|  |
| --- |
| Proven Web Concepts |
| Technisch Ontwerp |
| Husqvarna MX GP Website |
|  |
| **Jorn Harkema: s1060920 Ron Jonkers: s1063619 Bas van Koesveld: s1061343** |
| **5-6-2014** |

|  |
| --- |
| Versie: 0.2 gemaakt op: 6-6-2014 |

**Instelling: Windesheim**

**Begeleid door: Wasim Alsaqaf**

Arnout Joosse

# Inhoud

[Inhoud 1](#_Toc389824606)

[1. Introductie 3](#_Toc389824607)

[2. Entity Relational Diagram 4](#_Toc389824608)

[3. Klasse Diagram 5](#_Toc389824609)

[4. Onderzoek naar Cake PHP 7](#_Toc389824610)

[5. Onderzoek naar Facebook/Twitter/Google+ integratie 8](#_Toc389824611)

[5.1. Artikelen delen 8](#_Toc389824612)

[5.2. Posts Laden 9](#_Toc389824613)

[5.2.1. Facebook 9](#_Toc389824614)

[5.2.2. Twitter 10](#_Toc389824615)

[5.3. Foto’s en video’s laden 12](#_Toc389824616)

[5.3.1. Foto’s 12](#_Toc389824617)

[5.3.2. Video’s 13](#_Toc389824618)

[6. Onderzoek naar gebruik van coördinaten en implementatie kaart 14](#_Toc389824619)

[7. Onderzoek naar implementatie iDeal 15](#_Toc389824620)

[8. Implementatie slicing 16](#_Toc389824621)

[9. CakePHP CMS 17](#_Toc389824622)

[10. Cake PHP Routing 20](#_Toc389824623)

[11. Fotobibliotheek 21](#_Toc389824624)

[12. Event Countdownbalk 23](#_Toc389824625)

[13. Sponsors per Event 24](#_Toc389824626)

[14. Nieuwsoverzicht 26](#_Toc389824627)

[15. Nieuws View 27](#_Toc389824628)

[16. Nieuws tags 28](#_Toc389824629)

[17. Nieuws zoekbalk 30](#_Toc389824630)

[18. Racers View 31](#_Toc389824631)

[19. Merchandise overzicht 32](#_Toc389824632)

[20. Merchandise webshop 34](#_Toc389824633)

[21. Nieuws prioriteit 35](#_Toc389824634)

[22. Home pagina 36](#_Toc389824635)

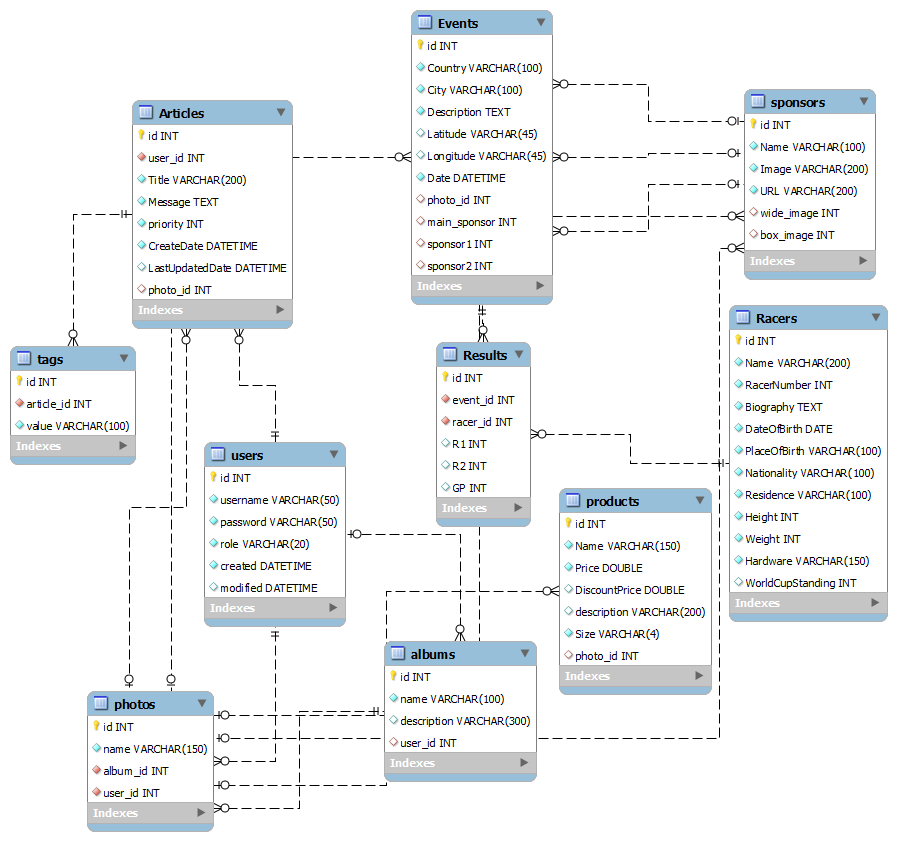
# Introductie

In dit document vertellen wij over de oplossingen die wij hebben gemaakt in het project en hoe wij op deze oplossingen zijn gekomen. Ook waarom wij voor deze oplossingen hebben gekozen. Over de inhoud van het project kan worden gelezen in het Plan van Aanpak en Verslag. Een eis van het bedrijf was dat de website volledig in CakePHP werd gemaakt. Daarom moesten wij eerst CakePHP leren, maar dit is geen probleem geweest. Ook daar kun je meer over lezen in de onderzoeken.

Wij zullen hier bijna exclusief schrijven over de PHP. Wij hebben de javascript, HTML en CSS niet gedaan. Alleen hier en daar was er een probleem met de CSS dat wij hebben opgelost. Dit omdat het probleem te maken had met het dynamisch maken van een statische pagina. Dit betekend dat wij nog best vaak aan responsive design hebben moeten werken. Maar omdat hier de basis al voor stond, was dit niet erg ingewikkeld.

Aangezien wij niet voor de slicing (HTML/CSS/Javascript) hebben gezorgd, zullen we voor sommige pagina’s die dynamisch gemaakt zijn weinig/niks vertellen. Een voorbeeld hiervan is de race resultaten. Hiervoor bestond eerder al een volledige pagina en daar hebben wij simpelweg de database data in gezet.

