Afstudeerscriptie

Interactieve webapplicaties bouwen met Cappuccino

Auteur:
Jeroen Tietema

Begeleider:
Jan Mooij

13 december 2009

Voorwoord

Dit document beschrijft de werkzaamheden tijdens mijn afstudeerstage voor de opleiding Informatica aan de Hogeschool Utrecht. De afgelopen 4 maanden is er hard gewerkt aan een nieuw CMS voor Voidwalkers wat tevens mijn eigen bedrijf is.

Afstuderen binnen mijn eigen bedrijf was een unieke ervaring. De combinatie van ondernemen en programmeren heb ik als prettig ervaren en bevestigd dat een eigen bedrijf een goede keuze is. Daarnaast was het erg motiverend om aan een opdracht te werken waar ik zelf de vruchten van zou plukken. Natuurlijk had ik dit nooit helemaal alleen gekund. Daarom wil ik een aantal mensen specifiek bedanken.

Allereerst wil ik Mattijs Hoitink bedanken voor het een half jaar naast mijn zijde uit te houden. Hij was altijd beschikbaar voor advies, brainstorm sessies en technische discussies over de architectuur van het CMS en heeft nuttige feedback gegeven over mijn scriptie.

Daarnaast wil ik Ronald van Zuijlen bedanken. Ronald heeft mij begeleid als externe en deed dit volledig in zijn vrije tijd. Hij heeft mij regelmatig opgezocht op kantoor en vele goede suggesties gegeven over het project en de scriptie.

Ook wil ik een woord van dank geven aan Jan Mooij. Jan heeft mij begeleid vanuit de Hogeschool Utrecht en daarvoor eindeloos veel revisies van dit document doorgelezen en van feedback voorzien.

Als laatste wil ik nog Marieke ten Velde bedanken die voor praktisch elke pagina van mijn scriptie bergen suggesties wist te doen op het gebied van zinsbouw en spelling. Zonder de inspanningen van Jan en Marieke was deze scriptie maar een fractie zo leesbaar geweest.

Samenvatting

Tijdens de afstudeerperiode is er door de afstudeerder gewerkt aan het bouwen van een CMS voor het bedrijf VOIDWALKERS. Uniek aan deze situatie was dat de afstudeerder mede-eigenaar is van het bedrijf VOIDWALKERS.

Ondanks dat VOIDWALKERS al wat langer bestaat, heeft het in zijn verleden vooral op dienstverlening gericht en niet op productontwikkeling. VOIDWALKERS merkte dat dit de groei in de weg stond. De grootste behoefte was vooral een degelijk CMS (Content Management Systeem).

Dit CMS moest naast visueel aantrekkelijk en gebruikersvriendelijk ook snelle ontwikkeling van websites ondersteunen. Om dit te bewerkstelligen is er gekozen voor een andere aanpak dan in veel traditionele CMS applicaties.

De implementatie van het nieuwe CMS is geheel geschied in Javascript en Objective-J, wat een taal bovenop Javascript is. Hiervoor heeft de afstudeerder de mogelijkheden van nieuwe frameworks moeten verkennen, zoals het Cappuccino framework. Dit framework maakt het ontwikkelen van webapplicaties mogelijk die zich gelijkwaardig gedragen aan desktop applicaties. Hierdoor kan ook dezelfde gebruikersvriendelijkheid behaald worden als van desktop applicaties.

Bij het ontwerpen is gekozen voor een strikte scheiding tussen voor- en achterkant. De CMS applicatie is opgedeeld in een *client* en een *server*. Door de *client* geheel in Javascript te bouwen, werd de *server* kant een stuk minder complex. De code bevatte bijna alleen nog maar domein logica en weinig applicatie logica. Dit opende de deuren voor code-generatie. Door gebruik te maken van het Doctrine framework (een ORM pakket voor PHP) kon gemakkelijk PHP-code en databasetabellen gegenereerd worden op basis van configuratie files. Dit versnelt de ontwikkeling van websites op basis van dit CMS aanzienlijk.

Ontwikkeling met Cappuccino bleek complexer dan gedacht. Tevens zaten erg nog veel bugs in. Dit, samen met de beperkte aanwezige tools, maakte dat veel tijd is besteed aan het uitpluizen van Cappuccino-specifieke problemen. Gelukkig bleek het ontwikkelen van de *server* kant een stuk simpeler.

Het is de afstudeerder gelukt om een werkend product op-te-leveren dat het merendeel van de functionaliteiten bevat. Het product is een gezonde basis voor VOIDWALKERS om in de toekomst nieuwe projecten mee te realiseren. Hiermee zijn de doelen van de stage succesvol behaald.

Inhoudsopgave

Hoofdstuk 1

Inleiding

Één van de dingen die mij altijd heeft tegengestaan aan programmeren voor het web is de beperking door de mogelijkheden van HTML formulieren en links. Een webapplicatie is eigenlijk een berg textbestanden aan elkaar geknoopt met links en formulieren. Hoewel ik al vóór mijn opleiding de kracht en de potentie hiervan inzag, bleven webapplicaties toch een statisch gevoel houden. Ze misten de dynamiek die desktop applicaties wel hebben. Ik miste altijd een echte API[1] zoals die van Java Swing[2].

Ik was dan ook blij verrast toen ik het Cappuccino framework ontdekte. Hiermee kun je echte applicaties bouwen in de *webbrowser*. Het biedt een rijke verzameling aan klassen om je applicatie mee te bouwen en abstraheert volledig weg van de HTML. Eindelijk een webframework voor "echte applicaties".

Zelf wilde ik graag afstuderen binnen mijn eigen bedrijf. De voornaamste motivatie hiervoor is dat ik tijd kon besteden aan een product dat ook echt gebruikt gaat worden na mijn afstuderen. Na mijn kennismaking met Cappuccino was meteen het idee geboren dit framework te gebruiken om een CMS applicatie te bouwen.

Deze unieke win-win situatie zou mij in staat stellen mijn afstudeerproject te vullen met een uitdagende en leerzame opdracht én zou een waardevol product opleveren voor het bedrijf.

[1] Application Programming Interface, bibliotheken die de programmeur kan gebruiken voor het bouwen van je applicatie.

[2] Swing is een bibliotheek van Java die gebruikt kan worden om Grafical User Interfaces (GUI's) te bouwen voor desktop applicaties

Hoofdstuk 2

De probleemstelling en planning

VOIDWALKERS is een onderneming opgericht door Mattijs Hoitink en Jeroen Tietema (de afstudeerder). Begonnen in januari 2007 als parttime onderneming, VOIDWALKERS is inmiddels uitgegroeid tot een fulltime onderneming.

2.1 De probleemstelling

Groeien betekent niet alleen meer klanten en meer opdrachten, maar ook groei in omvang van de opdrachten. VOIDWALKERS merkt dat dit niet kan zonder eigen producten. Een veelvuldige vraag van klanten is of VOIDWALKERS een eigen CMS (Content Management Systeem) heeft. Het is gebleken dat klanten grotere opdrachten uitbesteden aan bedrijven die eigen producten hebben. Dit komt omdat klanten niet steeds voor de extra ontwikkeltijd willen betalen om alles "vanaf niets" op te bouwen. Ook zijn de risico´s kleiner als er al met werkende software wordt gestart.

> Kortom: *Voidwalkers heeft behoefte aan een eigen CMS om aan de verwachtingen van klanten te voldoen.*

2.2 De Opdracht

2.2.1 In het kort

De opdracht zal bestaan uit het ontwerpen van een CMS en het bouwen van een werkend prototype.

2.2.2 Omschrijving

Het CMS zal bestaan uit een *client* en een *server* component. De *client* zal volledig in de *browser* draaien. Hiermee wordt bedoeld dat de *client* uit één enkele HTML pagina bestaat en dat alle functionaliteit in Javascript is geïmplementeerd. De *client* hoeft dan alleen voor de data met de *server* te communiceren. De user-interface van de *client* kan hierdoor veel sneller reageren dan bij traditionele webapplicaties. Om dit mogelijk te maken wil VOIDWALKERS gebruik maken van nieuwe technieken die beschreven worden in de HTML 5 specificatie. Het *server* component zal een webservice zijn geschreven in PHP.

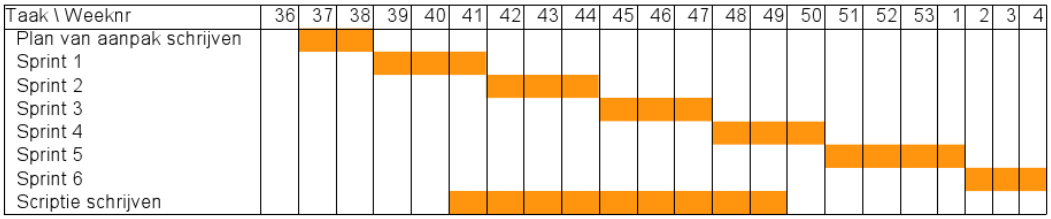
Activiteiten van de afstudeerder zullen zijn:

1. Onderzoeken of Cappuccino & Objective-J de meeste geschikte technologieën zijn voor de *client*
2. Het ontwerpen en bouwen van zowel de *server* en de *client*
3. Het schrijven van documentatie voor de *server* en de *client*

4. Eventueel het schrijven van tools die de ontwikkeling met het CMS moeten vergemakkelijken

2.3 De planning

Tijdens de afstudeerperiode zal er in zes korte ontwikkelsprints een CMS gebouwd worden waarmee VOIDWALKERS de toekomst tegemoet kan treden. De planning is gebaseerd op SCRUM. SCRUM zal verder toegelicht worden in paragraaf 6.4.



Figuur 2.1: Tijdsbesteding in een Gantt chart.

Bij aanvang zullen globaal alle gewenste functionaliteiten in kaart gebracht worden. Bij aanvang van een sprint zullen er een aantal functionaliteiten ingepland worden.[1] Deze zullen dan aan het begin van de sprint verder ontworpen worden en vervolgens gerealiseerd.

[1] De functionaliteit (SCRUM stories) zullen toegelicht worden in paragraaf 7.1, de volledige lijst van SCRUM stories zit bij de bijlagen.

Hoofdstuk 3

De producten

Het gebouwde CMS zal een opensource product worden . Een aantal van de hieronder beschreven producten zijn dan ook publiek beschikbaar. Om die reden, en om ze tot een zo groot mogelijk publiek toegankelijk te maken, zijn die producten in de Engelse taal geproduceerd. Dit zijn voornamelijk de SCRUM stories en de documentatie in de broncode en de documentatie op de wiki.

3.1 Plan van aanpak

Het plan van aanpak is begonnen als afstudeervoorstel, maar later uitgewerkt. In dit plan van aanpak heb ik een initiële lijst van SCRUM stories opgeleverd en een planning voor de eerste twee sprints (iteraties). Dat ik niet alle sprints heb ingepland is intentioneel. Het is in een SCRUM project ook niet de bedoeling om alles tot het eind vast in te plannen. Wel heb ik een aantal concept planningen gemaakt om een inschatting te maken van de haalbaarheid om alle stories in het afstudeerproject te realiseren. Het originele plan van aanpak uit september is meegeleverd als bijlage bij dit document.

3.2 SCRUM stories

Een SCRUM story is een korte beschrijving van functionaliteit vanuit de ogen van een gebruiker. Je beschrijft in een kort verhaal wat je als gebruiker wilt kunnen doen. Deze beschrijving dient als basis voor het ontwikkelen van de functionaliteit. Zelf heb ik er voor gekozen aan de story schermontwerpen en eventuele technische ontwerpen toe te voegen. In de planning worden een aantal stories per sprint uitgekozen om te realiseren.

Interessant is dat er twee soorten gebruikers zijn geïdentificeerd: de eindgebruiker en de ontwikkelaar van de website. Een CMS product is echte middleware en wordt ook door ontwikkelaars gebruikt om eerst de website mee te bouwen. In de stories is dit terug te zien doordat ook een aantal stories vanuit het perspectief van een website ontwikkelaar zijn geschreven. zijn.

3.3 Scriptie

De scriptie is een verplicht onderdeel van het afstuderen. In de scriptie zal het verloop van het afstuderen beschrijven en de gemaakte keuzes toelichten.

3.4 Het CMS

Het CMS is het eindproduct van de afstudeeropdracht. Het bestaat uit een *client* component geschreven in Objective-J en een *server* component geschreven in PHP. Het is wenselijk dat het product gereed is voor productie aan het eind van de afstudeerperiode. Aangezien de tijd (één semester) en de middelen (één student) vast staan, is automatisch de functionaliteit variabel. Belangrijk is dat de code van dusdanige kwaliteit is, dat het in de toekomst in productie kan draaien. Het is niet de bedoeling snel een prototype te

fabriceren, maar om een product te bouwen op degelijke design patterns zodat de code in de toekomst beheersbaar blijft.

3.5 Documentatie

Beheersbaarheid gaat hand in hand met documentatie. De makkelijkste manier van documenteren vind ik in de code. Het is weinig moeite om boven een methode of klasse te beschrijven wat het doet. Ik streef er naar om alle klassen en methoden van documentatie te voorzien. Zo is goede API documentatie die ervoor zorgt dat de code voor VOIDWALKERS te snappen is en blijft.

De API documentatie is voor een buitenstaander echter te specifiek. Er zal ook documentatie vanaf een hoger niveau moeten zijn om mensen bekend te maken met het systeem. Op de wiki zullen een aantal artikelen geschreven worden om dit doel te bereiken.

Er wordt nagedacht over een online demo voor eindgebruikers te voorzien van een video instructeur die een live rondleiding door het CMS verzorgt.

Hoofdstuk 4

De projectorganisatie

De afstudeerder: Jeroen Tietema
De docentbegeleider: Jan Mooij
De inhoudelijke begeleider: Ronald van Zuijlen & Mattijs Hoitink

Na overleg met Jan Mooij is besloten zowel Ronald als Mattijs aan te dragen als begeleiders.

Ronald van Zuijlen zal begeleiding en advies geven over de ontwikkeling van het CMS en feedback geven over de algehele koers van het project. Ronald is CMS consultant bij VLC en heeft ervaring met meerdere open- en closed-source CMS pakketten.

Mattijs Hoitink zal begeleiding geven op technisch gebied. Mattijs is afgestudeerd informaticus en gecertificeerd PHP ontwikkelaar en is mede-eigenaar van Voidwalkers.

Jan Mooij zal de afstudeerder begeleiden vanuit de HU.

Daarnaast zal er een projectpagina te vinden zijn op de project-portal van Voidwalkers `http://project.voidwalkers.nl/projects/show/cms`. Deze is voor iedereen toegankelijk.

Hoofdstuk 5

Analyse van de mogelijkheden

Uiteraard zou de gemakkelijkste oplossing voor het CMS probleem van VOIDWALKERS zijn om één van de bestaande open source PHP CMS pakketten te gebruiken die op het internet beschikbaar zijn. Maar er zijn een aantal redenen waarom VOIDWALKERS denkt dat het zelf ontwikkelen van een CMS de moeite waard is.

