|  |
| --- |
| DragonEye Software |
| Controlcenter2 |
| Analyse & uitwerking |

|  |
| --- |
| Copyright DragonEye Webdesign 2008 |

Inhoudsopgave

[1 Algemeen 5](#_Toc273287478)

[1.1 Wat? 5](#_Toc273287479)

[1.2 Hoe 5](#_Toc273287480)

[2 Verdere uitdieping van de functionaliteit 6](#_Toc273287481)

[2.1 Templatesystem 6](#_Toc273287482)

[2.1.1 Filosofie 6](#_Toc273287483)

[2.2 Gebruikerssysteem 7](#_Toc273287484)

[2.2.1 Algemene doelstellingen 7](#_Toc273287485)

[2.2.2 Community vs bedrijf 8](#_Toc273287486)

[2.2.3 Session Management 8](#_Toc273287487)

[2.3 Beveiliging 8](#_Toc273287488)

[2.4 Configuratie 9](#_Toc273287489)

[2.5 User preferences 9](#_Toc273287490)

[2.6 Logging 9](#_Toc273287491)

[2.7 Modules 9](#_Toc273287492)

[2.7.1 Algemeen 9](#_Toc273287493)

[2.7.2 De structuur van moddef.xml 10](#_Toc273287494)

[2.7.3 Upgrades 11](#_Toc273287495)

[2.8 Google Sitemaps 11](#_Toc273287496)

[3 Structuur van de code 11](#_Toc273287497)

[3.1 Algemeen 12](#_Toc273287498)

[3.2 Dataverbinding 12](#_Toc273287499)

[3.3 Structuur van CORE 12](#_Toc273287500)

[4 Bestandsstructuur 14](#_Toc273287501)

[5 Algemene structuur 15](#_Toc273287502)

[5.1 UML klassenmodel 16](#_Toc273287503)

[5.1.1 De klassen 17](#_Toc273287504)

[5.1.2 De Kardinaliteiten 19](#_Toc273287505)

[6 Kernprocessen 20](#_Toc273287506)

[7 Uitwerking 21](#_Toc273287507)

[7.1 Exception handling 21](#_Toc273287508)

[7.2 Beveiligde bestanden 21](#_Toc273287509)

[7.3 Increment1: Databaseverbinding 21](#_Toc273287510)

[7.3.1 Beveiliging 22](#_Toc273287511)

[7.3.2 Syntax van de configuratiefile 22](#_Toc273287512)

[7.3.3 Datacomponent selector 23](#_Toc273287513)

[7.3.4 Data Componenten 23](#_Toc273287514)

[7.3.5 Beveiliging tegen SQL Injection 27](#_Toc273287515)

[7.3.6 Samenvatting 27](#_Toc273287516)

[7.4 Increment2: Parameters 27](#_Toc273287517)

[7.4.1 De code 29](#_Toc273287518)

[7.4.2 Samenvatting 29](#_Toc273287519)

[7.5 Increment3: Templatesystem 29](#_Toc273287520)

[7.5.1 Structuur van het templatesystem 30](#_Toc273287521)

[7.5.2 Template aliases 30](#_Toc273287522)

[7.5.3 Template Addins 30](#_Toc273287523)

[7.5.4 Implementatie van meertaligheid 32](#_Toc273287524)

[7.5.5 Structuur van templatefiles 35](#_Toc273287525)

[7.5.6 Variabelen 36](#_Toc273287526)

[7.5.7 Custom Tags 36](#_Toc273287527)

[7.5.8 Overzicht van de custom tags 38](#_Toc273287528)

[7.5.9 De Klasse HTMLPage 40](#_Toc273287529)

[7.5.10 Templatesystem Revisie1 40](#_Toc273287530)

[7.5.11 Samenvatting 41](#_Toc273287531)

[7.6 Increment4: Aanmaken/wijzigen van gebruikers 42](#_Toc273287532)

[7.6.1 Inleiding 42](#_Toc273287533)

[7.6.2 De klasse USER 43](#_Toc273287534)

[7.6.3 De database 44](#_Toc273287535)

[7.6.4 Wachtwoordbeveiliging 45](#_Toc273287536)

[7.6.5 Validatie van gegevens 45](#_Toc273287537)

[7.6.6 Externe registratie 49](#_Toc273287538)

[7.6.7 Samenvatting 49](#_Toc273287539)

[7.7 Increment5: Gebruikersgroepen en Permissions 50](#_Toc273287540)

[7.7.1 Inleiding 50](#_Toc273287541)

[7.7.2 Uitbreiding van de database 51](#_Toc273287542)

[7.7.3 De Klasse usergroup 53](#_Toc273287543)

[7.7.4 Toevoegen en wijzigen van usergroups 54](#_Toc273287544)

[7.7.5 Samenvatting 54](#_Toc273287545)

[7.8 Increment6: Koppeling tussen gebruikers en gebruikersgroepen 54](#_Toc273287546)

[7.8.1 Doelstelling 54](#_Toc273287547)

[7.8.2 Toevoegen van gebruikers aan een gebruikersgroep bij creatie 54](#_Toc273287548)

[7.8.3 Gebruikersgroepen en externe registratie 54](#_Toc273287549)

[7.8.4 Samenvatting 54](#_Toc273287550)

[7.9 Increment7: Verificatie van extern geregistreerde gebruikers 55](#_Toc273287551)

[7.9.1 De Klasse Email 57](#_Toc273287552)

[7.9.2 Templatesystem revisie 2 57](#_Toc273287553)

[7.9.3 Wijzigingen aan de databasetabel users 58](#_Toc273287554)

[7.9.4 Wijzigingen aan het userobject 58](#_Toc273287555)

[7.9.5 Versturen van mail bij het activatieproces 58](#_Toc273287556)