# Entity Relational Diagram



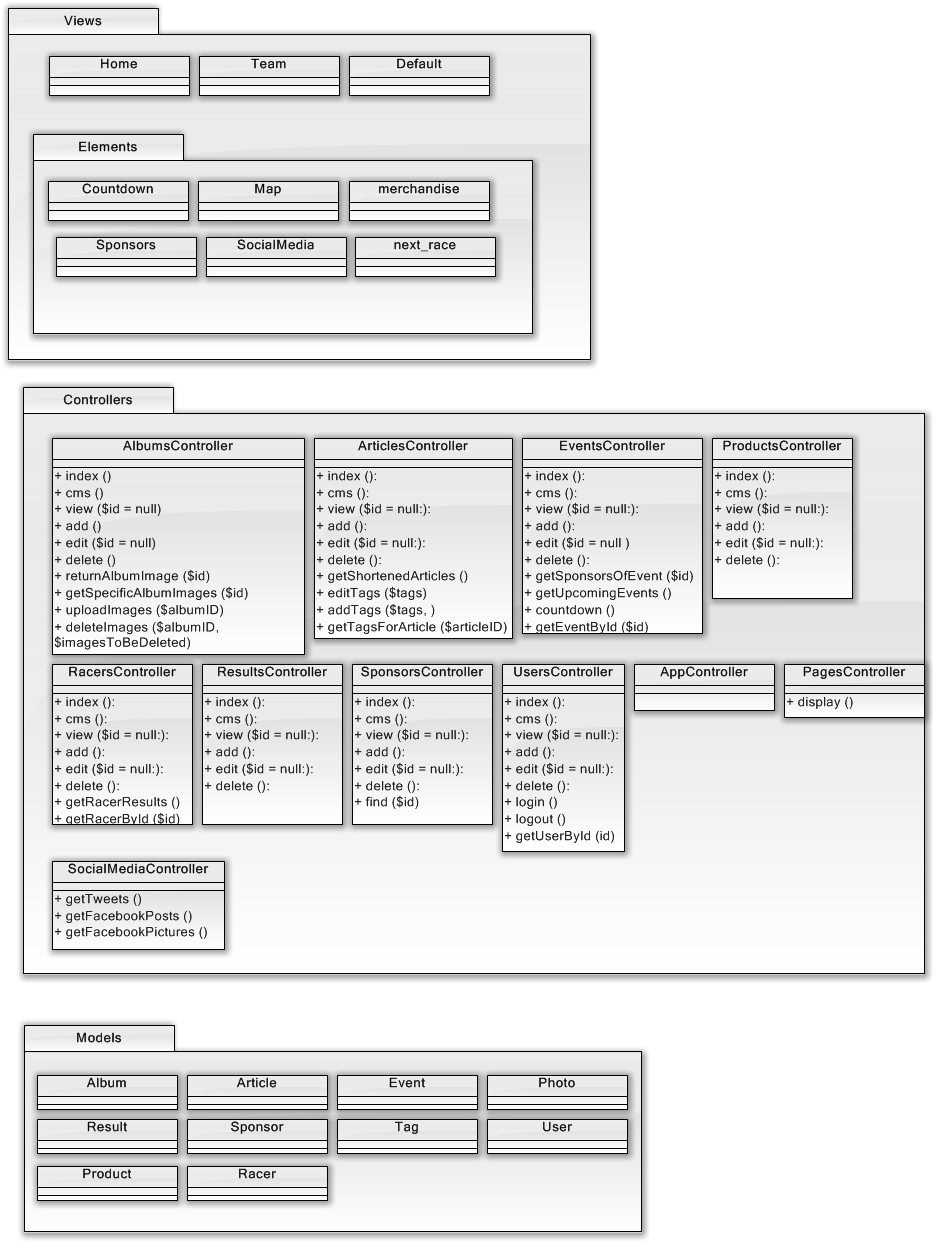
Dit is gelijk ook het klasse model. De enige klassen toegevoegd aan het model van de website zijn voor de CakePHP controllers. Dit zijn de PagesController, SocialMediaController en AppController.

# Klasse Diagram

(Zie volgende pagina)

Zoals in het klasse diagram duidelijk te zien is werken wij met een MVC structuur. Op basis van het ERD zijn er een stel models gemaakt. Elke model heeft een eigen controller. De meeste controller actions hebben views op basis van CRUD (Create, Read, Update, Delete). Deze heten op volgorde Add, Index, Edit en Delete. Index hebben wij gekozen om te gebruiken voor de pagina’s. Zo is articles/index de link naar het artikel overzicht. Dit geldt ook voor een paar andere klassen. Daarom hebben we ervoor gekozen om in het CMS de actie ‘cms’ te gebruiken. Zo leidt articles/cms naar de cms hoofdpagina voor articles.

Er zijn een stel die niet direct zijn verbonden aan een controller. Dit zijn de home en de team view. Dit is omdat deze niet direct verbonden zijn aan één model. Home gebruikt vele meerdere modellen. Team gebruikt helemaal geen modellen en is compleet statisch. Er zijn ook elementen, die in heel veel pagina’s terug komen en daarom als element kunnen worden opgeroepen. Deze gebruiken zelf requests om de data te krijgen die ze nodig hebben. Daarbuiten zijn ze niet direct verbonden aan één controller, daarom heb ik geen lijnen ertussen getrokken in het klassen model.



# Onderzoek naar Cake PHP

Wij hebben in de eerste week onderzoek gedaan naar CakePHP. CakePHP is een MVC framework voor de PHP programmeertaal. Wij hebben dit framework gebruikt omdat het bedrijf Proven Web Concepts dit framework ook gebruikt en ze willen graag alle toekomstige sites in dit framework hebben gebouwd.

Het onderzoek hebben wij aangepakt door verschillende tutorials uit te voeren die op <http://cakephp.org/> beschikbaar waren. Deze tutorials bestonden uit testprojecten die stap voor stap tot een goed werkende applicatie werden gemaakt.

Wij hebben allemaal apart van elkaar deze tutorials uitgevoerd en hierdoor hebben we snel het MVC framework onder de knie gekregen.

# Onderzoek naar Facebook/Twitter/Google+ integratie

## Artikelen delen

Onder elk artikel staan een paar knopjes om het te delen met de social media. Dit maakt het makkelijker voor de gebruiker om de website te delen en geeft de website meer publicatie. Als je op één van de knopjes drukt, wordt op één manier het artikel gedeeld met de gekozen social media. Ik wist alleen in de eerste instantie niet hoe.

Het onderzoek hiernaar ging redelijk gemakkelijk. Na een google zoek opdracht naar ‘Twitter Sharer’ vond ik een website die share links voor je genereerd: <http://www.sharelinkgenerator.com/>. De share integratie van Facebook en Google+ verwacht dat je een URL wilt delen. Dit zou dus de URL zijn voor het artikel. Dit was het moeilijke deel; om de URL te achterhalen van het artikel. Dit was eerst ook niet te testen op de localhost. Ik heb destijds wat gevonden dat online ZOU moeten werken en dat later op de testserver met succes getest. Na een paar kleine aanpassingen werkte dat prima. De gebruikte code was als volgt:

$url = "http://$\_SERVER[HTTP\_HOST]$\_SERVER[REQUEST\_URI]";

Aan deze URL voeg ik de request en action toe (article/view/ID). Door deze URL toe te voegen aan de share URL kon je de pagina makkelijk delen.

https://www.facebook.com/sharer/sharer.php?u=<?php echo $currentUrl ?>

De google URL is ongeveer hetzelfde. <https://plus.google.com/share?url=>.

Voor Twitter was het iets anders. Voor Twitter deel je namelijk een gehele Twitter post, die de gebruiker kan aanpassen voordat hij hem post. In de Twitter post zet ik de URL gevolgd door de titel van het artikel.

https://twitter.com/home?status=<?php echo $currentUrl . " " . $article['Article']['Title']; ?>

De standaard tweet zou dan zijn:

“http://www.jmracing.mx/articles/6 Romain Febvre in the top five in GP of Italy”

Door deze links te koppelen aan de ‘share’ knopjes van elk artikel, was het doel bereikt.

## Posts Laden

### Facebook

Het ophalen van de Facebook posts van een fan pagina is niet moeilijk. De posts op een fan pagina zijn publiek en kunnen zonder toestemming worden gelezen als data. De code hiervoor is daarom zeer voor zichzelf sprekend.

//Load Facebook posts

//The page from which to get the posts

$page\_id = 'husqvarnamxgp';

//The security token for the application

$token = '243799329140027|63915777e84ed91f711ac64cf25ca565';

// get JSON from adres, the @ represses the error it gives upon failing to get content

$page\_posts = @file\_get\_contents('https://graph.facebook.com/'.$page\_id.'/posts?

fields=message&access\_token='.$token);

//Also decode the facebook json.

$page\_posts = json\_decode($page\_posts, true);

Voor de security token van de applicatie is wat uitleg nodig. De applicatie is een applicatie die ik gemaakt heb op Facebook. Deze applicatie is op geen manier verbonden met een Facebook pagina, maar is nodig om requests te kunnen maken naar de Facebook server.

Door een request te maken naar graph.facebook.com kunnen we alle posts ophalen van een gegeven pagina. We kunnen in de URL simpelweg de token zetten, waardoor de request in één regel past. Hierdoor krijgen we JSON terug. (Voor meer informatie over JSON, zie <http://www.json.org/> ) Deze JSON decoden we naar een PHP object. Daarvan verkort ik de posts als ze te lang zijn zodat ze op de pagina passen en stuur ik ze naar de view waar ze worden uitgelezen om de posts te tonen.