5.1 Overzicht bestaande CMS'en

Hieronder volgt eerst een overzicht van de drie meest populaire PHP CMS producten en wat hun kenmerken zijn. Dit stelt ons later in staat een goede vergelijking te maken. Veel van de voor- en nadelen gelden ook voor de andere CMS´en.

5.1.1 Wordpress

Wordpress is van origine een weblog-systeem. Sinds een aantal jaar heeft het de mogelijkheid standaard-pagina's te beheren. Sindsdien wordt het ingezet door webdevelopers voor simpele websites. De ingebouwde media bibliotheek stelt gebruikers in staat plaatjes en bestanden te uploaden en deze toe te voegen aan hun pagina's.

De kracht van Wordpress is dat het heel geschikt is voor het bewerken van de standaard content (blog-artikelen en pagina's). Als de gebruiker meer wil, loopt hij al snel tegen beperkingen aan. Wanneer er extra velden in de content opgenomen moeten worden, kunnen dit alleen tekstvelden zijn van beperkte lengte. Er kan wel meer bereikt worden door zelf plug-ins te schrijven. De plugin architectuur is eigenlijk bedoeld voor het toevoegen van functionaliteit op de bestaande content, in plaats van het definiëren van nieuwe extra content types.

5.1.2 Joomla!

Joomla! is een PHP CMS pakket voor websites van middelmatige grootte. Ooit begonnen als afsplitsing van het CMS Mambo, heeft Joomla! een redelijk berucht verleden op het gebied van beveiligingslekken. Qua architectuur heeft Joomla! een MVC structuur, maar niet object georiënteerd. Dit maakt de code lastig te overzien omdat er veel gebeurt in de global scope. Dat wil zeggen: alle variabelen zitten samen in één grote bak in het geheugen. Het probleem is dat plugin-ontwikkelaars het risico lopen variabelen te overschrijven van de core, of van andere plugins. Los van het vergrootte risico op bugs heeft dit ook veel invloed op de veiligheid van het systeem.

Joomla! dankt zijn populariteit aan het feit dat het gemakkelijk te implementeren is. Hierdoor kunnen Joomla! ontwikkelaars verhoudingsgewijs veel functionaliteit bieden voor weinig geld. De huidige economische crisis wakkert dit voordeel extra aan.

5.1.3 Drupal

Drupal is eigenlijk een grote versie van Joomla!. Drupal is vanwege de extra functionaliteit die het bezit, meer geschikt voor grotere websites. Op het roerige verleden van

beveiliging na, geldt wat voor Joomla! geldt ook voor Drupal. De architectuur is grotendeels hetzelfde maar iets netter. Drupal geniet op het moment meer aandacht in de enterprise wereld dan Joomla!, vanwege het volwassen karakter en de extra functionaliteit.

5.2 Voor- en nadelen zelfbouw

Het is belangrijk om goed na te denken over de consequenties van het zelf bouwen van een CMS. Vaak wordt gekozen voor zelfbouw op grond van " Not invented here" argumenten. "Not invented here" wil zeggen dat programmeurs niet met andermans code willen werken. Ze denken dat de code slecht is omdat ze deze zelf niet snappen, en hebben onwil om deze te leren. In dit hoofdstuk wordt een kritische blik geworpen op het zelf bouwen van een CMS. Zelfbouw is voor ons geen vlucht van het onbekende.

5.2.1 Voordelen

Belangrijkste voordelen zijn de keuze van de technologie en het bepalen van de architectuur. Het stelt ons in staat een technologie te gebruiken die mogelijk nog niet bestond op het moment dat reeds bestaande producten werden gebouwd. Wellicht hunnen dingen gemakkelijker en sneller gerealiseerd worden dan met de verouderde technologie van bestaande producten.

Ook het bepalen van de architectuur is een belangrijk voordeel. Bestaande producten hebben niet altijd de juiste architectuur om te kunnen voorzien in uitbreidingen die je wilt bouwen op het product. Op het moment manifesteert dit zich in huidige webapplicaties die zich niet lenen voor zwaar Javascript gebruik. De meeste applicaties gebruiken een MVC (Model View Controller)[1] structuur waarbij elke view een stukje HTML is. Elke actie roept een nieuwe *request* op die de MVC structuur doorloopt en een nieuwe HTML pagina genereert. Als de applicatie volledig in Javascript gebouwd wordt, willen we niet steeds nieuwe HTML van de *server*. We willen lokaal de HTML manipuleren met Javascript en van de *server* alleen de data ontvangen. Het laden van nieuwe HTML zou de staat van de Javascript vernietigen. (zie figuur 5.1)

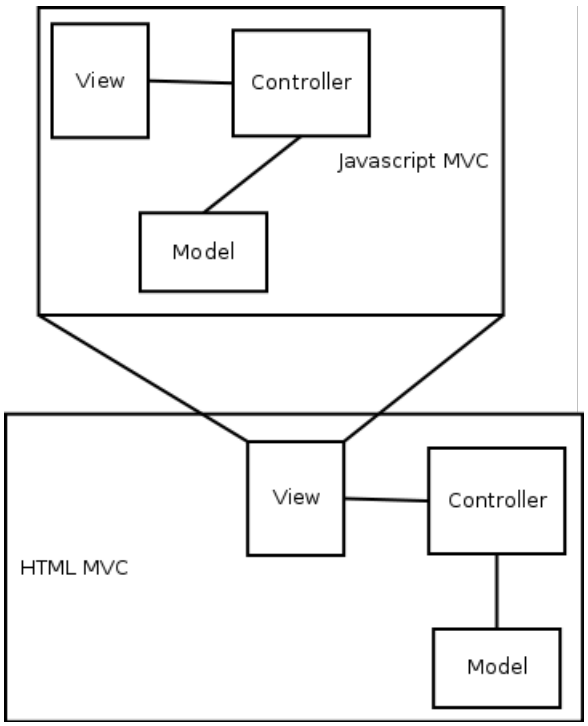
Een heel ander voordeel is dat je product niet snel het doelwit van mal-ware zal worden doordat je onbekend bent en weinig installaties hebt op het internet. Veel klanten bestellen eens een website en houden zich niet bezig het het up to date houden. Beveiligingslekken in Open Source producten worden aan de wereld publiek gemaakt bij het beschikbaar stellen van de patch. Dit geeft malware schrijvers de kans spambots en wormen te schrijven die inbreken op alle instanties die niet up to date zijn. Uiteraard vervalt dit argument zodra ons eigen product populair wordt.

5.2.2 Nadelen

Naast voordelen zijn er ook nadelen aan het zelf bouwen van een CMS. Ten eerste is er de grote tijdsinvestering die noodzakelijk is om een werkend systeem te produceren dat enigszins vergelijkbaar is met de concurrentie. Zeker voor een eerste product kan dit een probleem zijn, omdat in de beginfase er nog geen ander product is dat verkocht kan worden.

Een ander nadeel van veel nieuwe code is dat er veel bugs in zitten. De code heeft nog geen tijd gehad zich te bewijzen, wat betekent dat er verhoudingsgewijs veel bugs naar

[1] MVC wordt uitgebreid behandeld in 8.1.1 op pagina 25



Figuur 5.1: De twee MVC structuren op/in elkaar

voren zullen komen bij de eerste implementaties. Bestaande pakketten zoals Drupal en Joomla! zijn door vele duizenden mensen gebruikt en dus getest. Er zullen verhoudingsgewijs nu minder bugs in zitten.

Wanneer er bugs gevonden worden ben je 100% verantwoordelijk om die te verhelpen. Achter veel Open Source projecten zit een grote community. Het is een kwestie van de nieuwe versie downloaden en je bent klaar. Je eigen product zal in eerste instantie die gemeenschap niet hebben.

Naast het gemis van een gemeenschap, mis je ook een stukje naamsbekendheid. Met de beweging richting Open Source en aanhoudende bezuinigingen genieten Drupal en Joomla! een flinke hype. Beiden zijn Open Source en erg goedkoop te implementeren. Vaak hebben bedrijven de neiging de beslissingen van anderen te kopieëren. Naamsbekendheid en de publiekelijke keuze van mensen voor jouw product is dan heel belangrijk.

5.3 De gekozen aanpak

	Javascript MVC	OO/PHP5	Kort ontwikkel traject	Veel software beschikbaar
Wordpress	nee	nee	ja	ja
Joomla!	nee	nee	ja	ja
Drupal	nee	nee	ja	ja
Zelfbouw	ja	ja	nee	nee

Er is gekozen voor zelfbouw met als belangrijkste reden het zelf kunnen bepalen van de architectuur. In de aanloop naar de afstudeerperiode is gekeken naar de beschikbare Javascript frameworks waarmee onze applicatie gebouwd kon worden. Hier kwamen twee opties uit: Sproutcore en Cappuccino. Beide waren nog niet eerder gebruikt voor het

bouwen van een CMS. Zelfbouw was de enige optie als we een van die frameworks wilden gebruiken. Omdat beide frameworks strikte eisen stellen aan de architectuur, viel het niet in te passen in een bestaande applicatie.

Daarnaast wilde VOIDWALKERS graag een CMS dat gebaseerd is op degelijke software ontwikkelmethoden in verband met de onderhoudbaarheid. De voorkeur ging sterk uit naar een PHP5 object georiënteerd CMS en veel van de bestaande systemen zijn dat niet.

Uiteraard zorgt dit voor extra investering vanuit VOIDWALKERS, maar deze kan grotendeels gedekt worden door de afstudeeropdracht waardoor het geen extra kosten met zich meebrengt.

Hoofdstuk 6

Gebruikte technieken

Dit hoofdstuk geeft een overzicht van de talen, frameworks en tools die bij de afstudeeropdracht zijn gebruikt.

6.1 Objective-J: Cappuccino

Voor het bouwen van de *client* van het CMS is gekozen voor het Cappuccino framework, geschreven in Objective-J. Objective-J en Cappuccino zijn betrekkelijk nieuwe technologieën (één jaar oud). Objective-J is een uitbreiding op Javascript. Zowel Javascript als Objective-J waren voor de afstudeerder onbekend. De syntax van Objective-J (zie figuur 6.1) is vrijwel identiek aan die van Objective-C, een uitbreiding op de taal C.

```
@implementation ContentView : CPView
{
    CPView _documentContent;
    CPShadowView _shadow;
}

- (id)initWithFrame:(CGRect)aFrame
{
    self = [super initWithFrame:CGRectMake(0,0,CGRectGetWidth(aFrame) - 25, CGRectGetHeight(aFrame

    if (self)
    {
        _documentContent = [[CPView alloc] initWithFrame:CGRectMake(25,25, CGRectGetWidth(aFrame)
        [_documentContent setBackgroundColor:[CPColor whiteColor]];
        // add shadow around document
        _shadow = [[CPShadowView alloc] initWithFrame:[_documentContent bounds]];
        [_documentContent addSubview:_shadow];
        [self addSubview:_documentContent];
    }
    return self;
}
```

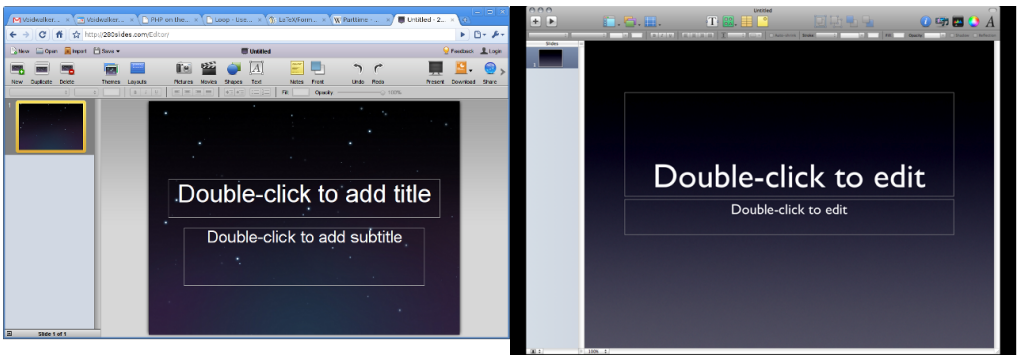
Figuur 6.1: Een stukje Objective-J code

Er zijn meerdere redenen waarom gekozen is voor het uitbreiden van Javascript. De eerste is natuurlijk om extra features aan de taal toe te voegen. Voorbeelden hiervan zijn overerving gebaseerd op klasse en het import statement.

Een andere reden vinden we in het Cappuccino framework. Het Cappuccino framework is een poort van Cocoa naar het web. Cocoa is geschreven in Objective-C en is het standaard GUI framework voor ontwikkeling op de Mac. Één van de lange termijn doelen van de auteurs van Objective-J en Cappuccino is dat je bestaande Cocoa applicaties kunt hercompileren naar Cappuccino applicaties. Zo kunnen bestaande applicaties van de Mac met minimale aanpassing draaien op het web. Om dit mogelijk te maken moet de Cappuccino-code zo veel mogelijk overeenkomen met de Cocoa code. Vandaar dat Cappuccino niet in pure Javascript is geschreven.

De reden dat er voor Cappuccino is gekozen, is dat het een hoger abstractieniveau heeft dan bijvoorbeeld Sproutcore. Sproutcore, en andere javascript frameworks, zijn nog heel erg DOM (Document Object Model)[1] gebaseerd. Voor Cappuccino is de DOM

[1]Het Document Object Model is een object-georinteerde benadering van gestructureerde documenten zoals HTML-, XHTML- en XML-documenten.



Figuur 6.2: Links een Cappuccino applicatie in de *browser* en rechts een native Cocoa applicatie op de Mac

een implementatie detail. Als ontwikkelaar heb je in principe alleen te maken met pure Objective-J klassen en niets met de DOM. Het is dan ook de bedoeling dat in de toekomst meerdere render-mogelijkheden voor Cappuccino ontwikkeld kunnen worden. Bijvoorbeeld een pure canvas oplossing of een oplossing op basis van SVG in plaats van DOM / HTML.

Daarnaast heeft Objective-J ook een sterke voorkeur omdat dit een aantal tekortkomingen in Javascript aanpakt.

6.2 PHP: Zend Framework & Doctrine

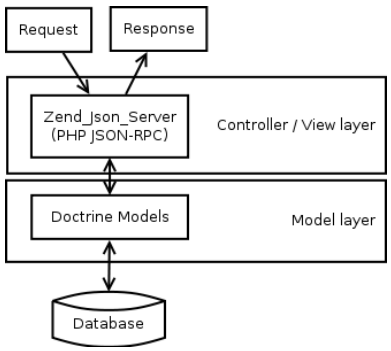
Voor het bouwen van de webservice van het CMS is gekozen voor het Zend Framework in combinatie met Doctrine. Beiden zijn geschreven in PHP. PHP is een taal voornamelijk bedoeld voor het web. Het leent een groot deel van zijn syntax van Perl en veel van de functienamen komen direct uit C (eigenlijk roept de PHP functie vaak ook de C equivalent direct aan). In tegenstelling tot Objective-J bestaat PHP al een tijdje.

