logica dient in de controller te zitten niet in de template engine. (naar mijn mening).

Xtemplate is te downloaden op:

<http://sourceforge.net/project/showfiles.php?group_id=16714&package_id=14016>

# Voorwerk voor ons blog

## Voor dat we beginnen

Voor dat we aanvangen met het schrijven van de controler en de model is het handig om een inventarisatie te maken van de functionaliteit de we nodig hebben

## Wat willen we als paginas

Wat in basis bij ons blog weer willen geven willen is de volgende paginas:

1. Een overzicht / samenvatting van de verschillende blogentries waarbij direct geedit maar ook weggegooid kan worden
2. Een gedetailleerde pagina van een entry (met edit of delete knop)
3. Een weergave om een entry aan te passen.
4. Een weergave om een entry te maken.

## Wat we nodig hebben in de model?

### Velden

Als je deze pagina’s nader bekijkt zie kun je data technische de volgende velden onderscheiden:

* Ad 1: Een lijst view moet ophalen:  
  **id, categorie, samenvatting**
* Ad 2 en Ad 3: hebben gedetailleerder datanodig voor een bepaald id:

**categorie, tijd, datum, volledige tekst, samenvatting**

* Ad 4 geen data ophalen maar moet de mogelijkheid hebben om de volgende data te bewerken:  
  **categorie, tijd, datum, volledige tekst, samenvatting**

### Acties

Beheren: Nieuwe aanmaken, Aanpassen en Verwijderen

Overzicht genereren.

Zowel het ophalen de data (velden als acties die gerelateerd zijn data worden in de model afgehandeld. De controller gaat er alleen maar voor zorg dragen dat views goed reageren en getoond worden.

We beginnen met bouwen van de model.Het huiswerk is het maken van de views. Volgende les komt de controler aanbod en hoe deze samenwerkt met de views (xtemplate) en de model.

## Blog model

Veel van de acties die worden uitgevoerd door de model zullen uitgevoerd worden met $\_GET (welke pagina) en $\_POST (bewaren etc) variabelen. Als deze allen een andere naam hebben dus $\_GET[‘naam’] bestaat er niet in de $\_POST variant kunnen we net zo goed de $\_REQUEST variant gebruiken. ($\_GET, $\_POST en $\_SESSION).

We bouwen de blog\_model als een class welke de query class gebruikt voor het uitvoeren van de Queries zelf.

class blog\_model {

public function \_\_construct (){

}

}

We hebben de query class nodig en deze zullen we inladen als een property van deze model zodanig dat we deze altijd kunnen gebruiken waneer we hem nodig hebben. Deze zullen we initiëren in de \_\_construct zodat we hem altijd hebben. Daarnaast zijn een aantal properties nodig: data om te verwerken. Id van de post, berichtgeving en een actie, dan kunnen we de model uitbreiden met:

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

class blog\_model {

//properties

public $id;

public $data;

private $QUERY;

public $action;

public $message;

public funccion \_\_construct (){

$this->QUERY = new query;

$this->message();

}

}

Alle door de class verzamelde data wordt verzameld in de $this->data property. We hebben dus een private class nodig die deze zet en ophaalt

private function data ($data){

$this->data = $data;

}

en

public function data (){

return $this->data;

}

Soort gelijke methodes voor voor id, message en action etc….

De basis is gelegd er zijn nu een aantal methodes nodig:

1:Alles selecteren met gelimiteerde velden:

2: Ophalen van detail data (voor de edit en details)

3: Invoegen data

4: Aanpassen van data

5: Verwijderen van data

Punt 1:

private function select\_all (){

//make select statement

$sql = “SELECT id, samenvatting, categorie FROM tbl\_blog”;

//opening the query object and the associative\_select\_method

$data = $this->QUERY->select\_associative ($sql);

//putting the data in the data property

$this->data($data);

}

Punt 2:

private function select\_post (){

$sql = “SELECT samenvatting, tijd, “;

$sql .= “datum, volledige\_tekst, categorie “;

$sql .= “FROM tbl\_blog ”;

$sql .= “WHERE id = ‘”.$this->get\_ID().”’;

$data = $this->QUERY->select\_associative ($sql);

$this->data($data);

}

Punt 3:

private function insert\_post (){

$sql = “INSERT INTO tbl\_blog (“;

$sql .= “id, samenvatting, volledige\_tekst, “;

$sql .= “categorie, tijd, datum “;

$sql .= “) VALUES ( “;

$sql .= “’’, ‘”.$\_REQUEST[‘samenvatting’].”’, “;

$sql .= “‘”.$\_REQUEST[‘volledige\_tekst’].”’, “;

$sql .= “‘”.$\_REQUEST[‘categorie’].”’, “;

$sql .= “NOW(), NOW() )“;

$this->message( $this->QUERY->insert($sql) );

}

Punt 4:

private function update\_post (){

$sql = “UPDATE tbl\_blog SET (“;

$sql .= “samenvatting = ‘”.$\_REQUEST[‘samenvatting’].”’, “;

$sql .= “categorie = ‘”.$\_REQUEST[‘categorie’].”’, “;

$sql .= “volledige\_tekst = ‘”.$\_REQUEST[‘volledige\_tekst’].”’,

$sql .= “) VALUES ( “;

$sql .= “WHERE id = ‘”.$this->get\_ID().”’;

$this->message( $this->QUERY->update($sql) );

}

Punt 5:

private function delete\_post (){

$sql = “DELETE FROM tbl\_blog “;

$sql .= “WHERE id = ‘”.$this->get\_ID().”’;

$this->message( $this->QUERY->delete($sql) );

}

Nu hebben de basis gelegd voor de voor de model we echter missen een onderdeel binnen de class die zelf bepaalt wat er afgehandeld wordt binnen de model. Hiervoor gaan we naar de \_\_contruct waarbij we een functie aan roepen die de afhandeling regelt. Aangezien deze niet de communicatie tussen de view en de model hoeft te lopen, sluiten we hier de controller uit.

public function \_\_construct (){

$this->QUERY = new query;

$this->message();

$this-> process\_action();

}

private function process\_action (){

this->id (@$\_REQUEST[‘id’]);

//call methods depending on action var that are only

//used by this class

if ($\_REQUEST[‘action’] == ‘delete’){

$this->delete\_post();

header(“Location: index.php”);

}else if ($\_REQUEST[‘action’] == ‘update’){

$this->update\_post();

header(“Location: index.php”);

}else if ($\_REQUEST[‘action’] == ‘insert’){

$this->insert\_post();

header(“Location: index.php”);

}

}

# Opdracht

We gaan wat voor werk verrichten voor de volgende les.

## Opdracht 1: xTemplate

Download xTemplate en lees de documentatie door: <http://sourceforge.net/project/showfiles.php?group_id=16714&package_id=14016>

## Opdracht 2: Html maken

Voor ons blog moet er natuurlijk html geproduceerd worden. Voordat we ze kunnen templates kunnen gaan gebruiken moet je eerste voorbeeld html maken. Opdracht is dan ook HTML maken voor de pagina’s: Overzicht met delete en leesmeer links, Detail met edit en delete knop en een edit-formulier. Voor de eenvoud is het, het handigst als je edit-formulier variabelen precies dezelfde namen heeft als de velden in de database. De html met volgens XHTML worden gemaakt (zonder frames) samen met vormgeving in css.

In de les gaan we deze paginas ombouwen tot templates om te gebruiken voor pagina generatie