### Twitter

Twitter posts ophalen was vergelijkbaar, maar een stuk ingewikkelder. Voor Twitter was naast de app token ook een consumer secret nodig. Dit betekend dat je voor je Twitter app eerst een gebruiker moet registreren. Van deze gebruiker krijg je een consumer key en een consumer secret. Deze moet je allebei samen met de app token en token secret in de header zetten van de request.

Omdat deze data in de header van de request moet, wordt het direct een hoop meer ingewikkeld. Je zult eerst een request moeten maken, en deze de benodigde data toewijzen. Gelukkig bestaan hiervoor meerdere externe libraries die dit verzorgen. Deze moest ik alleen de juiste data geven en die maakte dan een request naar de Twitter server. Daardoor kreeg ik de JSON die ik op dezelfde manier kon decoden.

//Load the external twitter API library  
require\_once(APP . 'Vendor' . DS . "twitter-api-php/TwitterAPIExchange.php");

// These are the twitter app access tokens - see: https://dev.twitter.com/apps/  
$settings = array(  
 'oauth\_access\_token' => "249730612-OWKcfzHN4EgjT5Lm4r2FEFGjkSfvydvAkOaDHCcO",

'oauth\_access\_token\_secret' => "HZTa0hNDUl4LJgCimaUFFPxumfH5xmFeR2eWUOva0Y8TX",

'consumer\_key' => "iMs9fbv99Ev5NqkW20Vio5uSZ",

'consumer\_secret' => "qvkmztmaWukquDEYBMGPPJ6Zf40NCmgiGgh0c1IKBP5vPyk6m7"

);

/\*\* Perform a GET request and echo the response

Note: Set the GET field BEFORE calling buildOauth(); \*\*/  
$url = 'https://api.twitter.com/1.1/statuses/user\_timeline.json';  
$getfield = '?screen\_name=HusqvarnaMXGP';  
$requestMethod = 'GET';  
$twitter = new TwitterAPIExchange($settings);  
$tweets = $twitter->setGetfield($getfield)  
 ->buildOauth($url, $requestMethod)  
 ->performRequest();

//Decode the twitter json and return them  
return json\_decode($tweets, true);

De tweede en derde paragraaf aan code hier zijn gekopieerd van het voorbeeld dat bij de library kwam en ik heb daar mijn benodigde data in gezet. Het grootste probleem was achterhalen hoe ik aan deze data kom. Dit is zonder twijfel het onderdeel waar ik in het gehele project het meeste tijd aan heb besteed. Dit omdat de Twitter documentatie voor het simpelweg ophalen van tweets extreem onduidelijk is.

Na de data te hebben achterhaald kan op een makkelijke manier de data worden geprint. Als eerste exception handling.

if($pageposts == NULL){…

Door de opgehaalde te controleren op ‘NULL’ weet je of de request is gelukt of niet.

//Take the facebook date info and format it to: 01 Januari 2000

$datetime = new DateTime($pageposts['data'][$i]['created\_time']);

echo $datetime->format('d M Y');

Deze code haalt de datum op en zet het in het juiste formaat. Voor Twitter werkt dit precies hetzelfde.

//Facebook

<?php echo $pageposts['data'][$i]['message']; ?>

//Twitter

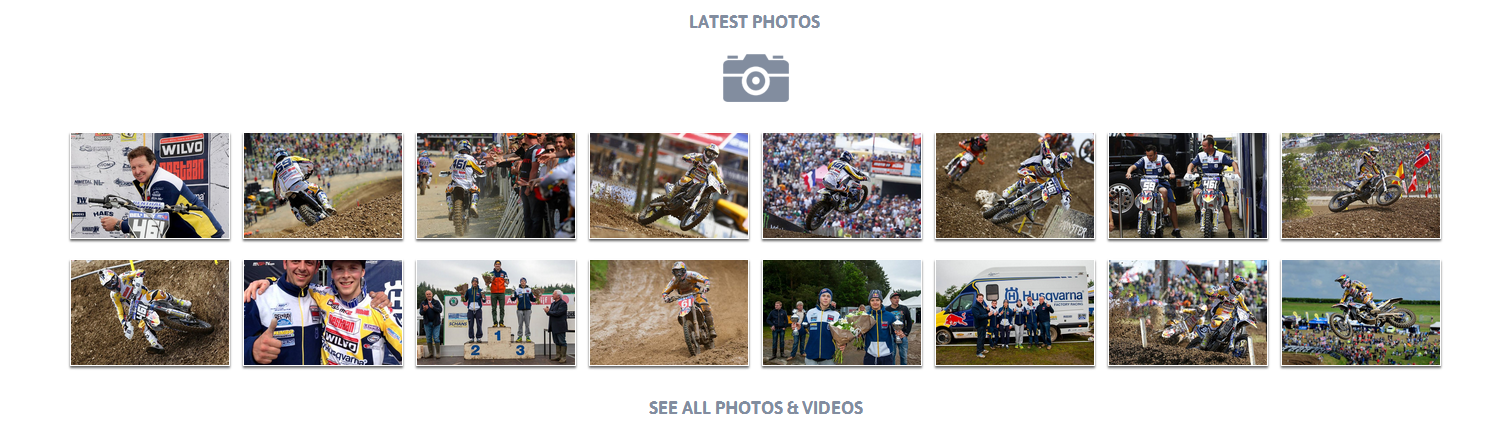
<?php echo $tweets[$i]['text'] ?>

Deze code print het bericht zelf. De $i staat voor de iteratie in de for-loop, aangezien er 2 posts moeten worden opgehaald.

Later waren er veel problemen met de grootte van het schermpje waarin de tekst wordt geprint. Als het éne scherm groter is dan de ander, verloopt de pagina niet meer goed. Dit heeft te maken met het javascript dat wordt gebruikt om de elementen op de pagina te laten verlopen. (Niet van ons en niet door ons geïmplementeerd) Om deze reden hebben we gekozen om de schermpjes redelijk groot te maken, in tegenstelling tot het originele ontwerp. Mocht het ooit gebeuren dat er een tekst komt die niet in het scherm past (wat bijna niet is voor te stellen) dan wordt de tekst simpelweg niet getoond.

## Foto’s en video’s laden

### Foto’s



Het laden van de Facebook foto’s was gelukkig erg makkelijk en lijkt heel erg op de code voor het ophalen van de posts.

//The ID of the facebook picture album

$album\_id = '630229000370596';

//The security token for the application

$token = '243799329140027|63915777e84ed91f711ac64cf25ca565';

// get JSON from adres

$page\_posts = file\_get\_contents('https://graph.facebook.com/'.$album\_id.'/photos?access\_token='.$token);

//Also decode the facebook json.

$page\_posts = json\_decode($page\_posts, true);

Als je de code hebt gezien voor het ophalen van de Facebook posts zal je een paar verschillen en overeenkomsten opmerken. Ten eerste, in plaats van een pagina hebben we nu een album\_id. In dit geval is het benodigde album de foto timeline van Husqvarna MX GP. Door naar het album in kwestie te gaan komt de album\_id in de URL te staan. Deze kun je simpelweg kopiëren en plakken.

Het andere verschil is de request URL. In plaatst van /posts, doen we nu een request naar ?photos. Dit is hoe Facebook zijn API heeft gestructureerd en is verder hetzelfde te gebruiken.

De verkregen JSON is op de volgende manier te lezen:

//Link naar de foto op facebook

<?php echo $photos[$i]['link']; ?>

//Link naar de foto data

<?php echo $photos[$i]['source']; ?>

Met een simpele for loop heb ik deze data in de slicing gezet. Dit verliep tot mijn verassing volledig zonder problemen en was een klusje van hooguit 2 uur, nadat ik de Facebook API al begreep.

### Video’s

Het ophalen van de video’s was helaas niet even makkelijk. De video’s van een gebruiker worden gezien als privé, en daarom moet de gebruiker toestemming geven om deze video’s te bekijken. Helaas kun je deze toestemming niet ‘publiek’ schenken.

De API verwacht dat de gebruiker inlogt op jouw app via de website. Nadat de gebruiker is ingelogd, kan de video data van de ingelogde gebruiker worden gelezen. Dit is helaas totaal niet wat wij willen met deze website. Wij willen gewoon de video data van Husqvarna MX GP ophalen. Dit is alleen (volgens mijn onderzoek) niet ondersteund door Facebook.