Het Zend Framework is een componenten framework. Uit het framework gebruiken wij het component Zend_Json_Server. Dit is een *server* implementatie van het JSON-RPC protocol. Dit is het protocol dat de *client* en *server* met elkaar praten. Zend_Json_Server verzorgt de vertaling van de JSON-berichten naar 'methoden aanroepen' in platte PHP klassen en het terugvertalen van PHP return types naar JSON berichten. Het gebruik van dit component versimpelt de ontwikkeling van de *server* aanzienlijk.

Doctrine is een ORM (Object-relational Mapping)[2] pakket. Doctrine stelt de programmeur in staat database-schema's en PHP objecten te genereren die op de database werken, zodat die deze niet zelf geschreven hoeven te worden. Naast het feit dat dit de ontwikkeling aanzienlijk versneld, opent dit ook de deur om dynamisch code te genereren voor het *server* component. Een uiteindelijk doel is (door configuratie) code van je back-end te genereren en op die manier de back-end gemakkelijk geschikt te maken voor elke willekeurige soort content. Doctrine verzorgt het genereren van de code en zelf hoeven we dan alleen nog maar Reflection[3] uit te voeren op de PHP klassen die Doctrine genereert.

[2]ORM is een programmeer techniek om te converteren tussen object georiënteerde modellen en database tabellen.

[3]Reflection is een techniek waarmee een programma over zijn eigen code kan redeneren en zelfs kan aanpassen in sommige gevallen



Figuur 6.3: De *server* architectuur

6.3 Tools: Git & Redmine

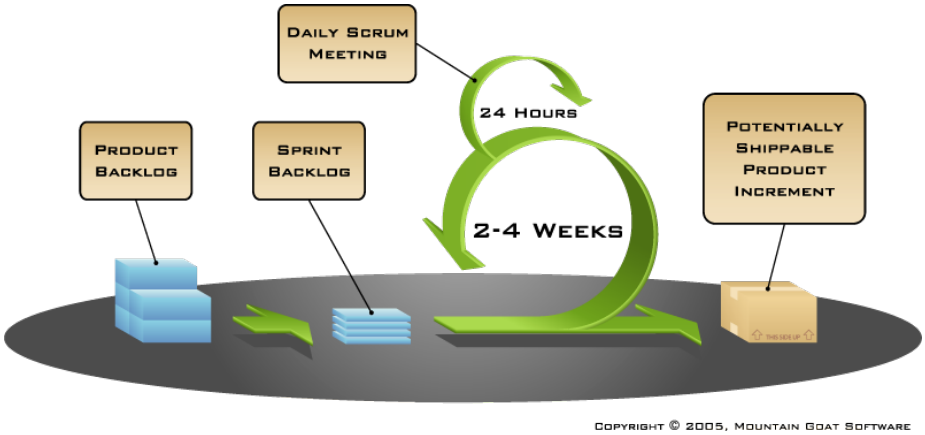
Om het project soepel te laten verlopen en overzicht te houden, zijn er een aantal tools gebruikt.

Git is gebruikt als versie beheersysteem. Dit helpt veel risico's te vermijden. Het beschermt tegen het kwijtraken van code door defect raken van hardware. Het stelt je in staat gemakkelijk kopieën van de code te verspreiden. Tevens houdt het versies bij. Dit zorgt dat er altijd terug gegaan kan worden naar een vorige versie, mocht de code niet meer goed functioneren.

Redmine is een webbased projectmanagement portal. Het biedt een ticket systeem aan, wiki, documentbeheer en integratie met Git. Via Redmine kun je al je informatie over je project beheren op één plek. Het stelt je in staat versies te plannen en hier functionaliteit aan toe te kennen.

6.4 SCRUM

Voor de planning van het project heb ik voor een SCRUM-achtige aanpak gekozen. SCRUM is een agile projectmanagement-methode die zijn naam dankt aan de scrum bij rugby. Ondanks dat het geen afkorting is, wordt het vaak met hoofdletters geschreven. Deze traditie lijkt te stammen van Ken Schwaber (één van de bedenkers van SCRUM), die in de titel van zijn paper SCRUM met hoofdletters schreef.



Figuur 6.4: Schematisch overzicht van projectverloop in SCRUM

Onderdelen van SCRUM die ik gebruikt heb zijn: de backlog, stories, de sprints en later ook de daily SCRUM meeting.

Bij aanvang van het project heb ik alle mogelijke functionaliteit van het te maken CMS kort beschreven in SCRUM stories. Deze lijst met stories heb ik gebruikt bij het inplannen van het project. In de planning heb ik zes sprints van drie weken gepland. Een sprint is een korte periode waarna een werkend product wordt opgeleverd. Hierdoor is er heel vroeg in het project al een werkend product. Dit maakt de risico's van het project een gemakkelijk in te schatten. Na elke sprint is bepaald wat er in de volgende sprint het meest noodzakelijk was, en werd dat ingepland.

De SCRUM stories worden verder toegelicht in paragraaf 7.1.

Hoofdstuk 7

Functioneel ontwerp

Het functioneel ontwerp is het totaal van de SCRUM stories. Bij het opstellen van de SCRUM stories is geprobeerd ook naar de behoeften van de ontwikkelaar te kijken. Veel van de huidige CMS pakketten op de PHP markt richten zicht alleen op de eindgebruiker, en dus op functionaliteit, maar zijn onder de motorkap een ramp om mee te werken. Om van dit CMS-product een succes te maken moeten zowel eindgebruikers als ontwikkelaars het prettig vinden.

Voor het functioneel ontwerp is een lijst met SCRUM stories gemaakt die functionaliteit beschrijven voor zowel de eindgebruiker als de ontwikkelaar.

7.1 Wat is een SCRUM story?

Idealitair bestaat een SCRUM story uit één regel. Die regel beschrijft wie, wat, waarom moet kunnen doen. Bijvoorbeeld: “Als eindgebruiker wil ik mijn plaatjes in het CMS kunnen bewerken, zodat ik dan geen extra software nodig heb.” De gedachte hierachter is dat een SCRUM story op een klein kaartje moet passen en te gebruiken is bij mindmapping / brainstorm sessies.

Zelf heb ik gekozen voor iets uitgebreidere stories. Ik heb de stories bijgehouden op een Wiki en had dus niet met de fysieke restrictie van kaartjes. Alle stories hadden een titel, zodat ik gemakkelijk naar ze kon verwijzen, een samenvatting in één regel en een functioneel overzicht. In het functioneel overzicht wordt de story afgebakend (wat hoort er bij en wat niet?). Optioneel kan er aan een story nog een schermontwerp worden toegevoegd en/of technische specificaties. Dit heb ik gedaan om alle documentatie die bij een story hoort, goed bij elkaar te hebben. De hieronder genoemde voorbeelden zijn echte stories uit het project. Ze zijn in het Engels omdat ze onderdeel uitmaken van de publieke documentatie.

7.1.1 Voorbeeld story gebruiker: Edit basic content

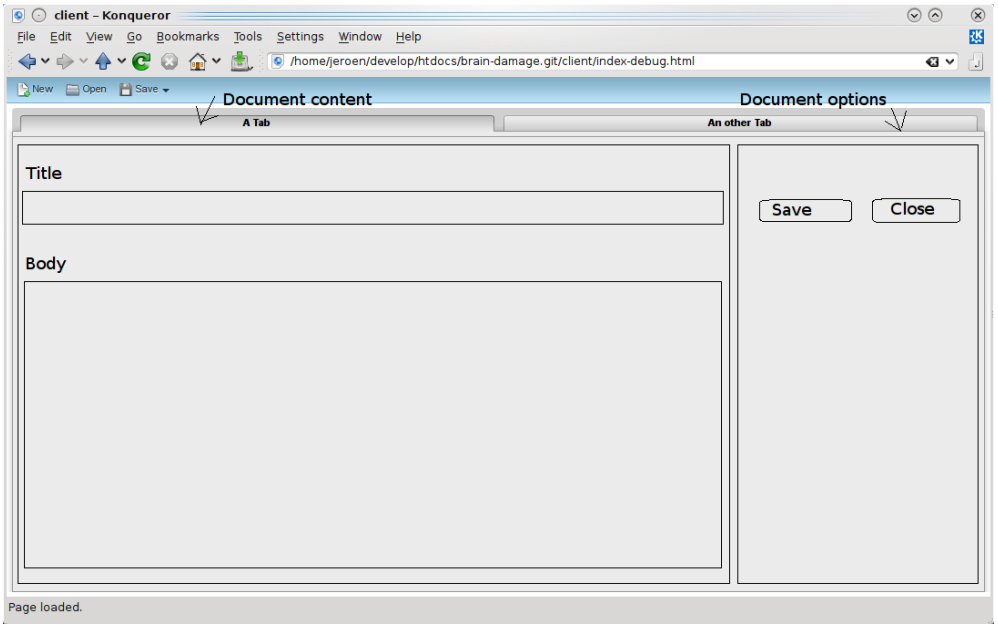
Summary

As a user, I should be able to add and edit basic content like pages. Pages should at least have a title and a body.

Functional overview

After adding new content or editing existing content the application should show all the Widgets filled with the content. After editing the content the user should be able to either save the content or discard the changes. Saving the changes should persist them in the database.

Screen design



7.1.2 Voorbeeld story ontwikkelaar: XML content specification

Summary

As a developer I want to link my Doctrine models and types with an easy XML specification.

Functional overview

The Doctrine models and the type system should be linked via a configuration file and not hardcoded. The CMS will refer to the configuration file to find out which code corresponds to which type.

Note that this file will be generated in the future: Code generation

Technical design

Steps to be taken:

Design a XML file specification Update the code to read the content types from the XML file instead of using hardcoded values. Make sure the server sends enough meta data to the *client* that it can work with the models without knowing about the XML. An example XML:

```
<types>
  <type>
    <name>page</name>
    <fields>
      <field type="string">title</field>
      <field type="html">body</field>
      <field type="relation" relation="user">author</field>
    </fields>
  </type>
  <type>
    <name>user</name>
    <fields>
      <field type="string">name</field>
      <field type="string">password</field>
    </fields>
  </type>
</types>
```

7.2 De SCRUM stories als Functioneel ontwerp

Elke story beschrijft een stukje functionaliteit. De totale lijst met stories omvat alle functionaliteit die in het CMS moet komen. Om een idee te krijgen van wat er gepland stond, is hieronder een lijst met titels van de stories opgenomen. De volledige stories zullen als bijlage bij dit document zitten.

- Edit (Basic) content

- Basic Navigation
- Tree navigation
- XML content specification
- Number Widget (Spinnerbox)
- Relations (Widget)
- Date Widget
- HTML Widget
- Authentication
- Media library (Images support)
- Access control lists (Authorization)
- Image manipulation
- Status page
- Code generation
- Searching
- Modularity
- Google maps support/Widget
- Video support (youtube/viddix/?)

Alle stories staan ongeveer op volgorde waarin ze geïmplementeerd zijn. Dit omdat de prioriteit zo ligt of omdat er een technische afhankelijkheid is. De stories zijn dusdanig klein dat er gemakkelijk twee of drie in een sprint pasten. Als dit niet kon met een story, dan moest deze gesplitst worden in twee stories. Een voorbeeld hiervan is de navigatie. De Basic Navigation was een platte lijst met content. In de sprint erna is een hiërarchische (Tree) navigatie geïmplementeerd.

7.3 Verantwoording van de gekozen functionaliteit

Bovengenoemde stories zijn natuurlijk niet uit de lucht komen vallen. Een groot deel (ongeveer de helft) van de stories bevatten functionaliteit die noodzakelijk zijn voor een CMS. Eerst zal ik uitleggen wat VOIDWALKERS onder een CMS verstaat. Een Content Management Systeem is een applicatie waarmee een 'niet technische gebruiker' de inhoud (content) van zijn website kan beheren. Het is voor ons belangrijk om te kijken wat de meeste gebruikers als content op hun website hebben staan, en tools te bouwen die deze die content kunnen beheren. Hieronder hebben wij geschaard het hebben van een "Rich Text"/HTML editor, platte tekstvelden, relaties kunnen leggen tussen content (bijvoorbeeld een gebruiker als auteur toevoegen aan een artikel) en plaatjes op de website zetten. Daarnaast moet de gebruiker ook gemakkelijk zijn content kunnen vinden in de applicatie. Er moet dus een systeem komen om eenvoudig door alle content te kunnen navigeren.

Naast de vereiste functionaliteit die een CMS een CMS maakt, hebben we ook wat functionaliteit gekozen die "extra" is en vernieuwend moet zijn. Dit zijn onder andere de

"*Image manipulation*", "*Google maps support*" en "*Video support*". *Image manipulation* betekent dat de eindgebruiker simpele bewerkingen op plaatjes kan doen vanuit het CMS. Hierbij moet gedacht worden aan een foto 90 graden roteren of een plaatje bijsnijden. De reden dat wij dit willen, is dat gebruikers nu vaak een apart programma moeten opstarten op hun computer om plaatjes te bewerken. Dit komt regelmatig voor. Het is dan ook de mening van VOIDWALKERS dat plaatjes bewerken een essentieel onderdeel is van het bewerken van de content van je website, dus dat tools daarvoor in het CMS ook voorhanden moeten zijn. *Google maps* en *video support* zijn twee vormen van *content* die steeds vaker voorkomen. Als we kijken hoe ze verwerkt zitten in de CMS´en van nu worden ze behandeld als platte tekst. Vaak moet je op Google maps of Youtube de code voor je video of map genereren en deze knippen/plakken naar een tekstveld in het CMS. VOIDWALKERS zou graag zien dat de gebruiker deze *content* in het CMS kan samenstellen. Hierdoor worden maps en video's als op zichzelf staande *content* types erkent en hoeft de gebruiker minder kennis van zaken te hebben.

Naast alle functionaliteit voor de eindgebruiker stond ook de ontwikkelaar hoog op de agenda bij het ontwikkelen van het CMS. Veel van de huidige PHP-systemen zijn lastig aan te passen doordat geen rekening gehouden is met het feit dat andere ontwikkelaars misschien wel iets zouden willen aanpassen of toevoegen. Bij veel Java systemen zie je dat daar meer rekening mee gehouden is, maar dat het alsnog veel werk is om aanpassingen uit te voeren. Om de concurrentiepositie te verbeteren moet met het CMS snel te ontwikkelen zijn. Er zijn drie stories geformuleerd die hier aandacht aan besteden, namelijk: "*XML specification*", "*Code generation*" en "*Modularity*". De eerste twee hebben met elkaar te maken. Het idee is dat met het systeem code gegenereerd kan worden en dat door configuratie (XML specification) generieke componenten met die gegenereerde code kunnen werken. Hierdoor kunnen de zich herhalende ontwikkeltaken geautomatiseerd worden. Hierbij kan gedacht worden aan het opstellen van databases en het schrijven van PHP-code die op die database werkt. Daarnaast moet de story over modulariteit zorgen dat het CMS eenvoudig uit te breiden is. Natuurlijk is hier niet alleen in deze story aandacht aan besteed, maar tijdens het hele ontwikkelprocess. De intentie van de story is om een moment in te lassen om de modulariteit te testen en mogelijke problemen op te lossen.

Hoofdstuk 8

Detail ontwerp

Veel van het ontwerp van het CMS wordt bepaald door de frameworks die gebruikt zijn. De frameworks hebben we dan ook niet willekeurig gekozen, maar juist omdat ze deze de architectuur afdwingen / voorzien waar we naar op zoek waren.

De grootste invloed op de architectuur van het CMS heeft het gebruik van het Cappuccino framework. VOIDWALKERS denkt dat één van de redenen waarom CMS applicaties zo langzaam innoveren, is dat het gebruik van veel Javascript in een gegenereerde HTML pagina erg lastig is. Het is moeilijk assumpties te maken in de Javascript code over de HTML, aangezien deze door elk stukje applicatie anders gegenereerd wordt. Het aantal combinaties van HTML pagina's en Javascript functionaliteiten neemt snel toe en de complexiteit is al gauw niet meer te overzien.

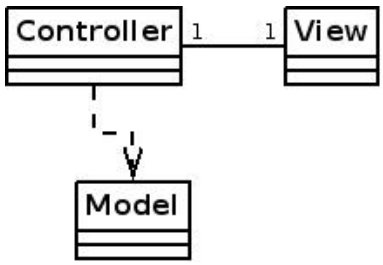
VOIDWALKERS wil dit probleem oplossen door de verantwoordelijkheden duidelijk te scheiden. Het manipuleren van de HTML wordt volledig de verantwoordelijkheid van de Javascript op de *client*. De *server* voorziet alleen de data. Dit is fundamenteel anders dan bij traditionele PHP CMS applicaties. Daarbij gegeneert de *server* de HTML en voert de Javascript hooguit kleine manipulaties uit.

8.1 Concepten

Voordat ik verder ga met het bespreken van het ontwerp wil ik eerst een aantal concepten uitleggen. Sommige concepten zijn bekend in de informaticawereld. Andere zijn specifiek voor deze applicatie.

8.1.1 Het *Model-View-Controller pattern*

Voor het begrijpen van de architectuur van de applicatie is het belangrijk om te weten wat het *MVC* (*Model View Controller*) *pattern* is. Het *MVC pattern*, een *design pattern*, zegt dat je de verantwoordelijkheden in je applicatie kunt onderverdelen in drie categorieën, te weten: *Model*, *View* en *Controller*, en dat je altijd klassen moet schrijven die slechts de verantwoordelijkheden uit één categorie op zich nemen. Dit zorgt voor een onderhoudbare nette architectuur, die in de toekomst ook nog gemakkelijk is aan te passen. De *Controller* klassen zijn bedoeld voor de logica van de applicatie. Een voorbeeld hiervan is: "Als de gebruiker op de knop drukt, dan slaan we het document op." Dit wordt bepaald door de Controller. De *View* klassen bepalen hoe de applicatie er uitziet. Die bepaalt welk soort button er getoond wordt en waar. Als er op de knop gedrukt wordt geeft de View hooguit het signaal door aan de Controller die bij hem hoort. In principe heeft een *View* altijd één *Controller* en andersom. De *Model* klassen bewerken de data. Ook de domeinspecifieke regels die bij de data horen zitten in de *Model* klassen. Zo kunnen deze specifieke regels overal hergebruikt worden. Voorbeeld van zulke regels zijn: "een wachtwoord moet langer dan 7 karakters zijn" of "een artikel moet altijd een titel hebben en deze moet beginnen met een hoofdletter". De relaties tussen de klassen zijn weergegeven in figuur 8.1. Samenvattend: *Model* bevat gegevens, *View* definiërt presentatie en *Controller* definiërt logica.

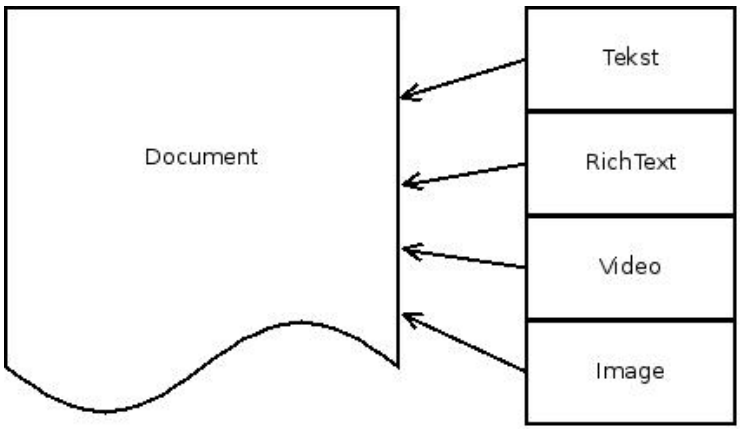


Figuur 8.1: De relaties tussen Model, View en Controller.

8.1.2 Document concepten

In het CMS bewerken we documenten. Hieronder beschrijf ik wat een document is en de gebruikte concepten bij de ontwikkeling van de documentmodule.

Een document is, naar onze definitie, een verzameling content die samen een logisch geheel vormt. Een voorbeeld van een document is een artikel of deze scriptie bijvoorbeeld. Een document kan verschillende soorten content in zich hebben. Bijvoorbeeld plaatjes, platte tekst of tekst met opmaak. Deze soorten content in een document noem ik "primitieve types" of "primitieve content". Binnen het CMS zullen we een aantal primitieve types als bouwsteentjes hebben voor het maken van documenten. Hoe méér van die bouwsteentjes we maken, hoe complexer de documenten zijn die we met het systeem kunnen samenstellen.



Figuur 8.2: Voorbeeld opbouw van een document uit primitieve types

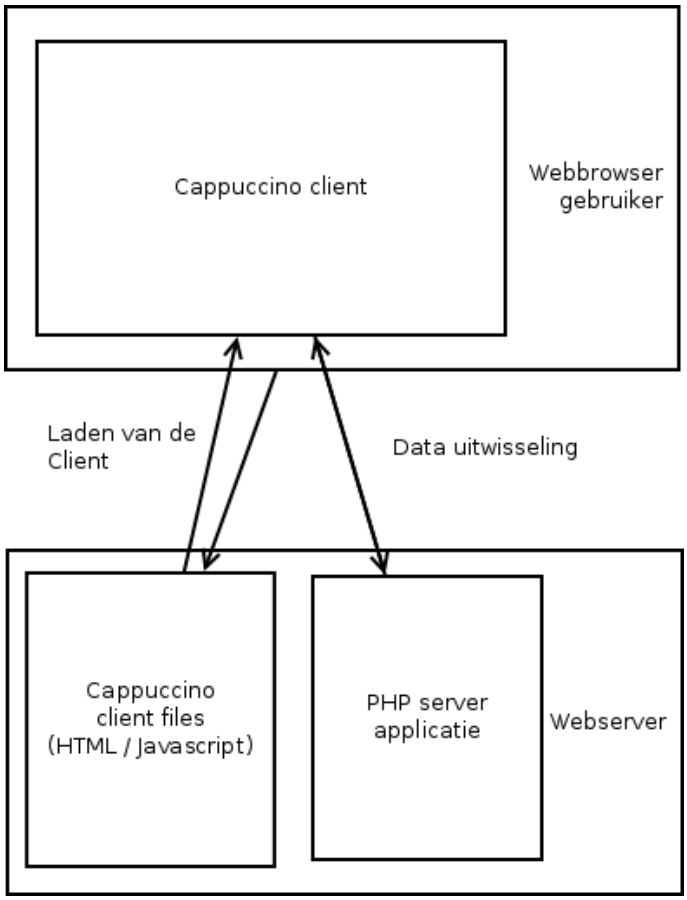
Figuur 8.2 geeft schematisch de opbouw van een fictief document weer. Als we het hebben over een type document, hebben we het eigenlijk over een specifieke combinatie van primitieve types.

Daarnaast hebben we nog het concept *DocumentHandle*. Om te zorgen dat de *client* niet veel data opvraagt bij de *server*, is gekozen om niet continue hele documenten over te sturen. Het is onnodig, als in het navigatiescherm alle documenten ook daadwerkelijk volledige geladen zouden worden, terwijl 90% van de getoonde documenten niet geopend wordt in die sessie. Daarom hebben we het concept *DocumentHandle* geïntroduceerd. Een *DocumentHandle* is een referentie naar een document. Deze bevat informatie voor het ophalen van het document plus een label dat getoond wordt in overzichten. Als de inhoud van het document opgevraagd wordt, wordt deze door de *DocumentHandle* geladen. Het laden (of opslaan, verwijderen, etc.) gebeurd asynchroon. Dit wil zeggen dat de applicatie niet wacht tot de actie is voltooid. Deze actie vereist communicatie met de *server* en kost verhoudingsgewijs veel tijd. Het is immers niet wenselijk dat

de applicatie meerdere seconden niet reageert tijdens het opslaan van een document. Om dit te voorkomen voert de *DocumentHandle* de acties op de achtergrond uit en genereert deze notificaties als er een actie voltooid is. Applicatie onderdelen kunnen zich registreren op deze notificaties en worden dan op de hoogte gesteld als het document geladen, opgeslagen of verwijderd is. Op deze wijze blijft de applicatie altijd reageren op de gebruiker en wordt deze ook in staat gesteld om meerdere taken tegelijk uit te voeren.

8.2 De globale architectuur

Cappuccino dwingt ons alle HTML manipulaties in Javascript en Objective-J uit te voeren. We bouwen eigenlijk een losstaande *client* die van de *server* alleen de data opvraagt die het nodig heeft. Alle visuele aanpassingen worden vanuit Cappuccino geregeld. In figuur 8.3 is te zien hoe de applicatie op het hoogste niveau in elkaar steekt.



Figuur 8.3: Globale architectuur van het CMS

Als de bezoeker de CMS applicatie opvraagt wordt eerst alle HTML en Javascript voor de *client* geladen. Er is hier nog geen sprake van interactie met het PHP *server* component. Nadat de *webbrowser* alle HTML en Javascript-code heeft geladen, wordt de applicatie gestart en begint de *client* applicatie data op te vragen van het *server* component. Hierbij kan gedacht worden aan het uitwisselen van gebruikersnaam en wachtwoord of het opvragen van documenten.

8.2.1 De communicatie

Zoals uit het eerder genoemde al blijkt, is de *client* afhankelijk van de *server* voor data. Deze zal dan ook veel communiceren met de *server*. Een nadere kijk op hoe de communicatie in zijn werk gaat, is wel op zijn plaats.

Alle communicatie geschiedt in de vorm van JSON berichten. JSON staat voor "Javascript Object Notation" en is een serializatieformaat voor Javascript. Fijn is dat PHP standaard ondersteuning heeft voor het converteren van PHP datatypes naar JSON berichten. Fijner nog is dat het Zend Framework een *server* component bevat dat de conversie van JSON naar PHP automatiseert op basis van de JSON-RPC[1] standaard.

```
{
    "method": "hello.echo",
    "params": ["Hello JSON-RPC"],
    "id": 1
}
```

Figuur 8.4: Een JSON-RPC *request*

In figuur 8.4 staat een voorbeeld van een JSON-RPC *request*. Deze bestaat uit drie stukken: Ten eerste de *method* die weer bestaat uit een *namespace* (hello) en de daadwerkelijke *method* (echo). De *namespace* mapt op de *server* direct naar een *Controller* klasse. De *method* mapt direct naar een methode in die *Controller* klassen. Deze mapping wordt volledig gedaan door het Zend Framework. Ten tweede de *params* zijn de parameters die aan de methode meegegeven worden. Ten derde de *id*. De *id* kan door de ontwikkelaar gebruikt worden om de *requests* uit elkaar te houden. Dit is handig wanneer je vanuit de *client* meerdere *requests* naar dezelfde *namespace* en *method* stuurt. Zo kun je de juiste *response* bij de juiste *request* te vinden. De *id* van de *request* wordt op de *server* ook weer meegegeven aan de *response*.

```
{
    "result": "Hello JSON-RPC",
    "error": null,
    "id": 1
}
```

Figuur 8.5: Een JSON-RPC *response*

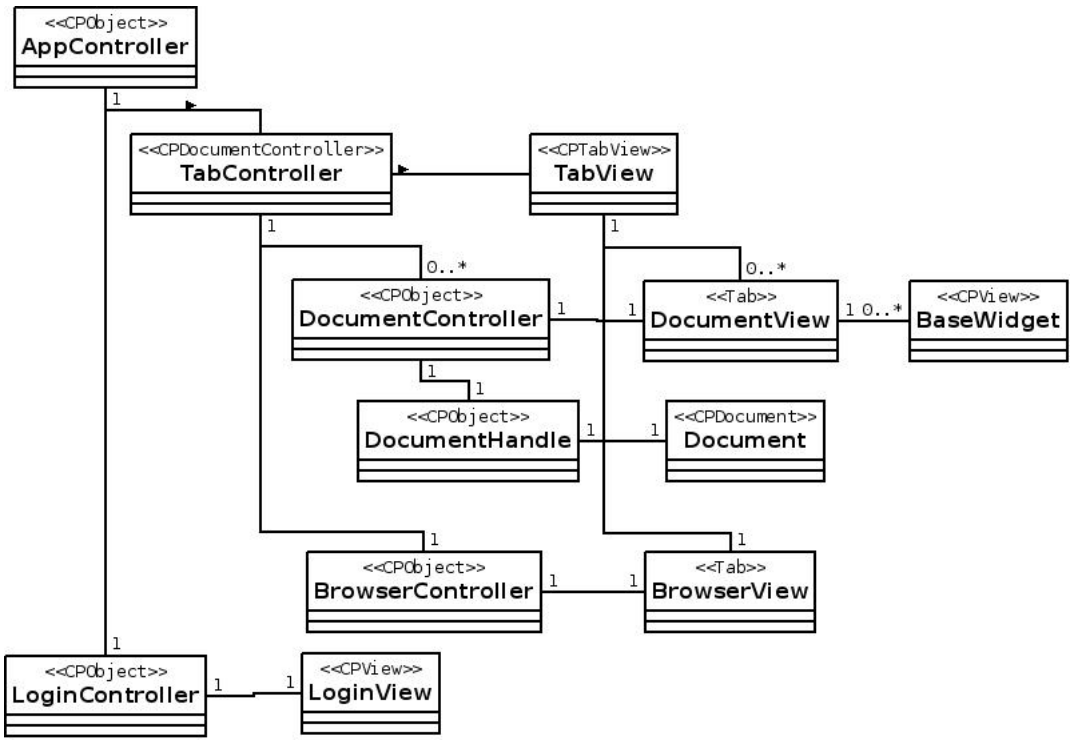
In figuur 8.5 is een *response* te zien. Deze heeft een vergelijkbare opbouw als de *request*. De result bevat het resultaat van de aangeroepen functie. De *error* bevat eventueel een *error code* als er iets fout gegaan is. De *id* is gelijk aan die in de *request*.

Op de *client* moet iets meer werk verricht worden voor het vertalen van de berichten. Cappuccino heeft faciliteiten voor het vertalen van en naar JSON, maar men moet zelf code schrijven voor het opmaken van *request* objecten en het analyseren van de *response* objecten in JSON-RPC formaat.

8.2.2 Globale architectuur van de *client*

Ik zal aan de hand van een klassediagram uitleggen hoe de structuur van de *client* er uitziet.