Daarom heb ik op het video blokje gewoon een foto gezet (één van de foto’s die eerder is opgehaald) en dit video blokje gelinkt naar het video album op facebook. Dit ziet er prima uit en heeft het akkoord gekregen van de bedrijfsbegeleider.

# Onderzoek naar gebruik van coördinaten en implementatie kaart

Op elke pagina is er onderaan een kaart te zien waarop alle race locaties zichtbaar zijn.

Het was oorspronkelijk de bedoeling dat deze locaties dynamisch via coördinaten in de database zichtbaar werden op de kaart. Maar na de schaal op orde te hebben en alle locaties werden getoond, kwamen we erachter dat dit niet mogelijk was. De kaart heeft een kleine bolling en is niet gelijk aan een officiële kaart, dit zorgt ervoor dat sommige locaties niet goed staan.

Daardoor hebben we nu gekozen om de locaties hard coded erin te zetten.

# Onderzoek naar implementatie iDeal

Het onderzoek naar de implementatie van iDeal was moeizaam. Ik heb eerst heel erg lang naar documentatie en voorbeeld materiaal gekeken voor iDeal. Toen ik eindelijk dacht er een volledig genoeg begrip van te hebben, heb ik de bedrijfsbegeleider gemaild om te vragen naar wat ze daar al hebben. Toen gaven ze aan een bepaalde plugin te gebruiken genaamd Sisow. (<https://www.sisow.nl/>)

Sisow is een dienst die tegen een klein deel van je betalingen alles regelt. Het enige wat je moet doen is registreren en de plugin te installeren.

*(Tot op heden wacht ik tot PWC hier een account maakt voor Husqvarna MX GP. Ik heb er verder nog niet naar gekeken. Er is al eerder duidelijk gemaakt dat het niet verplicht is dat wij dit afmaken. Meer hierover in het volledige TO.)*

# Implementatie slicing

De slicing (losse html en css pagina’s) hebben wij verwerkt in CakePHP door de body in de default pagina te verwerken en de verschillende pagina’s op te halen via een fetch.

Ook hebben we hier en daar kleine onderdelen zoals een countdown balk, deze plaatsen we via een element in de default pagina als hij zichtbaar moet zijn.

# CakePHP CMS

Het grote voordeel dat CakePHP biedt ten opzichte van standaard PHP is het feit dat aanmaken, bewerken en verwijderen van content zwaar vereenvoudigd is. Je maakt allereerst een model aan dat gerelateerd is aan een tabel in de database. In ons geval is dat bijvoorbeeld ‘Album’. Vervolgens kun je binnen het model de validatieregels opgeven die van belang zijn. De ‘Albums’ tabel in de database bevat bijvoorbeeld de kolom ‘name’. Dit staat voor de naam van het album. De invoer van deze kolom mag niet leeg zijn en daarom kan in het model de volgende validatieregel worden opgenomen:

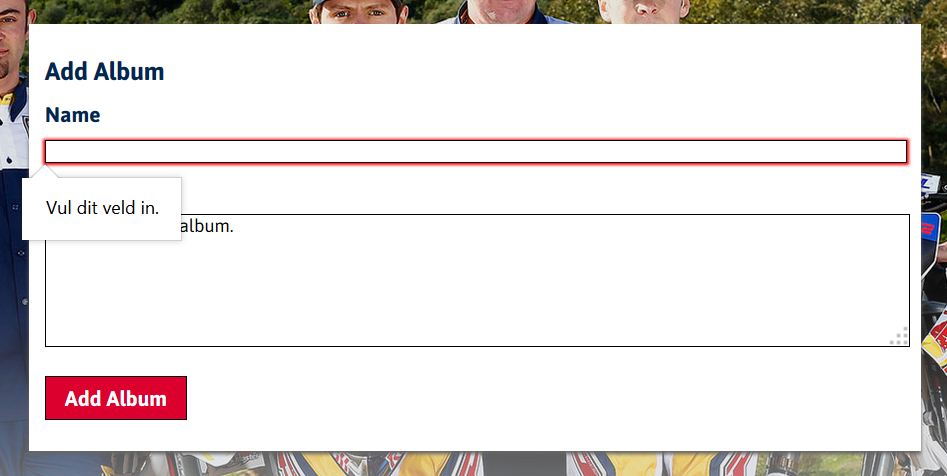
'name' => array(

'rule' => 'notEmpty'

)

Door deze regel controleert CakePHP of name niet leeggelaten is. Het grote voordeel hiervan is dat je niet zelf voor elke database aanpassing die je doet verschillende if/else blokken hoeft te schrijven. Daar komt nog bij dat het zowel client-side controleert of de ingevoerde gegevens wel kloppen (door middel van html5 input validatie) en server-side controleert of de ingevoerde gegevens kloppen.

Het CMS dat ontwikkeld is voor de website van Husqvarna MX bevat voor alle tabellen binnen de database die gewijzigd moeten kunnen worden van dit soort modellen, zodat de validatie volledig afgehandeld wordt door CakePHP.

Een voorbeeld van een foutmelding die CakePHP toont door middel van een validatieregel.

Nadat het model gemaakt is kan aan de hand hiervan een zogenaamde view gemaakt worden. Dit bevat uiteindelijke hetgene wat de gebruiker op zijn scherm te zien krijgt. Bij het aanmaken of bewerken van (in dit geval) een album wordt de volgende code gebruikt om aan CakePHP duidelijk te maken dat er een album aangemaakt moet worden:

echo $this->Form->create('Album');

Vervolgens wordt aangegeven welke velden allemaal zichtbaar moeten zijn. In dit geval zijn dat ‘name’, ‘description’ en een voor de gebruiker onzichtbaar veld dat de ‘user\_id’ van de gebruiker bevat:

echo $this->Form->input('name');

echo $this->Form->input('description', array('rows' => '5'));

echo $this->Form->input('user\_id', array('type' => 'hidden', 'default'=>AuthComponent::user('id')))

Uiteindelijk wordt er aangegeven dat daarmee het formulier klaar is door de volgende regel, die automatisch een verzendknop aanmaakt met de opgegeven tekst:

echo $this->Form->end('Add Album');

Het resultaat hiervan is terug te zien in de bovenstaande afbeelding.

Het enige wat dan nog resteert, is de zogenaamde controller. Daarin wordt de daadwerkelijke functionaliteit gerealiseerd. Omdat dit voorbeeld in het teken heeft gestaan van het aanmaken van een album, zal ik ook de controller code voor het aanmaken van een album gebruiken om een deel van de werking van controllers binnen het CMS uit te leggen.

Allereerst is de naam van de zogenaamde ‘action’ binnen de controller hetzelfde als de naam van de view die op dat moment geopend is. In ons geval is dat nu dus de action ‘add’. Deze gaat allereerst voor ons controleren of de action bereikt is door middel van een gepost formulier:

public function add() {

if ($this->request->is('post')) {

}

}

Vervolgens wordt er een object gemaakt van het model in kwestie, in dit geval dus van een album:

$this->Album->create();