[1] `http://json-rpc.org/`



Figuur 8.6: Versimpelde klassediagram van de *client*

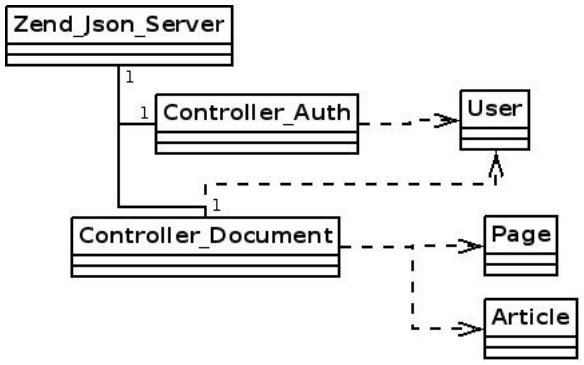
In figuur 8.6 is een klassediagram van de *client* te zien. De *View* en *Controller* klassen vallen hier meteen op, de namen eindigen respectievelijk op *View* en *Controller*. De applicatie heeft op het moment van schrijven twee *Model* klassen: *DocumentHandle* en *Document*. Deze komen uitgebreid aan bod in paragraaf 8.3. Voor nu is het voldoende om te weten dat hun verantwoordelijkheid het laden en opslaan van documenten is.

De AppController is het centrale startpunt van de applicatie. Het enige wat deze *Controller* doet is de LoginController instantiëren als de gebruiker nog niet ingelogd is of TabController instantiëren wanneer dat wel het geval is. De AppController geeft zelf niets weer en heeft daarom geen view. De LoginController en LoginView verzorgen het inloggen van de gebruiker. De TabController en TabView bevatten het grootste deel van de applicatie. De TabController is verantwoordelijk voor het aanmaken van nieuwe Controllers en de TabView toont de bijbehorende *Views* bij deze Controllers in een tab. Op dit moment beheert de TabController twee soorten klassen. De BrowserController, samen met de BrowserView, verzorgt een tab waarmee de inhoud van de website genavigeerd kan worden en de DocumentControllers, en DocumentViews, voor het bewerken van documenten. De BrowserController communiceert niet zelf met DocumentControllers, maar doet dit via de TabController.

Interessant is het duidelijke verschil in gebruik van *MVC* tussen traditionele webapplicaties (*stateless*) en *statefull* applicaties. In een traditionele webapplicatie wordt bij het binnenkomen van de *request* op de *server* één *Controller* opgestart die de *request* afhandelt. Het verloop door de *Controller* en dus de hele applicatie is dan erg lineair. Bij een *statefull* applicatie is dit anders. Er zijn meerdere *Controllers* die allemaal tegelijk draaien. Ze hebben allemaal hun eigen verantwoordelijkheid, maar moeten vaak ook communiceren. Dit brengt veel scopeproblemen met zich mee aangezien niet elke *Controller* zomaar toegang heeft tot elke andere *Controller*. Dit maakt het bouwen van *statefull* applicaties dan ook een stuk lastiger dan van *stateless* webapplicaties.

8.2.3 Globale architectuur van de *server*

In figuur 6.3 op pagina 18 is kort de globale *server* architectuur getoond. Hieronder zal ik kort het klassediagram toelichten. Dit geeft in wat meer detail weer hoe de *server* applicatie werkt.



Figuur 8.7: Versimpeld klasse diagram van de *server*

De *server* is, zoals in figuur 8.7 te zien is, een stuk simpeler dan de *client*. De *server* omvat dan ook veel minder logica en hoeft zich niet bezig te houden met de presentatie van de data.

Grofweg functioneert de *server* als volgt: Er is één instantie van Zend_Json_Server. Deze heeft een aantal geregistreerde *Controller* klassen, in dit geval twee. Zend_Json_Server inspecteert de *request* van de *client* en bepaalt welke klasse het moet instantieëren en welke methode het op die klasse moet aanroepen. Zend_Json_Server, Controller_Auth en Controller_Document zijn de *Controller* klassen. User, Page en Article zijn *Model* klassen. Deze *Models* gebruiken Doctrine om met de database te praten. Ze worden gebruikt door de *Controllers* om gegevens op te halen, of weg te schrijven naar de database. Eventueel bevatten ze logica die specifiek is voor de data.

8.3 De document architectuur

Een doelstelling is om de *client* zo generiek mogelijk te houden. Er wordt bedoeld dat de *client* met elk soort document overweg kan zonder aanpassingen in de code, zolang de *client* de primitieve types allemaal ondersteunt. Alleen de *server* hoeft dan naar de wens van de klant ingericht te worden. De *client* past zich automatisch aan aan de *server* instellingen.

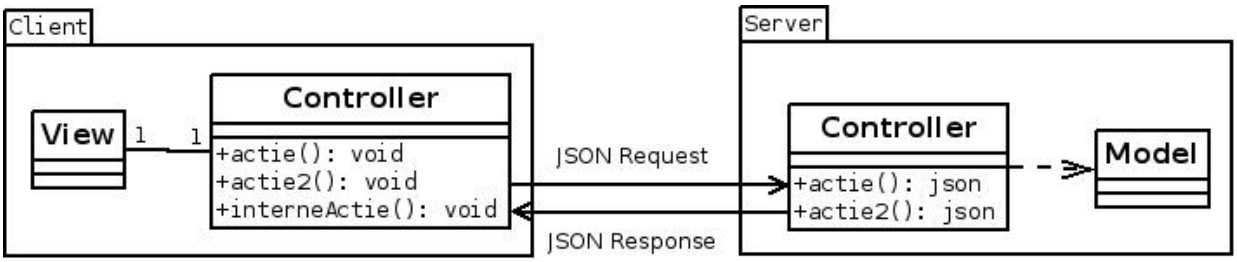
Om dit doel te bereiken, zal de *server* extra meta-data moeten meesturen met elke *request*. De *client* gebruikt deze meta-data om te bepalen welke Objective-J klasse gebruikt moeten worden voor de weergave. Deze Objective-J klassen noemen we Widgets. Deze bevatten de nodige code voor het tonen van de data aan de eindgebruiker. Een document op de *client* is dan niets meer dan een verzameling Widgets die gezamenlijk het hele document tonen. Per primitief datatype is er één Widget op de *client*. Het aanmaken en beheren van de Widgets per document wordt geregeld door de DocumentController. Elk geopend document heeft een eigen instantie van een DocumentController en een eigen instantie van een DocumentView. De *controller* instantie regelt alle acties en het laden van de Widgets, de *view* instantie regelt de weergave en layout van de Widgets.

Op de *server* wordt de meta-data gebruikt om te bepalen waar en hoe de data moet worden opgeslagen. De meta-data wordt op de *server* gespecificeerd in een XML bestand. In

figuur 7.1.2 op pagina 22 is te zien hoe een XML specificatie er uitziet. In de specificatie is te zien dat elk type document een naam heeft. Deze naam wordt gebruikt door de Controller_Document om te bepalen welke *model* klasse hij moet laden voor dat soort document. Op deze manier kunnen we door nieuwe Doctrine *model* klassen te genereren en de XML specificatie bij te werken, de Controller_Document en dus de *client*, met nieuwe soorten documenten laten werken. Het is zelfs mogelijk om Doctrine *models* te genereren op basis van de XML specificatie. Het gehele CMS kan snel aangepast worden aan de wensen van de klant.

8.4 De overige modules

Het CMS bevat uiteraard meer modules dan de documenten module. Echter, deze hebben niet de dynamische aard die de documentenmodule heeft en zijn ze daarom een stuk eenvoudiger.



Figuur 8.8: Voorbeeld opbouw van een simpele module

Alle modules zijn variaties op één thema, zoals te zien is in figuur 8.8. Er is een *controller* en een *view* op de *client*. Deze biedt functionaliteit aan. Sommige acties hebben communicatie met de *server* nodig. Op de *server* is een corresponderende *controller* waarmee de *client controller* communiceert. Dit gebeurt in JSON. De *server controller* communiceert, waar nodig, met vooraf gedefinieerde *model* klassen voor het verwerken van data en stuurt een *response* terug naar de *client controller* met het resultaat.

Doordat de overige modules geen dynamische opbouw hebben zoals de documentmodule, hebben ze geen configuratie opties. Sommige zullen in de toekomst wel configuratieopties krijgen, maar er zal geen code geïnjecteerd kunnen worden zoals de gegenereerde document modellen in de documentmodule.

Hoofdstuk 9

De realisatie

Na het schrijven van het plan van aanpak, was alle nodige informatie er om met het project aan de slag te kunnen. Ik had de SCRUM stories opgesteld en mijn keuze voor de technologie gemaakt. Met als gegeven heb ik een overleg gehad met mijn externe begeleider. Ik heb mijn planning voor de eerste twee sprints besproken en de SCRUM stories die ik bedacht had. Na goedkeuring ben ik aan de slag gegaan.

9.1 De eerste sprint, de weg naar een werkende basis

Elke sprint had als einddoel een werkend eindproduct op te leveren. Dit is een fijne basis om de volgende sprint mee te starten. Dit geldt natuurlijk voor alle sprints behalve de eerste. Hier begin je met niets. Veel van de technologie was mij nog onbekend en het was dan ook de vraag of de schattingen in de planning realistisch waren. Gelukkig bleek dit het geval. Het lukte een werkend systeem te maken met één documenttype en een simpele lijstnavigatie. Na de opluchting dat mijn schattingen haalbaar bleken, vielen me toch een aantal dingen op. Er ging verhoudingsgewijs veel tijd zitten in de ontwikkeling van de *client* en weinig in de *server*. Bij de *server* bleken de frameworks krachtig en hoefde ik minder code zelf te schrijven dan geanticipeerd. Het ontwikkelen van de *client* bleek tegen te vallen. Er zaten meer bugs in Cappuccino dan verwacht, en met de beperkte tools beschikbaar was het moeilijk de problemen op te sporen. Soms was ik een dag bezig met uitzoeken waarom iets niet werkte, maar geen foutmelding was. Gelukkig middelden deze tegenvaller en meevaller elkaar uit, en kreeg ik mijn sprint op tijd af.

9.2 Hobbels op de weg en wijzigingen in de koers

Na ook succesvol de tweede sprint te hebben afgerond, heb ik nog eens kritisch met mijn begeleiders naar de geplande SCRUM stories gekeken. Alle SCRUM stories stonden op de backlog gerangschikt naar prioriteit. De prioriteit was grofweg: eerst alles wat een CMS nodig heeft en daarna alle leuke extra´s die ik met het Cappuccino framework kan bouwen. Dat leek een logische volgorde, maar zou betekenen dat het product pas zou gaan uitblinken in de laatste twee sprints. Ook zou dit betekenen dat als de tussenliggende sprints zouden uitlopen, dat het product dus nooit zou uitblinken. Dit zou de verkoopbaarheid niet ten goede komen. Om deze reden is besloten dat een aantal belangrijke, maar niet strikt noodzakelijk SCRUM stories naar achteren geplaatst werden. Een aantal aantrekkelijke extra's werden naar voren geschoven. Dit resulteerde vooral in dat de Media Library en HTML Widget naar voren werden gehaald. Het rechtensysteem (ACL) werd naar achteren gezet.

Na deze wijzigingen ben ik begonnen aan de derde sprint. De scriptie bleek op dat moment meer tijd in beslag te nemen dan gepland, ondanks dat ik er in sprint twee al aan begonnen was. Daarnaast was ik een aantal dagen ziek. Een poging tot inbraak in ons kantoor, zorgde voor de nodige oponthoud. Het resulteerde er in dat ik een groot deel van de resterende tijd aan de scriptie werkte. Hierdoor was de functionaliteit niet goed afgewerkt aan het einde van sprint drie. Het eindproduct werkte, maar er zaten nog bugs in en maakte een onafgewerkte indruk.

Aan het einde van sprint drie heb ik met mijn externe begeleider de planning voor sprint vier herzien en de grote story (de Media Library) naar sprint vijf geschoven. Ik had de verwachting dat het voorbereiden van mijn afstuderen in januari ook de nodig tijdsinvestering met zich mee zou brengen. Een aantal stories zou waarschijnlijk niet geïmplementeerd zullen worden.

9.3 Kwaliteitsbewaking & Testen

Het geautomatiseerd testen van een Cappuccino-applicatie bleek lastig. Traditionele tools zoals Selenium[1] zijn niet bruikbaar, aangezien daarvoor speciale tags in de HTML gezet moeten worden om Selenium te helpen bij het herkennen van bepaalde elementen. Dit vond ik jammer. Ik had graag een geautomatiseerd test-platform willen opzetten aan de hand van een soortgelijke tool.

Wel is er OJunit, een Unit-testing framework voor Objective-J. Helaas is Unit-testen niet bruikbaar voor user interface klassen. Om deze redenen is besloten tijdens deze afstudeeropdracht geen tijd te besteden aan het opzetten van een testomgeving met OJunit.

Zeker is wel dat hier in de toekomst tijd aan besteed gaat worden, wanneer de tools er beschikbaar voor zijn. Op het moment van dit schrijven zijn er prototype-tools met Selenium-achtige functionaliteit. Samen met OJunit zal het dan mogelijk worden deze tests alsnog te schrijven.

De applicatie is gewoon met de hand getest. Aan het eind van elke sprint werden alle functionaliteiten langsgelopen om te controlleren of deze correct werkten. Daarnaast is er gewerkt aan de beheersbaarheid van het product. Het eerder genoemde versiebeheer is hier een voorbeeld van. Door de geschiedenis van de code goed bij te houden, en voor het aanmaken van elke revisie te controlleren wat je precies veranderd hebt, voorkom je onbedoelde wijzigingen. Daarnaast is documentatie erg belangrijk. Door zowel in de code als daarbuiten documentatie beschikbaar te stellen, wordt bijgedragen aan het begrijpbaar maken van de code. Hoe beter de code te begrijpen is, hoe minder fouten er gemaakt zullen worden.

9.4 Het eindresultaat

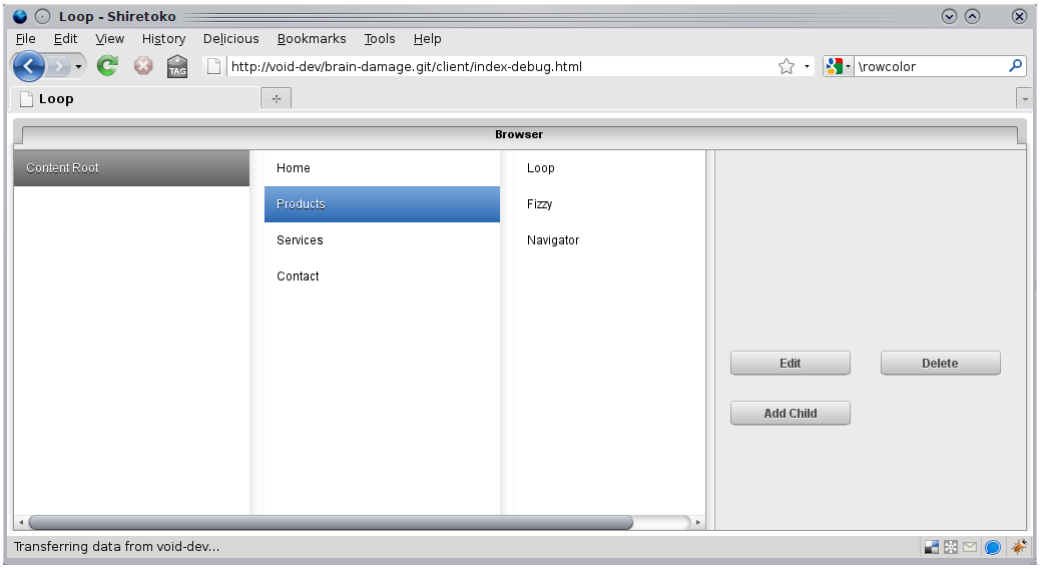
Zoals in paragraaf 9.2 te lezen is, ging niet alles zoals gepland. De geplande hoeveelheid functionaliteit was erg ambiteus. Toch ben ik ver gekomen. In deze paragraaf wil ik de stand van zaken laten zien, ten tijde van dit schrijven.