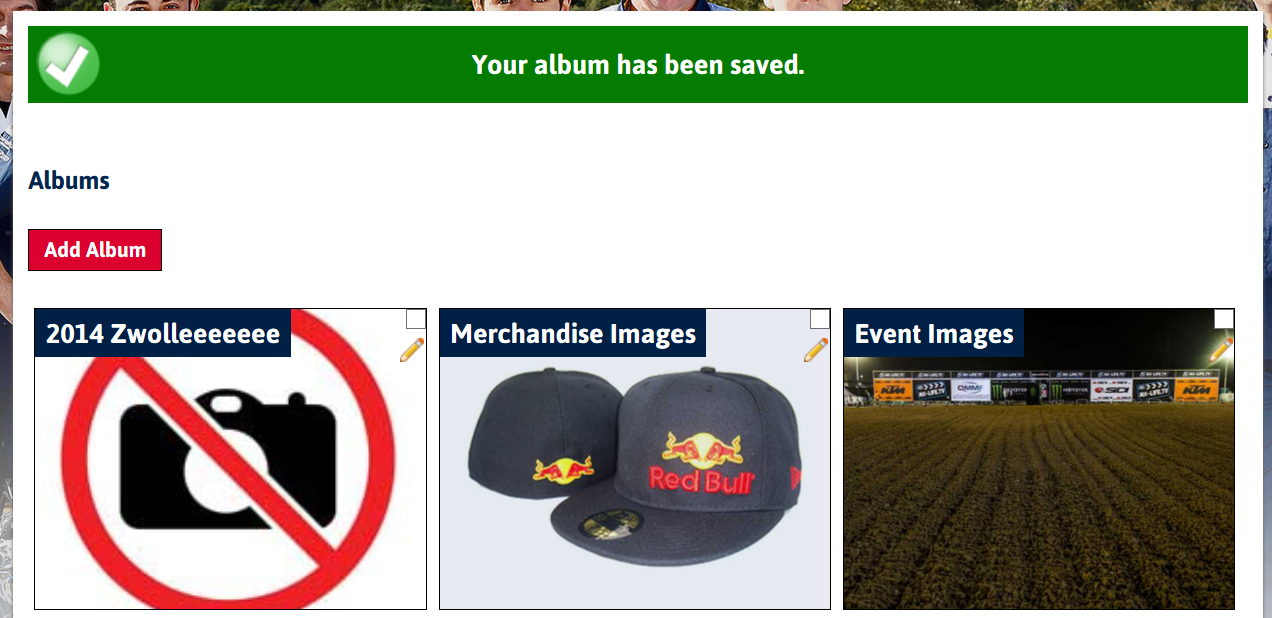
De volgende stap is het controleren of de ingevoerde data voldoet aan de voorwaarden die we in het model gesteld hebben en wanneer dat het geval is wordt deze opgeslagen en wordt een succesmededeling getoond op de CMS pagina:

if ($this->Album->save($this->request->data)) {

$this->Session->setFlash(\_\_('Your album has been saved.'));

return $this->redirect(array('action' => 'cms'));

}



*Een voorbeeld van een succesmelding die CakePHP toont wanneer een gebruiker geldige data heeft ingevoerd.*

Mocht de ingevoerde data niet aan de gestelde voorwaarden voldoen, dan wordt een server-side foutmelding getoond en kan de gebruiker een aanpassing doen om alsnog de juiste data in te voeren:

$this->Session->setFlash(

'Unable to add your album.', 'default', array('class' => 'flashError')

);

In dit geval heb ik album dus als voorbeeld genomen, maar dit principe gaat op voor het complete CMS. Natuurlijk is de specifieke invulling per pagina verschillend, maar het komt uiteindelijk voor elke pagina hierop neer.

# Cake PHP Routing

Voor de homepagina en de teampagina in ons project gebruiken wij het CakePHP routing bestand: Config/routes.php

Dit bestand is nu zo ingesteld dat de url: ‘/’ doorverbindt naar pages/home. Dit zorgt ervoor dat ‘home’ de standaard pagina is die iedereen te zien krijgt als zij onze site bezoeken.

Voor de team pagina hebben we ook een route ingesteld met de url: ‘/team’. Deze verbind weer door naar pages/team.

# Fotobibliotheek

De fotobibliotheek is een systeem dat gebruikers de mogelijkheid geeft om een album aan te maken. Binnen dat album kunnen vervolgens weer foto’s toegevoegd worden.

Een voorwaarde van de fotobibliotheek was dat er van de foto’s die toegevoegd werden ook een kleinere versie (thumbnail) gemaakt zou worden. Om het maken van thumbnails te bewerkstelligen heb ik besloten gebruik te maken van een bestaande functie genaamd: ‘[PHP - easy and smart image resize function](https://github.com/Nimrod007/PHP_image_resize)’. Deze functie is in staat om alle ondersteunde fotoformaten te verkleinen en aangezien het onnodig is om het wiel opnieuw uit te vinden, heb ik daarom besloten hier gebruik van te maken.

Bij het aanmaken van een album wordt de data van het album in de database opgeslagen en wordt er daarnaast een map aangemaakt met als naam het id van het album. Binnen deze map wordt ook een map ‘thumbs’ aangemaakt, waar de thumbnails uiteindelijk in terecht komen. Dit gebeurt met de volgende code:

/\* Create a folder for the album if the folder does not yet exist (which will

\* never be the case, unless albums get deleted from another place than the CMS

\* page

\*/

if (!file\_exists('images/albums/' . $this->Album->getInsertID())) {

// Create an albums dir and inside that create a thumbs dir

mkdir('images/albums/' . $this->Album->getInsertID() . "/thumbs", 0777, true);

}

Vervolgens kunnen er afbeeldingen toegevoegd worden aan het album. Hierbij heb ik besloten gebruik te maken van een input field dat meerdere bestanden tegelijkertijd accepteert. Het grote voordeel hiervan is natuurlijk dat een gebruiker niet tien keer een enkele foto hoeft te uploaden, maar in één keer een hele rits foto’s kan uploaden.

In overleg hebben we afgesproken om de naam van elke foto die geüpload wordt ook in de database op te slaan met daarnaast een ‘album\_id’ en een ‘user\_id’. Dit hebben we gedaan om ervoor te zorgen dat er zicht is op welke gebruiker welke foto’s heeft geüpload. Op deze manier kunnen er in één album ook door meerdere gebruikers foto’s geüpload worden en kunnen we gebruikers alleen het recht geven om hun eigen foto’s te verwijderen.

Als validatie heb ik besloten niet te controleren op extensie, maar op mime type. Het voordeel hiervan t.o.v. controleren op extensie is dat het eventueel misbruik van een gebruiker oppikt. Stel dat een gebruik een virus heeft verwerkt in een virus.exe programma en dit programma hernoemt naar virus.jpg. Wanneer er gecontroleerd wordt op extensie zal virus.jpg succesvol geüpload worden. Dit is natuurlijk niet de bedoeling en zal ook niet het geval zijn bij een controle op mime type. Hierbij wordt namelijk het échte type van het bestand gekeken en deze zal dan ook herkennen dan virus.jpg geen echt jpeg bestand is.

Foto’s kunnen individueel verwijderd worden, maar een album kan ook in zijn geheel verwijderd worden, waarna alle foto’s in het album ook verwijderd worden.

# Event Countdownbalk

Op de homepagina en op de eventspagina wordt een countdownbalk getoond die aftelt tot het eerstkomende evenement. Allereerst is het daarbij van belang om het eerstkomende evenement op te halen:

// Get the first upcoming event

$event = $this->Event->find('first', array(

'conditions' => 'Date >= NOW()'

)

);

Vervolgens wordt er een vergelijking gemaakt tussen de huidige datum en de datum van het eerstkomende evenement, om aan de hand daarvan vast te stellen hoeveel dagen en uren het nog duurt totdat het evenement begint. Allereerst stellen we vast hoeveel uur het nog duurt tot het volgende event:

// The difference between the two events in hours

$timeUntilEvent = round((strtotime($eventDate) - strtotime($currentDate)) / 3600, 1);

Hierna controleren we of het aantal uren tussen de twee groter is dan 24, omdat we op die manier vast kunnen stellen of er ook dagen tussen de huidige datum en het aankomende evenement zitten. Als dat het geval is stellen we vast hoeveel dagen tot het volgende evenement en zo niet weten we al hoeveel uren:

/\* If the difference between the two is bigger than 24, there's more than a day difference, so we need to calculate the days / remaining hours \*/

if ($timeUntilEvent > 24) {

// The difference in days

$daysUntilEvent = floor($timeUntilEvent / 24);