Alle stories met betrekking tot de navigatie zijn succesvol afgerond. Zoals in figuur 9.1 te zien is, hebben we een mooi navigatiescherm. Via dit scherm kan alle content gevonden worden. Ook is dit scherm direct te raadplegen door de eindgebruiker, het is altijd aanwezig als eerste tab.

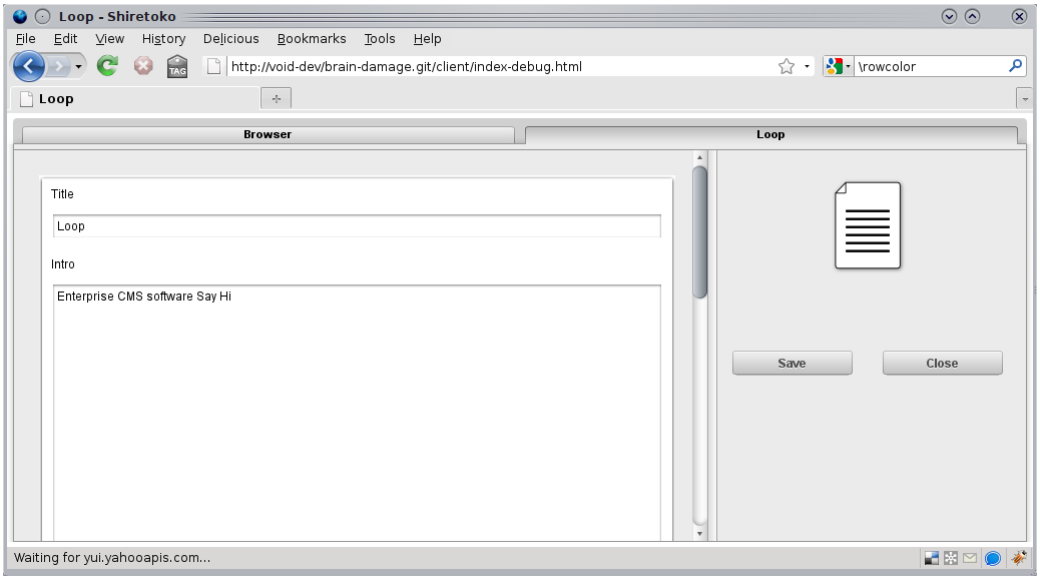
Alle architectuur is klaar om Widgets te implementeren. Figuur 9.2 toont een open document. In het scherm zijn een aantal Widgets (de velden) te zien. Bewerking op het document worden direct met de navigatie gesynchroniseerd. De informatie in het navigatiescherm is altijd actueel. Ten tijden van dit schrijven is de implementatie van Widgets nog niet voltooid.

Om een compleet beeld te geven van de huidige situatie, staat in 9.3 een overzicht van

[1] Een vergebruikte user interface test applicatie speciaal voor webapplicaties



Figuur 9.1: Het navigatiescherm



Figuur 9.2: Het documentscherm

alle stories en hun huidige status. De groene stories zijn voltooid en functioneren goed. Alle gele stories staan in de planning voor de komende weken. Hoe meer de kleur naar geel neigt, hoe groter de kans dat deze story afgerond wordt binnen dit project. Deze inschatting is gemaakt op basis van prioriteiten en huidige status. De rode stories gaan vrijwel zeker niet geïmplementeerd worden.

Story	Status
Edit (Basic) content	Gereed
Basic Navigation	Gereed
Tree navigation	Gereed
XML content specification	Gereed
Relations (Widget)	Gereed
HTML Widget	Gereed
Authentication	Gepland
Media library (Images support)	Gepland
Date Widget	Gepland
Google maps support/Widget	Gepland
Video support (youtube/viddix/?)	Gepland
Modularity	Gepland
Number Widget (Spinnerbox)	Gepland
Code generation	Gepland
Access control lists (Authorization)	Niet gereed
Image manipulation	Niet gereed
Status page	Niet gereed
Searching	Niet gereed

Figuur 9.3: Status van alle SCRUM stories op moment van schrijven

Hoofdstuk 10

Conclusie

Uit het eindresultaat blijkt dat Cappuccino een veelbelovende technologie is. In combinatie met de nieuwe innovaties die HTML 5 ons nog gaat brengen, zoals *drag and drop* en *client side storage*, zullen de grenzen tussen *desktop* en het web verder vervagen. Een concreet voorbeeld hiervan is het direct van de *desktop* slepen van een plaatje naar de CMS applicatie op het web. Het verschil in interactie tussen de *desktop* en het web zal minimaal worden.

Tijdens de stage is mijn vermoeden bevestigd dat het noodzakelijk is om een applicatie volledig in Javascript te bouwen, wanneer je als doel hebt de gebruikerservaring van de *desktop* te evenaren. *Desktop* applicaties kunnen veel taken tegelijk uitvoeren. Denk aan chatten met meerdere gebruikers, spellingscontrolle terwijl je typt, notificaties van dingen die gebeuren op de achtergrond, en zo verder. Om dit te kunnen bereiken is het noodzakelijk om de *client* applicatie *state-full* te programmeren.

Daarentegen is uit het project gebleken dat we nog een lange weg te gaan hebben. HTML 5 is nog geen standaard en ook nog niet af. Cappuccino bevat nog erg veel bugs en mist nog veel tools. Vanuit een bedrijfsmatig oogpunt kwam deze afstudeerperiode nog erg vroeg om met deze jonge technologieën aan de slag te gaan.

Vanuit een educatief oogpunt was dit juist een heel geschikte tijd. Door de bugs en missende functionaliteit ben ik gedwongen mij te verdiepen in Cappuccino. Hierdoor heb ik niet alleen met Cappuccino leren werken, maar snap ik ook hóe het werkt. Dit is erg nuttig voor mij als programmeur, maar ook voor VOIDWALKERS.

Naast de opgedane kennis is er ook een werkend product. Dit product kan in eerste instantie dienen als demonstratie van kennis en vaardigheden. VOIDWALKERS kan dan aantonen wat ze in huis heeft. Anderzijds kan VOIDWALKERS het product verder uitwerken en productie gereed maken. Daarnaast is er nog de kennis van Cappuccino. Deze maakt het mogelijk om in de toekomst nieuwe projecten met Cappuccino uit te voeren. Deze redenen samen moeten het voor VOIDWALKERS gemakkelijker maken door te groeien naar een hoger marktsegment.

Hoofdstuk 11

Evaluatie

Zoals in de conclusie te lezen was, was dit project een erg leerzame uitdaging. Cappuccino en Objective-J zijn vernieuwende ontwikkelingen. Ik ben blij om te zien dat er steeds meer geavanceerde frameworks komen voor het ontwikkelen van rijke webapplicaties. Het web voelde altijd beperkt omda je niet met hetzelfde (hoge) abstractieniveau als met desktopapplicaties kon ontwikkelen. De frameworks en API's van desktopapplicaties lijken nu naar het web te komen.

Het uitvoeren van de opdracht binnen mijn eigen bedrijf is me goed bevallen. Ondanks het continue 'multi-tasken' tussen bedrijf en afstuderen, denk ik niet dat mijn afstudeeropdracht daaronder geleden heeft. Ik denk zelfs dat het mijn afstudeeropdracht goed heeft gedaan. Doordat het product ten goede zou komen aan mijn eigen bedrijf was de motivatie erg hoog. Andere activiteiten in het bedrijf zorgden voor afwisseling. Die afwisseling voorkwam dat ik verveeld raakte en maakte dat ik uit keek naar de momenten waarop ik weer verder kon aan mijn stage opdracht. Die momenten waren dan ook extra productief.

Ik denk dat deze motivatie blijkt uit het opgeleverde product. Ik ben dan ook erg trots op het CMS zoals het nu is.

Een aantal concrete dingen heb ik geleerd. Bij aanvang had ik weinig kennis van Javascript. Door met Cappuccino te werken en de broncode te bestuderen, heb ik veel geleerd over Javascript en de manier waarop Javascript met de DOM overweg gaat. Tijdens dit project heb ik mijn eerste ervaringen opgedaan met JSON en Doctrine. Ook heb ik de nodige ervaring opgedaan met plannen en SCRUM. Tijdens dit project had ik veel vrijheid in vergelijking tot voorgaande projecten. Dit was even wennen. SCRUM is in een éénmansproject een beetje raar werken, want SCRUM is bedoeld voor teams. Wel heb ik door SCRUM in te zetten een beter idee gekregen van wanneer ik SCRUM in de toekomst zou willen toepassen en hoe.

Uiteindelijk kijk ik dan ook met een goed gevoel terug op het afstuderen. Ik heb erg veel geleerd, maar vooral ook een semester lang aan een heel erg leuk en uitdagend project kunnen werken.

Bijlagen

- Plan van aanpak
- SCRUM stories

Plan van aanpak stage Jeroen Tietema

Jeroen Tietema (1524740)

6 oktober 2009

Inhoudsopgave

1 Inleiding

Het internet heeft in zijn korte bestaan alle verwachtingen al overtroffen. Dagelijks worden er nieuwe ideeën bedacht en technologieën ontwikkeld die de mogelijkheden van het internet nog verder verruimen. Op dit moment staan we aan de vooravond van de HTML 5 [1] standaard. HTML 5 zal een aantal technieken beschikbaar stellen waarmee een geheel nieuw soort applicaties ontwikkeld kan worden. Gmail, Google Calendar en Google Docs zijn voorbodes van de RIA´s (Rich Internet Applicaties). RIA´s die in staat zijn traditionele desktop applicaties te vervangen.

Tijdens mijn stage wil ik de mogelijkheden verkennen van het ontwikkelen van een product dat van deze technologieën gebruik maakt. Tevens wil ik deze stage uitvoeren in opdracht van mijn eigen bedrijf zodat het product na mijn stage kan worden verder ontwikkeld.

In dit document zal ik toelichten hoe ik precies invulling wil geven aan mijn stage.

2 Achtergronden

Voidwalkers is een bedrijf dat in 2007 is opgericht door Mattijs Hoitink en Jeroen Tietema. Op dat moment vooral bedoeld om freelance bij te verdienen, maar inmiddels heeft Voidwalkers al een aantal grote opdrachten achter de rug. De klantenkring bestaat vooral uit ontwerpbureaus waarvoor Voidwalkers de technische implementatie van websites realiseert. Via die ontwerpbureaus werkt Voidwalkers onder andere aan websites van VNU en diverse overheidsinstellingen.

Door de informatica achtergrond van de oprichters en hun certificering in PHP blinkt Voidwalkers gemakkelijk uit in dit marktsegment. Klanten herkennen dit snel in ons en hierdoor doen klanten vooral een beroep op ons als het gaat om de wat complexere maatwerk projecten. Hierbij kan gedacht worden aan onder andere: het bouwen van een webservice, een bestaande website koppelen aan een webservice, de architectuur van bestaande applicaties aanpassen als deze geen plaats biedt voor nieuwe functionaliteit en advies geven aan de ontwikkelaars van klanten op het gebied van architectuur.

Voidwalkers wil zich graag nog meer profieleren in de markt als een technische specialist. Om dit te bereiken wil Voidwalkers een CMS pakket ontwikkelen dat haar klanten in staat stelt beter en gemakkelijker software projecten te realiseren. Door dit pakket opensource beschikbaar te stellen hoopt Voidwalkers snel een hoop ontwikkelaars te bereiken (en dus ook een hoop potentiële klanten).

3 Probleemstelling

Groeien betekent niet alleen meer klanten en meer opdrachten, maar ook groei in omvang van de opdrachten. Voidwalkers merkt dat dit niet kan zonder eigen producten. Een veelvuldige vraag van klanten is of Voidwalkers een eigen CMS (Content Management Systeem) heeft. Het is gebleken dat klanten grotere opdrachten eerder uitbesteden aan bedrijven die al eigen producten hebben. Dit komt deels omdat klanten niet steeds voor bergen extra ontwikkeltijd willen betalen om alles vanaf niets op te bouwen. Ook zijn de risico´s minder en kleiner als er al met werkende software wordt gestart.

Kortom: *Voidwalkers heeft behoefte aan een eigen CMS om aan de verwach-*

[1] `http://dev.w3.org/html5/spec/Overview.html`

tingen van klanten te voldoen.

Uiteraard zou de gemakkelijkste oplossing zijn één van de opensource CMS paketten te gebruiken die op het internet beschikbaar zijn. Echter zijn er een aantal redenen waarom VOIDWALKERS denkt dat het zelf ontwikkelen van een CMS de moeite waard is.

Ten eerste leggen veel van de huidige CMS´en op welke technologieën je moet gebruiken om de uiteindelijke website te bouwen. Vaak moet dit in een template taal/systeem van het CMS. De CMS´en zijn er niet op gemaakt dat je een eigen implementatie methode gebruikt. VOIDWALKERS wil een CMS bouwen waarin de ontwikkelaar zelf kan kiezen welke talen en frameworks hij gebruikt om de website te bouwen. Hierdoor wordt het gemakkelijker voor organisaties om het CMS te gebruiken aangezien ontwikkelaars hun eigen technologië en kennis kunnen blijven gebruiken voor het bouwen van de uiteindelijke website.

Ten tweede zijn de meeste opensource projecten al erg oud. Hierdoor is er in de architectuur geen rekening gehouden met hedendaagse wensen als het gebruik van veel Javascript. Voorbeelden hiervan zijn Joomla en Drupal. Beide CMS producten zijn nog pure PHP 4. De architectuur bestaat voornamelijk uit een hele berg functies en een template taal. Een Javascript intensieve interface maken zou betekenen dat er in het template Javascript gestopt moet worden. Dit is misplaatst doordat Javascript ook gebruikt wordt voor applicatie logica en het dan wenselijk is een soort MVC achitectuur op te zetten binnen de Javascript code. Deze architectuur past niet binnen de bestaande architectuur van die systemen aangezien hier helemaal nooit rekening mee is gehouden. Dit maakt deze systemen alleen geschikt voor platte HTML met hooguit een beetje Javascript. VOIDWALKERS wil een CMS dat een meer dynamisch karakter heeft dan een verzameling statische HTML pagina´s. Dit vereist een andere aanpak dan die genomen is bij de op dit moment beschikbare CMS projecten.