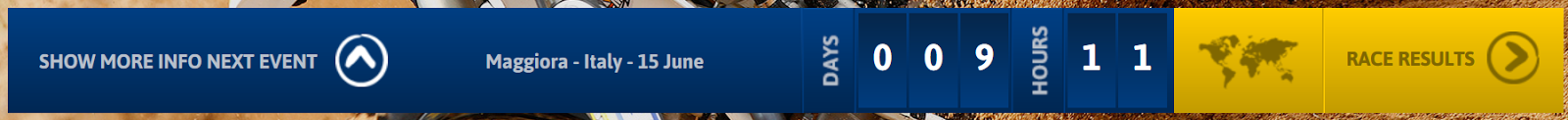
// The remaining hours

$hoursUntilEvent = $timeUntilEvent % 24;

} else {

$hoursUntilEvent = $timeUntilEvent;

}



*De countdownbalk in volle glorie.*

# Sponsors per Event

Voor elk event moest het mogelijk zijn om selectieve sponsors te tonen. De eisen hiervoor waren een beetje ingewikkeld. De bovenste drie zijn altijd de hoofdsponsors Husqvarna, Nestaan en Wilvo. Maar welk van de drie het bovenste blokje opneemt is variabel. In dit voorbeeld staat er Husqvarna bovenaan. Daarom staan Nestaan en Wilvo eronder. Mocht Wilvo bovenaan staan, dan zou Husqvarna het blokje van Wilvo overnemen en blijft Nestaan waar het al staat. De overgebleven twee blokjes onderaan zijn voor twee andere sponsors, die ook per event gekozen kunnen worden. (In het voorbeeld zijn het ook Nestaan en Wilvo, want dit zijn de enige waarvan wij het juiste logo hebben gekregen).

Om dit te verzorgen heeft elk event drie velden met daarin de drie sponsors. ‘main\_sponsor’, ‘sponsor1’ en ‘sponsor2’. Aangezien de drie hoofdsponsors altijd hetzelfde zijn maar op een andere plek, hoef je alleen te selecteren welke bovenaan komt. Sponsor 1 en 2 zijn de twee sponsors onderaan.

In de View worden de sponsors van het event opgehaald

$sponsors = $this->requestAction('events/getSponsorsOfEvent/' . $events[0]["Event"]["id"]);

$event werd eerder al opgehaald bij event countdown balk (zie vorig hoofdstuk), $events[0] is het vroegst opkomende event. Als antwoord op de request krijg je de sponsor objecten die bij het event horen. Die worden op de volgende manier geprint.

if($sponsors[0]['Sponsor']['Name'] == "Wilvo"){

echo "<li class=\"fright\"><a href='<http://www.husqvarna.com>'>

<img src='/images/event\_logo2.png' alt=''></a></li>";

}

else{

echo "<li class=\"fright\"><a href='<http://www.wilvo.nl>'>

<img src='/images/event\_logo3.png' alt=''></a></li>";

}

De sponsor URL is de link naar de sponsor website. Als de bovenste sponsor Wilvo is, dan komt Husqvarna op de plek van Wilvo te staan. Zo niet, zet gewoon Wilvo op zijn eigen plek. Hetzelfde gebeurt voor Nestaan. Daarna worden simpelweg de overige twee sponsors getoond.

Voor het aanpassen van de sponsors per event moet de naam worden opgehaald vanuit de DB en in de lijst worden gezet. Voor het ophalen van de sponsors en de namen wordt nogmaals een request gebruikt. Deze request geeft alle sponsors terug op volgorde van ID. De eerste drie zijn altijd de hoofd sponsoren.

Binnen de lijsten zijn de namen verbonden aan een ID. Dit is door CakePHP ook makkelijk gemaakt. Om dit te doen moet je een array maken met als key het ID, en als waarde de tekst die je wilt tonen.

$sponsorList = array();

foreach($sponsors as $sponsor){

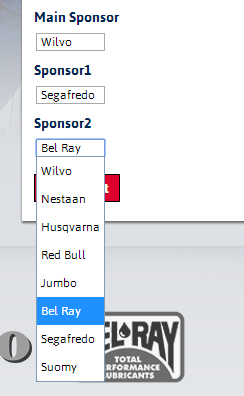
$sponsorList[$sponsor['Sponsor']['id']] = $sponsor['Sponsor']['Name'];

}

Ik was hier even door verward, omdat ik het tegenovergestelde zou verwachten. Ik heb niet onderzocht waarom het zo is. Deze lijst kun je daarna koppelen aan een input. (Voor meer over velden, zie CakePHP CMS)

echo $this->Form->input('main\_sponsor', array('options' => array\_slice($sponsorList, 0, 3), 'default' => $event['Event']['main\_sponsor']));

echo $this->Form->input('sponsor1', array('options' => $sponsorList, 'default' => $event['Event']['sponsor1']));

echo $this->Form->input('sponsor2', array('options' => $sponsorList, 'default' => $event['Event']['sponsor2']));

Na de input te hebben gekoppeld aan een object attribuut kun je er bepaalde configuratie aangeven. Hier gebruik ik ‘options’ om de lijst van namen te maken en ‘default’ om te zorgen dat hij standaard op de juiste sponsor staat.

# Nieuwsoverzicht

Het nieuwsoverzicht toont alle artikelen. Hier wordt prioriteit genegeerd om simpelweg de artikelen te weergeven in volgorde van datum. (Voor meer over prioriteit, zie: ‘Nieuws Prioriteit’) Dit was programmeer technisch heel makkelijk om te doen in CakePHP. Voor het ophalen van ‘alles’ van iets heeft Cake een voorgebakken actie die op alles is toe te passen die er als volgt uit ziet.

$NAME = $this->-OBJECT->find('all', array('order' => array('id' => 'desc')));

Voor articles is dit:

$articles = $this->Article->find('all', array('order' => array('id' => 'desc')));

Met ->find(‘all’) zou je alle data al kunnen krijgen, maar hier voegen wij ook de volgorde toe. Deze data stuur je naam de view door vooraf een variabel te maken.

$this->set('articles', $articles);

Bij de functie ‘set’ is de eerste parameter de naam die je het variabel wilt geven en de tweede parameter de waarde. Dit kan natuurlijk alles zijn.

Voor deze pagina bestond ook al een slicing, dus deze hebben we simpelweg dynamisch gemaakt door de benodigde data erin te zetten en loops te maken voor herhalende objecten.

# Nieuws View

De nieuws view toont een specifiek artikel. Binnen het CakePHP framework is het heel makkelijk een specifiek object op te halen door in de controller deze code uit te voeren:

$NAME = $this->OBJECT->findById($id);

Voor articles ziet het er dan zo uit:

$article = $this->Article->findById($id);

Met de findByID functie die al verwerkt is in het CakePHP framework krijg je het article object dat het id nummer heeft die wordt opgevraagd.

Om deze variabele ook in de view te kunnen gebruiken gebruiken we dit stukje code:

$this->set('article', $article);

Dit is ook weer een functie die uit het CakePHP framework komt. De eerste parameter is de naam die je de variabele wilt geven in de view. De tweede parameter is waarmee je de variabele voor de view wilt vullen, in dit geval dus $articles.

Om deze pagina te bereiken gebruik je simpelweg de url: articles/view/(id).

Bijvoorbeeld: articles/view/3

De controller gebruikt het id nummer van de link om het bijbehorende artikel bij te zoeken.

Onderaan de pagina worden ook nog de 4 laatst aangemaakte artikelen getoont. Dit wordt op dezelfde manier gedaan als Nieuwsoverzicht, maar bij deze pagina worden er maximaal 4 getoont.

# Nieuws tags

Aan alle artikelen op de website kunnen tags toegevoegd worden. Dit kan simpelweg door één of meerdere tags in te vullen in het tags veld. Ik heb het zo ingesteld dat wanneer iemand meerdere tags aan een artikel wil toevoegen hij deze moet scheiden door een komma en een spatie.



*Een voorbeeld van een artikel met meerdere tags, gescheiden door een komma en een spatie.*

Binnen de betreffende action van de Articles controller worden de tags na elke komma gevolgd door een spatie omgezet naar een element in een array. Dit gebeurt met de volgende code:

$tags = explode(", ", $this->request->data["Article"]["Tags"]);

In het geval van een nieuw artikel kunnen de tags simpelweg toegevoegd worden aan de database, maar in het geval van een bewerkt artikel zal bekeken worden of er nieuwe tags zijn die toegevoegd moeten en of de “oude” tags van het artikel nog steeds bestaan. Dat gebeurt op de volgende manier:

/\* Loop through the old and new tags and see which old tags still exist and

\* which new tags should be added

\*/

$oldTagsThatStillExist = array();

$newTagsToBeAdded = array();

foreach ($tags as $tag) {

$alreadyExists = false;

foreach ($oldTags as $oldTag) {

if ($oldTag["Tag"]["value"] == $tag) {

$alreadyExists = true;

array\_push($oldTagsThatStillExist, $tag);}}

if (!$alreadyExists) {

array\_push($newTagsToBeAdded, $tag);}}

Na dit stukje code weet je dus welke nieuwe tags er zijn toegevoegd en welke oude tags nog bestaan. Aan de hand daarvan kun je vervolgens oude tags die niet meer bestaan verwijderen:

/\* Now loop through the old tags and the old tags that still exists and

\* remove old tags that no longer exist

\*/

$tagsToBeDeleted = array();

foreach ($oldTags as $oldTag) {

$tagShouldRemain = false;

foreach ($oldTagsThatStillExist as $oldTagThatStillExists) {

if ($oldTag["Tag"]["value"] == $oldTagThatStillExists) {

$tagShouldRemain = true;}}

if (!$tagShouldRemain) {

array\_push($tagsToBeDeleted, $oldTag);}}

// Delete the tags that no longer exist

foreach ($tagsToBeDeleted as $tag) {

$this->Article->Tag->delete($tag["Tag"]["id"]);

}

Het enige wat vervolgens nog resteert, is de nieuwe tags toevoegen aan de database.

# Nieuws zoekbalk

De zoekbalk stelt de gebruiker in staat om specifieker te zoeken naar bepaalde artikelen op de site. In eerste instantie werd alleen gekeken of de zoekopdracht overeen kwam met tags die aan een artikel gekoppeld waren, maar na overleg met de opdrachtgever hebben we besloten ook te zoeken op de titel en inhoud van een artikel. Vervolgens worden op de gewone nieuwspagina enkel de artikelen getoond die gerelateerd zijn aan de zoekopdracht. Dit gebeurt op de volgende manier:

if ($this->request->is('post') && $\_POST["searchQuery"] != "") {

$articles = $this->Article->Tag->find('all', array('conditions' => array('Tag.value' => $\_POST["searchQuery"])

));

} else {

$articles = $this->Article->find('all', array('order' => array('id' => 'desc')));

}

# Racers View

De racers view toont een specifieke racer. Binnen het CakePHP framework is het heel makkelijk een specifiek object op te halen door in de controller deze code uit te voeren:

$NAME = $this->OBJECT->findById($id);

Voor racers ziet het er dan zo uit:

$racer = $this->Racer->findById($id);

Met de findByID functie die al verwerkt is in het CakePHP framework krijg je het racer object dat het id nummer heeft die wordt opgevraagd.

Om deze variabele ook in de view te kunnen gebruiken gebruiken we dit stukje code:

$this->set('racer', $racer);

Dit is ook weer een functie die uit het CakePHP framework komt. De eerste parameter is de naam die je de variabele wilt geven in de view. De tweede parameter is waarmee je de variabele voor de view wilt vullen, in dit geval dus $racer.

Om deze pagina te bereiken gebruik je simpelweg de url: racers/view/(id).

Bijvoorbeeld: racers/view/1

De controller gebruikt het id nummer van de link om de bijbehorende racer bij te zoeken.

Ook de resultaten voor de racer moet hier zichtbaar zijn. Dit gebeurd door dit stukje code:

$results = $this->Racer->Result->find('all');

Hier worden de resultaten opgehaald zodat deze later in de view laten zien kunnen worden. Om ervoor te zorgen dat er in de controller van Racers ook Results opgehaalt kunnen worden is er een koppeling nodig. De koppeling tussen racers en results is geregeld in de model met deze code:

Voor de racers:

public $hasMany = 'Result';

En voor de results:

public $belongsTo = array(

       'Racer' => array( 'className' => 'Racer', )

);

# Merchandise overzicht

Merchandise overzicht werkt dankzij CakePHP voor het grootste deel hetzelfde als het nieuws overzicht. Je kunt hiernaar kijken voor meer informatie over overzicht pagina’s. Merchandise overzicht is uniek omdat je producten ook moet kunnen kopen. Door op een product te klikken, krijg je meer informatie over dat product en kun je het in je winkelwagen zetten.

Het verkrijgen van meer informatie wanneer iemand op een product klikt hebben we gerealiseerd met jQuery. Wanneer er op een product geklikt wordt, wordt er een pop-up met meer details en een ‘Add To Cart’ knop getoond. Het voordeel hiervan is dat een potentiële klant niet hoeft te wachten tot een nieuw venster geopend is en vervolgens weer op de terug knop moet drukken om het overzicht weer voor zich te krijgen. Alles kan op één pagina afgehandeld worden wat zowel de snelheid als de gebruiksvriendelijkheid goed doet.

Dit hebben we gerealiseerd door een onclick event aan de ‘BUY THIS ITEM’ knop van alle producten toe te voegen:

onclick="showProductPopup('<?php print($albumID); ?>', '<?php print($imageName); ?>', '<?php print($photoID);?>', '<?php print($productDescription);?>')"

Deze stuurt de gegevens die nodig zijn om een gedetailleerder beeld van het product te schetsen mee aan de jQuery functie ‘showProductPopup’, die er op zijn beurt voor zorgt dat de pop-up getoond wordt:

// This function shows a product popup

function showProductPopup(albumID, imgName, photoID, productDescription)

{

$("#merchandisePopup").css('background-image', 'url(/images/albums/' + albumID + '/' + imgName + ')');

$("#merchandisePopup").css('display', 'block');

var popupContent = '<div class="productPopupDescription"><p>' + productDescription + '</p>';

popupContent += 'Size: <select class="sizeSelector"><option>S</option><option>M</option><option>L</option><option>XL</option></select>';

popupContent += '<input type="button" value="Add to cart" class="button addToCartButton"></div>';

$("#merchandisePopup").html(popupContent);

// Don't close the box when the user clicks the sizeSelector

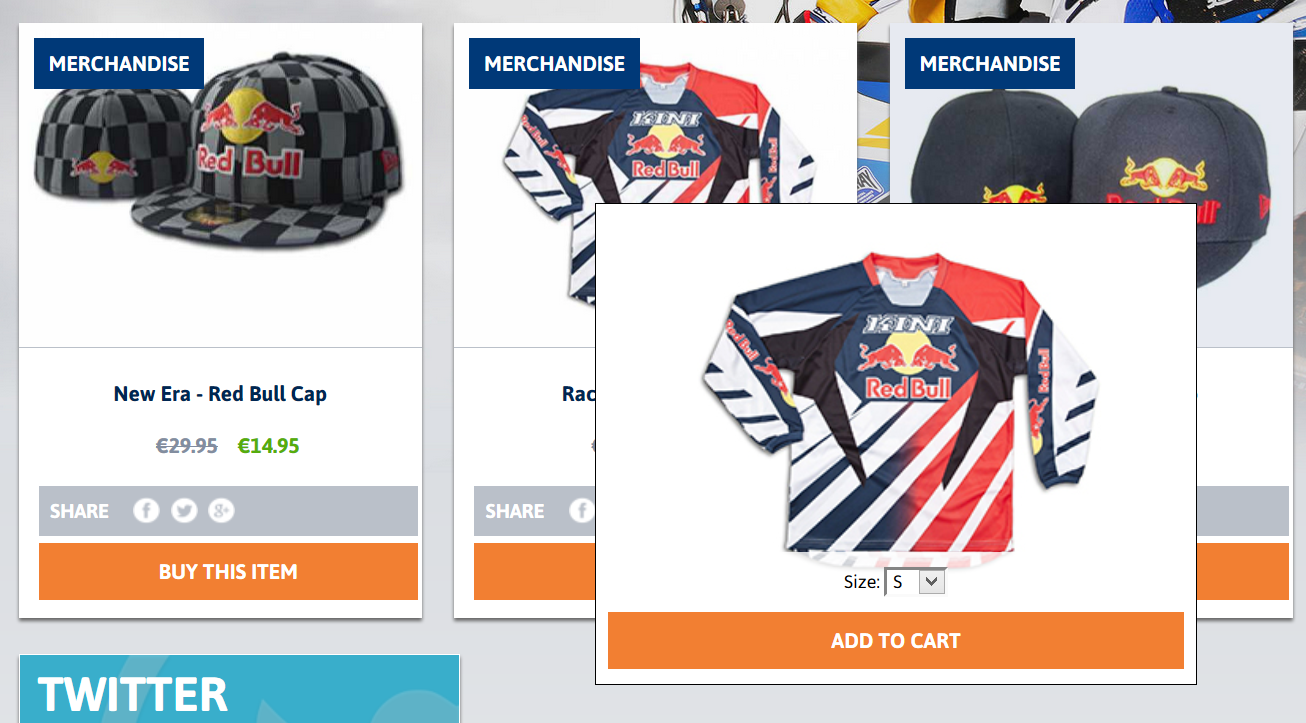
$(".sizeSelector").click(function(e)