4 De opdracht

4.1 In het kort

De opdracht zal bestaan uit het ontwerpen van een CMS en het bouwen van een werkend prototype.

4.2 Omschrijving

Het CMS zal bestaan uit een client- en een servercomponent. De client zal volledig in de browser draaien. Hiermee wordt bedoeld dat de client uit 1 enkele HTML pagina bestaat en dat alle functionaliteit in Javascript is geïmplementeerd. Hierdoor hoeft de client alleen voor de data met de server te communiceren. De user-interface van de client kan hierdoor veel sneller reageren dan bij traditionele webapplicaties. Om dit mogelijk te maken wil VOIDWALKERS gebruik maken van nieuwe technieken die beschreven worden in de HTML 5 specificatie. Het servercomponent zal een webservice zijn geschreven in PHP.

Activiteiten van de stagiair zullen zijn het ontwerpen en bouwen van zowel de server en de client.

4.3 De te gebruiken technieken en ontwikkelomgeving

Voor het ontwikkelen van de client zullen de volgende technieken gebruikt worden:

- Javascript
- Objective-J (een taal bovenop Javascript)[2]
- Het Cappuccino framework[3]
- HTML (5)[4]
- JSON & XML

Voor het ontwikkelen van de server zullen de volgende technieken gebruikt worden:

- PHP[5]
- MySQL[6]
- Doctrine (ORM pakket voor PHP & MySQL)[7]

Objective-J is een uitbreiding op Javascript net als dat Objective-C een uitbreiding op C is. Objective-J is dan ook gemodeleerd naar Objective-C. Dit is nog volledige onbekend voor de stagair en daar valt dus veel te leren.

Het Cappuccino framework is eigenlijk het Cocoa framework (bekend van Mac OS X en de iPhone) voor het web. Het Cocoa framework (en Cappuccino) is een GUI componenten framework net als Swing in Java. De stijl van applicaties programmeren lijkt dan ook erg op het ontwikkelen van Swing applicaties in Java. Cappuccino stelt de ontwikkelaar in staat om echte applicaties voor het web te bouwen. *280Slides* (280slides.com) is een voorbeeld van zo´n applicatie. Het is een volledige presentatie applicatie zoals Keynote of Powerpoint in de browser.

Zowel Objective-J en Cappuccino zijn betrekkelijk jong (ongeveer 1 jaar).

5 De projectgrenzen

De primaire beperking is tijd. Hieronder probeer zal er wat duidelijkheid geschappen worden over wat wel en wat niet bij het project zal horen.

Omdat Objective-J een erg nieuwe taal is wordt deze nog niet door veel tools ondersteund. Dit is lastig want het legt erg veel beperkingen op welke tools gebruikt kunnen worden. Bijvoorbeeld de syntax van Objective-J wordt nog maar door 2 editors goed herkend op Linux. Echter is het de bedoeling zo weinig mogelijk tijd kwijt te zijn met bijzaken als het een tool geschikt maken voor Objective-J. Dit zal alleen gedaan worden als het een aanzienlijke tijdswinst op gaat leveren voor de rest van het project. In

[2] `http://objective-j.org/`
[3] `http://cappuccino.org`
[4] `http://dev.w3.org/html5/spec/Overview.html`
[5] `http://www.php.net`
[6] `http://www.mysql.com`
[7] `http://www.doctrine-project.org`

alle andere gevallen zal er gewerkt moeten worden met het beperkte assortiment wat beschikbaar is.

Ook is Cappuccino, het framework wat geschreven is met Objective-J, nog lang niet af. Er moet voor gewaakt worden dat er niet meer tijd gestoken wordt in Cappuccino ontwikkeling dan in het daadwerkelijke project. Uiteraard is het goed om code die ook nuttig kan zijn voor het Cappuccino project te doneren, maar het is niet de bedoeling code te ontwikkelen die niet direct nodig is voor de bouw van het CMS.

Veel van onze klanten zijn erg benieuwd naar onze aanpak voor het bouwen van dit CMS. Het zou kunnen dat het CMS in een vroeg stadium al nuttig kan zijn voor klanten. Toch behoren (vroegtijdige) implementaties van het CMS niet tot dit project. Hooguit zal hier een advieserende positie in genomen worden en zal het uitvoerende werk vanuit tijdsoverwegingen niet door de stagair gebeuren.

6 De producten

Hieronder volgt een opsomming van alle producten die tijdens het project opgeleverd zullen worden.

6.1 Plan van aanpak

Dit document.

6.2 Functioneel ontwerp

Dit document zal bestaan uit een aantal stories met schermontwerpen. Stories zijn korte omschrijvingen van functionaliteit en zijn een methodiek van SCRUM. De stories uit dit document zullen gebruikt worden voor het opstellen van de planning. Deze stories zijn terug te vinden op de project wiki waar ze gaande weg het project verder uitgewerkt zullen worden.

6.3 Technisch ontwerp

Voor alle nieuwe functionaliteit zal eerst een een technisch ontwerp gemaakt worden alvorens er tot implementatie overgegaan zal worden. Dit technisch ontwerp is terug te vinden op de project wiki.

6.4 Werkend CMS prototype

De bedoeling van het CMS prototype is dat het aan het eind van de stage de juiste architectuur en technologiëen bevat. Het is niet de per definitie de bedoeling dat er een productie klaar CMS klaarstaat aan het eind van het project. Prioriteit ligt bij dit project dat het technisch deugd en niet dat alle functies erin zitten die voor een productie omgeving vereist zijn.

6.5 Documentatie

In eerste instantie zal de documentatie vooral bestaan uit een functioneel- en technisch-ontwerp wat tijdens de loop van het project continue wordt bijgewerkt. Eventueel als

de ontwikkeling van het CMS voorspoedig gaat kunnen hieraan kortere tutorials worden toegevoegd die derden in staat stellen gemakkelijk met de software aan de slag te gaan.

6.6 Scriptie

De scriptie is een noodzakelijk onderdeel van het afstuderen. Deze zal tijdens een groot deel van de stage parallel aan het ontwikkelen geschreven worden.

7 Projectorganisatie

De stagair: Jeroen Tietema
De docentbegeleider: Jan Mooij
De inhoudelijke begeleider: Ronald van Zuijlen & Mattijs Hoitink

Na overleg met Jan Mooij is er besloten zowel Ronald als Mattijs aan te dragen als begeleiders.

Ronald van Zuijlen zal begeleiding en advies geven over de ontwikkeling van het CMS en feedback geven over de algehele koers van het project. Ronald is CMS consultant bij VLC en heeft ervaring met meerdere open- en closed-source CMS pakketten.

Mattijs Hoitink zal begeleiding geven op technisch gebied. Mattijs is afgestudeerd informaticus en gecertificeerd PHP ontwikkelaar en is mede-eigenaar van Voidwalkers.

Jan Mooij zal de stage begeleiden vanuit de HU.

Daarnaast zal er een projectpagina zijn te vinden op de project portal van Voidwalkers http://project.voidwalkers.nl. Deze zal voor iedereen toegankelijk zijn.

8 De activiteiten

Hieronder volgt een beschrijving van alle activiteiten die bij het project horen. Tevens worden ook wat activiteiten genoemd die niet bij het project horen, maar die gezien de situatie dat de stagair afstudeert in zijn eigen bedrijf toch zal moeten uitvoeren. Het noemen van deze activiteiten geeft wat duidelijkheid over de context waarin het project uitgevoerd wordt.

8.1 Project activiteiten

De primaire activiteiten zullen bestaan uit het ontwikkelen van het CMS en het schrijven van de scriptie. Met ontwikkelen van het CMS wordt bedoelt: het opstellen van een functioneel ontwerp, het opstellen van een technisch ontwerp, programmeren van het CMS op basis van de ontwerp documenten en het schrijven van secundaire code om de CMS code (automatisch) te kunnen testen (Unit tests). Tevens zal er regelmatig overleg zijn met de inhoudelijk begeleiders (Ronald en Mattijs). Daarnaast zal de staigair als de situatie zich daar voor leent het CMS product demonstreren bij klanten. Dit dient het bedrijf om klanten te werven voor het product, maar dient ook de stage-opdracht doordat klanten hun feedback kunnen geven over het product.

Naast de scriptie moet er ook een plan van aanpak opgeleverd worden (dit document).

8.2 Overige activiteiten

Naast de stage opdracht is de stagair mede verantwoordelijk voor het bedrijf. Dit betekend dat deze ook moet meehelpen aan algemene zaken om het bedrijf draaiende te houden. Hierbij moet gedacht worden aan: administratie en facturen versturen, schoonmaken van het kantoor, koffie drinken / borrellen met potentiele klanten / partners (ook wel netwerken genoemd), onderhouden van de ICT infrastructuur, etc.

Deze activiteiten zullen in verhouding tot het stage project weinig tijd innemen en gezien de royale openingstijden van het VOIDWALKERS kantoor, hoeft dit geenzins de stage opdracht van de staigair te belemmeren.

9 Kwaliteitsbewaking

Om te voorkomen dat het project afdrijft in ongewenste richtingen zal het project opgedeeld worden in korte sprints / iteraties. Voor het managen van deze sprints zullen een aantal methodieken uit SCRUM gebruikt worden. Het is niet de bedoeling het project volledig volgens SCRUM te draaien aangezien dat teveel overhead veroorzaakt voor een één mans project.

De ruim 5 maanden die er voor het project staan zijn vanaf nu nog moeilijk te overzien en het is nog lastig om in te schatten hoe snel de ontwikkeling zal vorderen van het CMS. Om duidelijkheid te scheppen wat er gebouwd gaat worden zal er van te voren een klein functioneel ontwerp gemaakt worden waarin globaal alle functionaliteit uiteengezet wordt in (SCRUM) stories. Deze stories zullen dan vervolgens toegekend worden aan een iteratie. Tijdens de iteraties zal regelmatig overleg met de functioneel begeleider zijn om de voortgang te bespreken en ook vooruit te kijken naar wat er haalbaar is voor de volgende iteratie. In dit document zal de inhoud van de eerste twee iteraties al volledig ingepland zijn en zal er een lijst met functionele eisen aanleverd worden in volgorde van implementatie. Bij elke iteratie zal een haalbaar deel van die lijst (in SCRUM ook wel de backlog genoemd) worden ingepland.

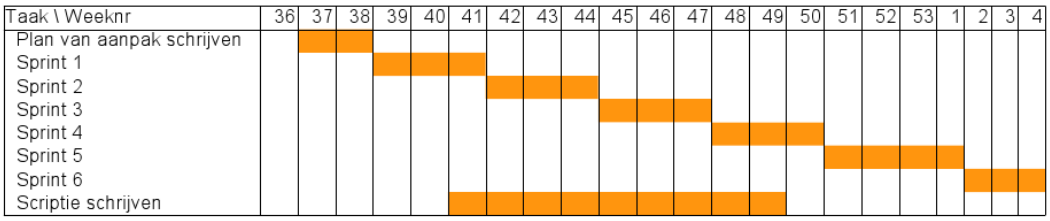
Op technisch vlak zal er gebruik gemaakt worden van versie beheer. Hierdoor zullen alle wijzigingen vast gelegd worden en kan er op elk tijdstip terug gezocht worden. Hierdoor zullen er minder snel bugs in de code komen aangezien er geschiedenis van de regels wordt bijgehouden (dit functioneert ook een beetje als documentatie).

10 Planning

10.1 Deadlines

Voor de stage zijn de volgende deadlines van belang:

- 8 Oktober 2009 - Plan van aanpak & Contract inleveren
- 16 December 2009 - Scriptie inleveren
- 18 December 2009 - Bedrijfsbeoordeling inleveren
- 11 t/m 22 januari 2010 - Afstudeerzittingen



Figuur 1: Tijdsbesteding in een gantt chart.

10.2 Sprint planning

Sprint nr begint op:

1. 21 september 2009 t/m 09 oktober 2009
2. 12 oktober 2009 t/m 30 oktober 2009
3. 2 november 2009 t/m 20 november 2009
4. 23 november 2009 t/m 12 december 2009
5. 14 december 2009 t/m 8 januari 2010
6. 11 januari 2010 t/m 29 januari 2010

10.3 Sprint inhoud

Sprint 1:

- Simpele navigatie mogelijkheden bouwen
- Simpele pagina´s aanmaken, wijzigen en verwijderen

Sprint 2:

- Uitgebreide navigatie via een boom structuur
- Mogelijkheid meer content types te specificeren dan alleen pagina´s via een XML bestand.
- Een speciaal nummer invoer veld.

Backlog:

- Code generatie van PHP code op basis van XML specificatie
- Relaties kunnen specificeren tussen verschillende content types.
- Een speciaal datum type / veld.
- Authenticatie implementeren (login systeem).
- Authorizatie implementeren (een rollen systeem).

- Een speciaal HTML invoer veld (WYSIWYG editor).
- Media Library implementeren voor uploaden en beheeren van plaatjes.
- Mogelijkheid plaatjes te kunnen bewerken (roteren en snijden).
- Een PHP status pagina om buiten het CMS om problemen te kunnen identificeren.
- Mogelijkheid het CMS efficiënt te kunnen doorzoeken.
- Modulariteit van de architectuur optimaliseren.
- Speciaal Google maps invoerveld.
- Video support in de Media Library implementeren.

11 Risico's

Hieronder wordt een poging gedaan alle risico's te beschrijven die mogelijk impact kunnen hebben op dit project.

11.1 Voidwalkers gaat failliet

Voidwalkers is een startende onderneming. Het zou kunnen dat als de crisis lang aanblijft er niet genoeg werk binnenkomt om alle vasten lasten en het levensonderhoud van de eigenaren te kunnen betalen. Hoewel het niet het einde van dit project hoeft te betekenen, zal het wel heel veel gevolgen hebben.

Echter valt er vanuit het project weinig invloed uit te oefenen op dit risico.

11.2 Voidwalkers wordt immens succesvol

Er bestaat natuurlijk de mogelijkheid dat Voidwalkers erg succesvol en dat er daardoor een dusdanige werkdruk en belangenverstrengeling plaatsvind waardoor stage en bedrijfsvoering niet meer te combineren zijn voor de stagair.

Hoewel het bovenstaande scenario onwaarschijnlijk is op het moment van schrijven is het wel mogelijk dat het op kleine schaal op treed. Deadlines die gehaald moeten worden kunnen resulteren in korte termijn keuzes die het stage project niet ten goede komen. Dit risico is te beperken door duidelijke afspraken te maken met alle betrokken partij´en. Duidelijk te zijn in waar de prioriteiten liggen.

Edit (Basic) content

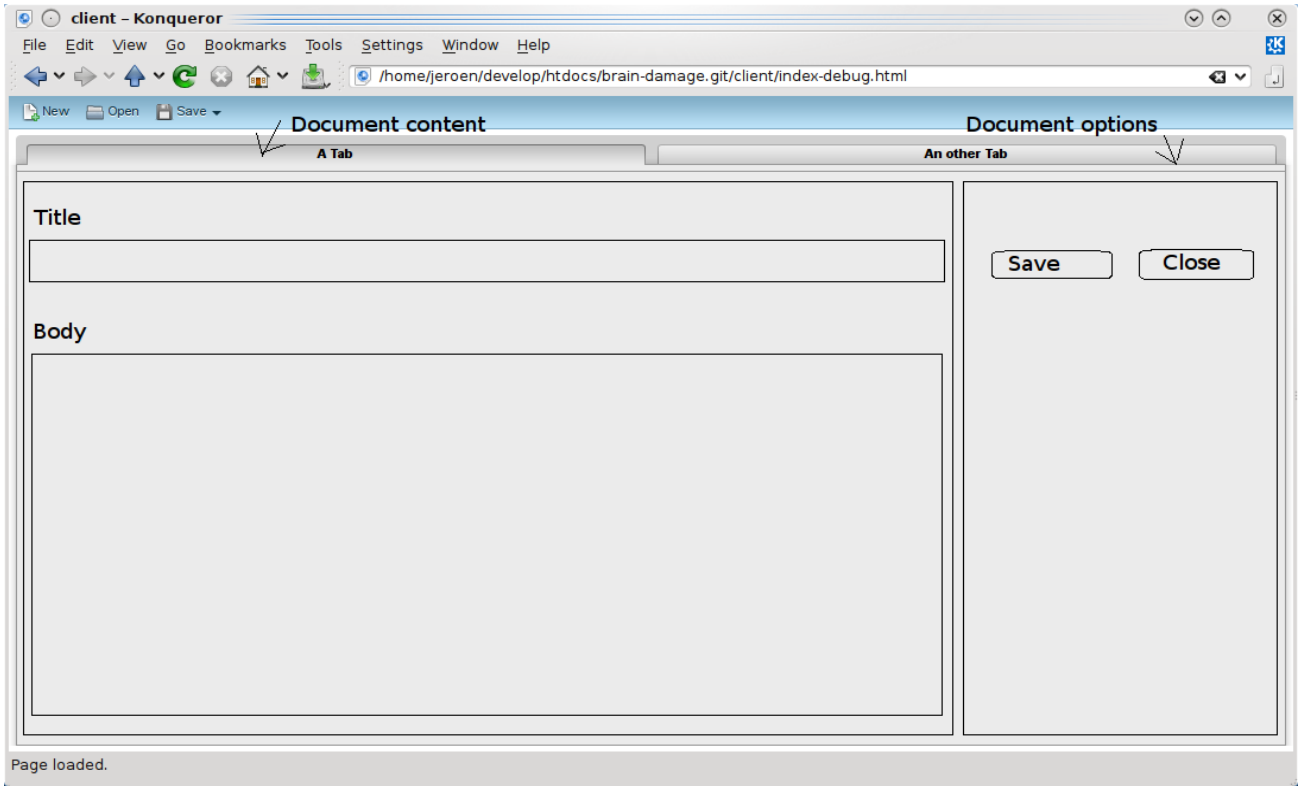
Summary

As a user, I should be able to add and edit basic content like pages. Pages should at least have a title and a body.

Functional overview

After adding new content or editing existing content the application should show all the Widgets filled with the content.
After editing the content the user should be able to either save the content or discard the changes. Saving the changes should persist them in the database.

Technical design



Uc-Basic navigation

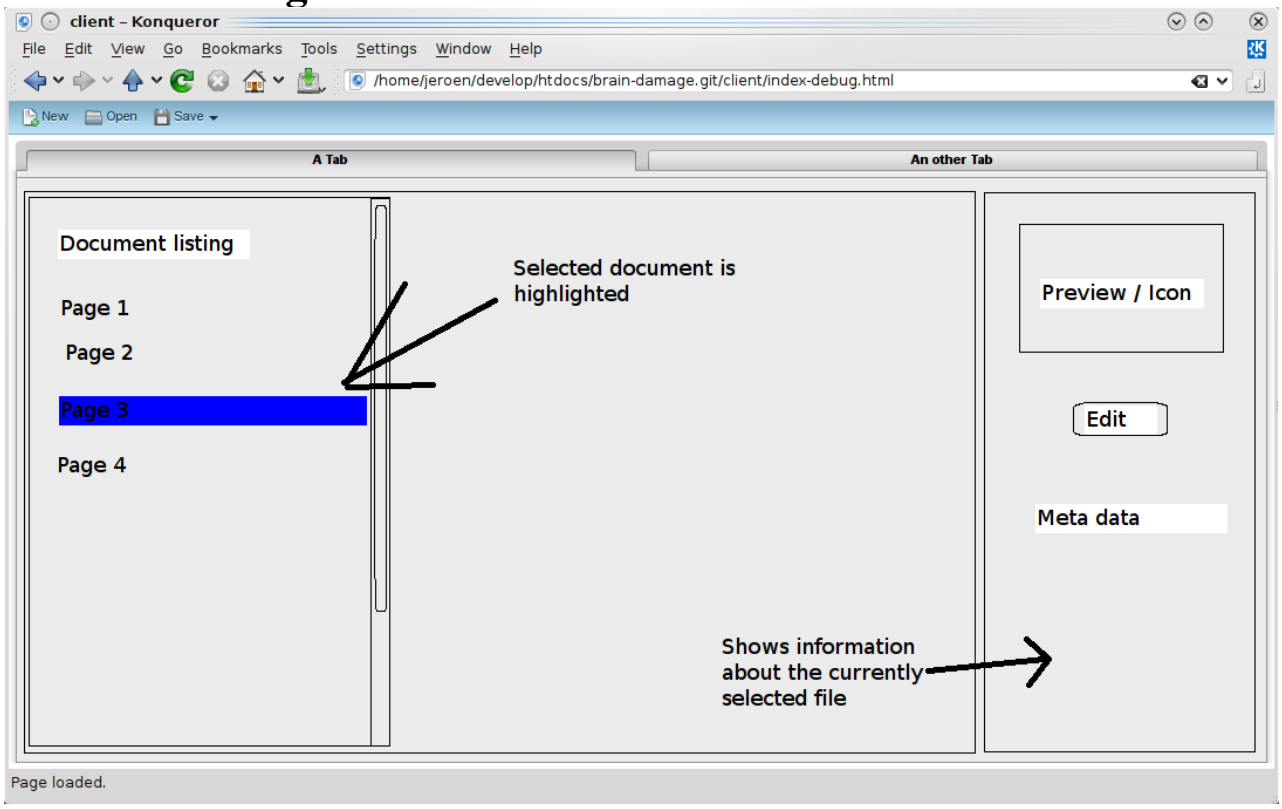
Summary

As a user I want an overview (a list will do) of all my pages and be able to select which one I want to edit.

Functional overview

The overview will be a list off pages.
Selecting a page will present some actions to perform on the page (edit / delete).
The overview will be a flat list, not a tree.

Technical design



Uc-Tree navigation

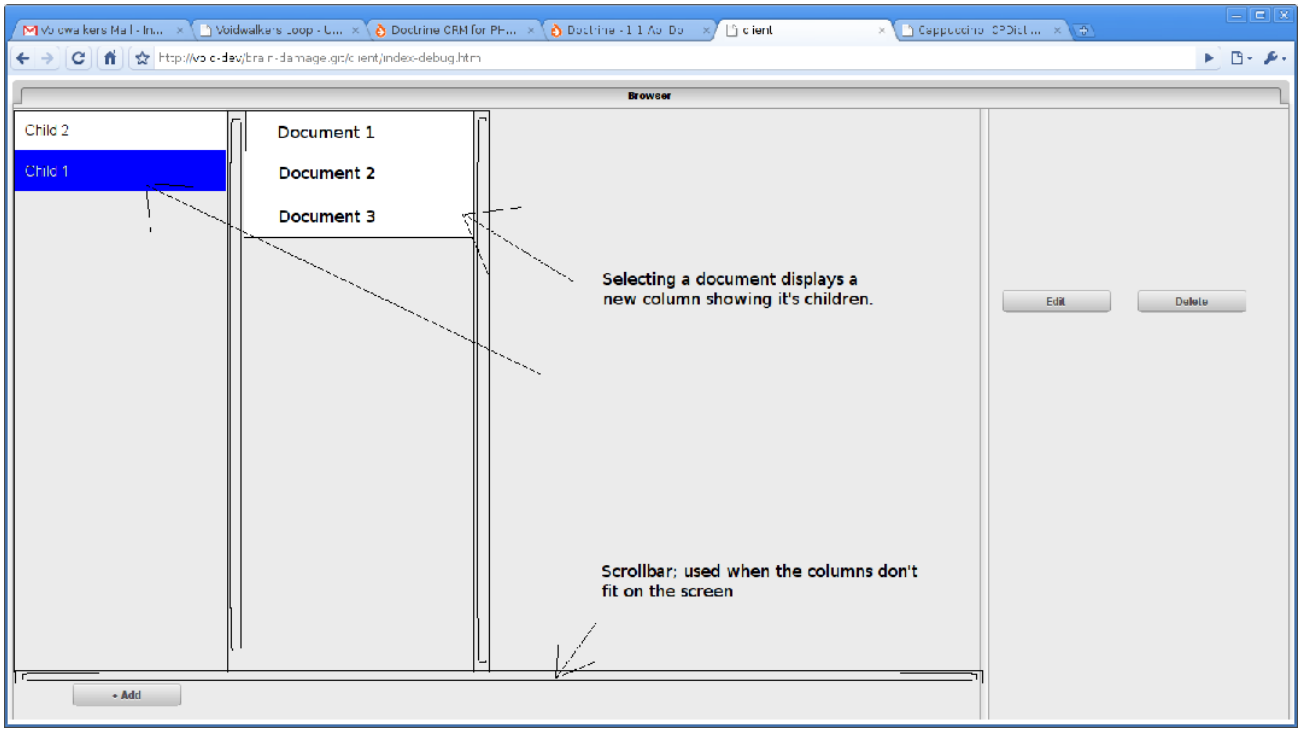
Summary

As a user I want to be able to organize my content in a tree-like fashion.

Functional overview

The user interface should provide a tree-like navigation.
The add/edit dialog off content should provide the possibility to select a parent.

Technical design



The data-structure on the server will use the nested set implementation off Doctrine:

Nested set documentation, Doctrine

Uc-XML content specification

Summary

As a developer I want to link my Doctrine models and types with an easy XML specification.

Functional overview

The Doctrine models and the type system should be linked via a configuration file and not hardcoded. The CMS will refer to the configuration file to find out which code corresponds to which type.

Note that this file will be generated in the future: uc-Code generation

Technical design

Steps to be taken:

- Design a XML file specification
- Update the code to read the content types from the XML file instead of using hardcoded values.
- Make sure the server sends enough meta data to the client that it can work with the models without knowing about the XML.

An example XML:

```
<types>
  <type>
    <name>page</name>
    <fields>
      <field type="string">title</field>
      <field type="html">body</field>
      <field type="relation" relation="user">author</field>
    </fields>
  </type>
  <type>
    <name>photo</name>
    <fields>
      <field type="string">title</field>
      <field type="string">url</field>
      <field type="relation" relation="user">author</field>
    </fields>
  </type>
  <type>
    <name>user</name>
    <fields>
      <field type="string">name</field>
      <field type="string">password</field>
    </fields>
  </type>
</types>
```

Uc-spinnerbox

Summary

As a developer I want a integer Widget type to prevent users typing text in fields that can only contain numbers.

Functional overview

The widget should only allow integer input.
Whould be nice if the input Widget is a spinnerbox, allowing alternative ways of manipulating the input.

Uc-Relations

Summary

As a developer I want to be able to specify relations between the different types I declare in my config file.
As a user I want to be able to easily manipulate relations between content using a dropdown box.

Functional overview

Doctrine allows for generation of models with relations. The CMS code generation should take advantage of that and also generate CMS types with relations between them.
A relation should be implemented as a special type which should result in a dropbox in the CMS for manipulation.

Uc-Datewidget

Summary

As a user I do not want to add dates as text but instead use a more visual way of inserting dates.

Functional overview

Date input fields should have a button that presents a calendar to chose a date from.

Uc-HTML widget

Summary

As a CMS user I do not want to input plain HTML. Instead I would like to have a WYSIWYG editor that handles the transformation to HTML for me.

Functional overview

There should be a HTML text type with a corresponding widget. The frontend (website) should be able to retrieve the raw HTML result.

Technical design

http://www.ibm.com/developerworks/library/wa-aj-build/

Uc-Auth

Summary

As a site owner I want to select the people that are able to manipulate my content.

Functional overview

User of the site should need to identify them selfs using a username and password.

Uc-Media library

Summary

As a CMS user I want to upload and manage images. Also I would like to be able to add images to content.

Functional overview

There should be some sort of gallery that allows managing and uploading of new images.
There should be a widget / image type that allows adding of images to content.
There will NOT be and option for adding images inline in HTML fields.

Uc-ACL

Summary

As a big website owner I want to give specific privileges about who is able to edit what content.

Functional overview

The admin user interface should be able to grant users access to specific parts of the content tree.
Also the admin interface should be albe to deny or grant users access to whole parts of the CMS.

Uc-Image manipulation

Summary

As a user I would like to have simple options of image manipulation in the CMS, so I don't have to resort things like Photoshop.

Functional overview

The media library should have options to rotate and crop images. The CMS should also make sure the image is saved using a "sane" compression level, preventing giant image sizes of less competent users uploading files.

Uc-Status page

Summary

As a developer / sysadmin I would like to have a robust status page to be able to diagnose problems in case the webservice / CMS is not functioning properly.

Functional overview

This status page should be able to show the status of all CMS components.
Maybe show the lasts few log entries of several log files.
Check database connections. Show database "health".

Uc-Code generation

Summary

As a developer I do not want to generate and link my models manually. This should be done automatic.

Functional overview

Developers should declare their types in a simple file format. This file should then be used to generate the Doctrine code and the XML specification.
Developers should only need to worry about the models in case of non-standard modifications.

Uc-Search

Summary

As a CMS user I want to be able to search for my content in case I forgot where it is.

Functional overview

It would be nice to search all textfields in all content.
Maybe even index image files (descriptions, etc.).

Technical design

Use Zend_Search_Lucene to index all text and keep it fast?

Uc-Modularity

Summary

As a developer I do not want my custom code to interfere with CMS code ensuring a clean upgrade path.

Functional overview

This should be a design goal throughout the whole project, but this story should be a gatekeeper to make sure the CMS is modular and a developer can simple overwrite the CMS with a new version without worrying about changes getting lost.

Uc-Google maps

Summary

As a CMS user I want to be able use Google maps natively in the CMS instead of using textfields to enter x-y coords.

Functional overview

The Widget should use the Geocode webservice to translate adres information to coordinates.
The Widget should allow the user to manually place a marker somewhere on a map.
The Widget should also allow the user to still manually enter x-y coords.

Uc-Video

Summary

As a user I want an easy way to add video content to my website.

Functional overview

There should be a way to upload video's.
It would be nice to integrate with one or more flash video content providers (youtube, viddix, ?)
It mite also be nice to support HTML 5 video element.
The video's should integrate with the media library.