{

e.stopPropagation();

});

}



*Een voorbeeld van een pop-up waarmee de gebruiker een product aan zijn winkelwagen kan toevoegen.*

Wanneer er vervolgens binnen de pop-up geklikt wordt, verdwijnt de pop-up weer.

# Merchandise webshop

BAS

# Nieuws prioriteit

Sommige nieuws artikelen zijn belangrijker dan anderen. Als één van de racers iets heeft gewonnen, wilt de editors misschien dat dit artikel voor de komende paar weken meer exposure krijgt. Daarom kan hij dit artikel een verhoogde prioriteit geven. Dan komt dit artikel hoger op de home pagina te staan.

Er zijn 5 niveaus van prioriteit, de eerste twee zijn overheersend en blijven bovenop totdat de prioriteit ervan verandert. 1 t/m 3 wordt gesorteerd op datum, maar een hogere prioriteit blijft één week langer staan. Hoe lang dit is wordt berekend met de volgende formule:

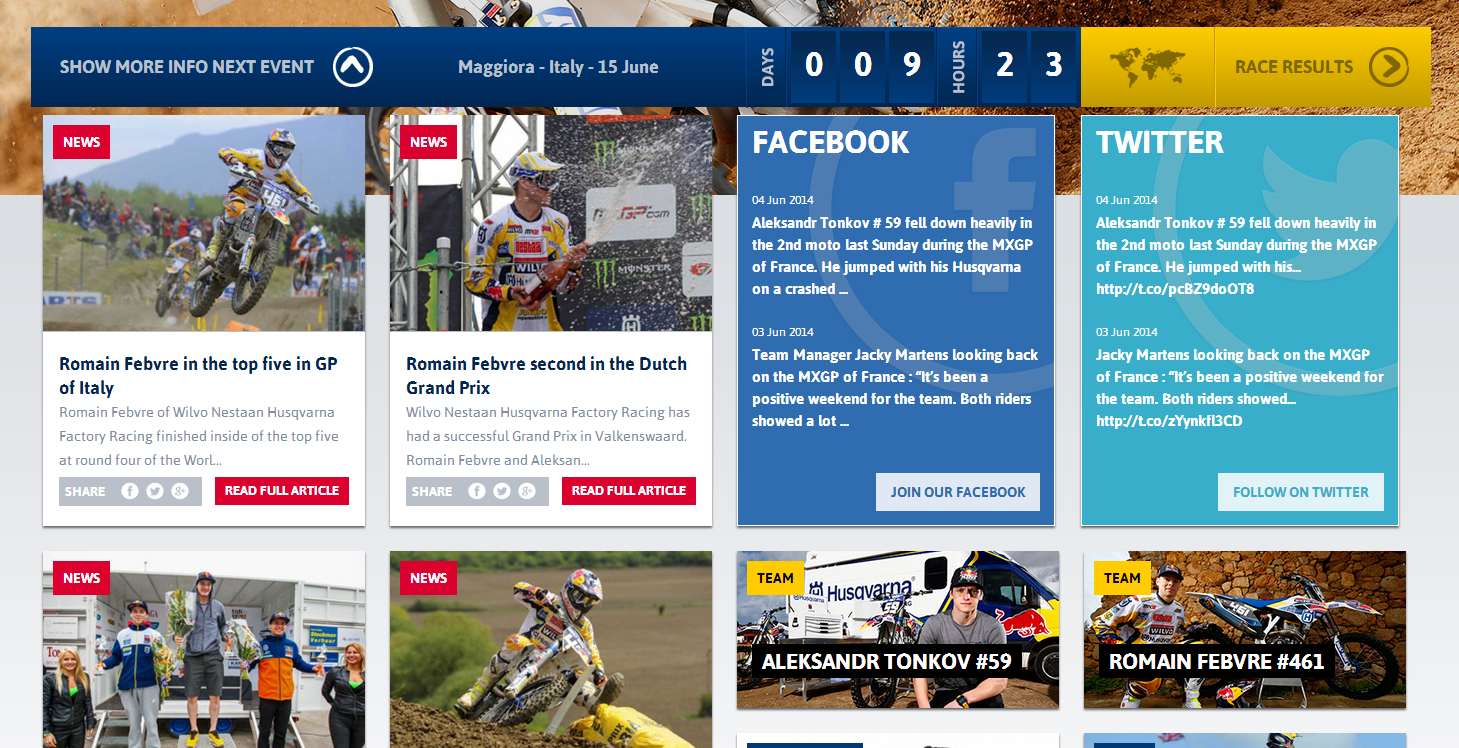
score = 0 - verschil in dagen + 7 \* prioriteit

Als het verschil in dagen omhoog gaat, gaat de score omlaag. Als de prioriteit omhoog gaat, gaat de score omhoog. De 7 staat voor het aantal dagen dat een artikel meer geldig is dan een artikel in de lagere prioriteit. Tot slot wordt 0 alleen op de overzicht pagina getoond. In de code ziet dit er als volgt uit:

$score = 0 - $difference + 7 \* $priority;

Voor elke van de artikelen wordt de score berekend en gekeken of deze de hoogste is, als dat zo is, voeg hem toe aan de lijst. Als er 4 artikelen zijn gesorteerd, wordt de lijst op de pagina getoond.

# Home pagina



Op de home pagina staan veel verschillende objecten die op een zeer specifieke volgorde moeten worden weergeven. Wij gebruiken een javascript plugin genaamd ‘js-masonry’ die div’s in een grid-layout plaatst binnen een andere div. Het gedrag hiervan is heel ingewikkeld en wordt op letterlijk elke pagina gebruikt. Voor meer hierover, zie de website <http://masonry.desandro.com/>.

Wij hebben niet voor de implementatie hiervan gezorgd, maar gebruiken het zoals het is geleverd. Daardoor heeft het een vreemde eigenschap waar het horizontaal alles uitlijnt. Daardoor is er een probleem als één van de elementen langer is dan de andere. Daarom hebben wij dit ten alle kosten moeten vermijden. Dat is de reden waarom de Facebook en Twitter blokjes zo lang zijn; om te vermijden dat ze langer moeten zijn bij lange posts. Want als de één langer is dan de ander, ziet de pagina er opeens heel vreemd uit.

Voor de volgorde van de elementen gebruiken we een foreach die door de artikelen loopt. Maar bij bepaalde tellen print hij eerst de nodige elementen voor het verder gaat. Dat ziet er ongeveer zo uit (pseudo code):

$count = 0;

foreach ($sorted as $article) {

printarticle();

$count++;

if ($count == 2)

printsocialmedia();

}

printothers();

Als het niet duidelijk is, na het tweede artikel wordt de social media getoond. Na alle artikelen komt ‘de rest’. De rest zijn de blokjes voor de racers, het nieuwste merchandise artikel en de link naar de video’s.