[7.9.6 Integratie van reCaptcha in Controlcenter2 59](#_Toc273287557)

[7.9.7 Toegevoegde functies 59](#_Toc273287558)

[7.10 Samenvatting 59](#_Toc273287559)

[8 Increment 8: inloggen en beveiliging 60](#_Toc273287560)

[8.1 Inloggen 60](#_Toc273287561)

[8.2 Login.php 60](#_Toc273287562)

[8.3 CheckPermission 61](#_Toc273287563)

[8.4 Samenvatting 62](#_Toc273287564)

# Algemeen

## Wat?

Controlcenter2 moet een geïntegreerde oplossing bieden voor het maken van dynamische websites. Dergelijke sites hebben vaak problemen voor het verzorgen van updates. Gebruikers kunnen vaak zelf gegevens toevoegen die geverifieerd moeten worden.

De bedoeling is om een framework te voorzien dat de steeds terugkerende functionaliteit voorziet voor dergelijke sites. Er wordt dan vooral het volgende bedoeld:

* Templatesysteem
* Gebruikersbeheer
* Configuratie
* Beveiliging
* Logging
* Uitbreidbaarheid

Door een systeem te bouwen dat ervoor zorgt dat bovenstaande functionaliteit kan worden voorzien en dat toch voldoende mogelijkheden biedt tot uitbreiding kan aan de noden van dergelijke sites worden tegemoetgekomen.

Het idee achter controlcenter is om een management omgeving te voorzien. De omgeving biedt toegang tot de standaardfuncties van controlcenter maar kan uitgebreid worden met modules die specifieke functies voorzien. Op die manier moet er geen tijd geïnvesteerd worden in het ontwikkelen van een zoveelste gebruikersbeheer en kan de beschikbare tijd aangewend worden voor het ontwikkelen van de specifieke functionaliteit. Dit platform zou ervoor zorgen dat de DragonEye websites betrouwbaarder worden en sneller kunnen worden ontwikkeld.

## Hoe

De bedoeling is om controlcenter te ontwikkelen in PHP5. Op deze manier wordt er ondersteuning voorzien voor zowel Windows als Linux omgevingen.

Het project zal in een Multi-tier systeem worden opgebouwd. Aanvankelijk zal de enige ondersteunde database technologie MySQL zijn.

Volgende lagen worden voorzien:

* Presentation: Door de complexiteit die een grote HTML code meebrengt zal deze laag in 2 worden opgesplitst
  + Template systeem
  + Algemene presentation Layer
* Business Logic: het geheel van controles
* DataAccess: de verbinding met de database.
* FileAccess: Opladen en openen van bestanden + verwerking van bijvoorbeeld XML files.
* BusinessEntity: Objecten die in alle lagen gebruikt worden.

Voordeel van deze benadering is dat enkel de DataAccess layer afhankelijk is van een databasesysteem. Er kunnen dus later DataAccess componenten worden ontwikkeld voor andere database systemen zoals MsSQL en PostGreSQL.

Eén van de problemen is dat PHP geen manier van opsplitsing van de tiers voorziet zoals bijvoorbeeld .NET Om die reden zal de splitsing tussen de layers dun kunstmatig geprogrammeerd moeten worden . Controle op effectieve splitsing van de layers zal dus noodzakelijk zijn om de verdere ontwikkeling makkelijker te maken.

Er wordt voor geopteerd om geen gebruik te maken van HTML maar wel van XHTML, deze variant is strenger maar zorgt voor snellere code met grotere compatibiliteit. Een bijkomende bepaling is dat AJAX waar mogelijk de norm moet zijn. Dit kan een stuk de belasting verplaatsen van de serverkant naar de clientkant, maar de gebruikerservaring wordt hierdoor verbeterd. Er zal dus in de presentationlayer een stuk Javascript moeten worden voorzien.

# Verdere uitdieping van de functionaliteit

## Templatesystem

### Filosofie

Dit systeem wordt een evolutie van het systeem dat werd gebruikt in controlcenter1.x .Er worden echter een aantal uitbreidingen voorzien die de functionaliteit ten goede moeten komen en ook tegemoet moeten komen aan een aantal noden die controlcenter1 niet kon invullen.

Een belangrijke wijziging ten opzichte van vorige versies is de meertaligheid. De bedoeling is om controlcenter2 te leveren in verschillende talen en ook de mogelijkheid te integreren om zowel backend als frontend in meerdere talen te kunnen aandrijven. Om dit implementatie van deze meertaligheid zo simpel mogelijk te maken moet deze voorzien worden in het templatesysteem.

De wijzigingen zullen ervoor zorgen dat er een koppeling zal noodzakelijk zijn tussen de database en het templatesysteem (via BusinessLogic -> DataAccess).

In het templatesysteem van controlcenter 1 gebeurde er een aanroep van een bestand op de server dat de HTML-code van het template bevatte. Dit werkte zeer limiterend wanneer men meerdere templates wilde voorzien en wilde overschakelen tussen de verschillende templates zonder code te veranderen en zonder het oude template te verwijderen.

Daarom wordt in controlcenter2 een nieuw systeem van templatealiases voorzien. De aliases zullen gedefinieerd worden in de database. In de code zal er enkel nog verwezen worden naar een alias. In de databasetabel wordt deze alias dan gekoppeld aan een HTML file.

Standaard zullen volgende aliases aanwezig zijn:

* Backend: deze alias zal gebruikt worden voor de backend
* Login: Deze alias zal gebruikt worden voor het loginscherm
* Frontend: Dit zal een algemene opmaak zijn voor de frontend.
* Error: een template voor het weergeven van errormessages

Door deze aliases in de database te definiëren zal de mogelijkheid bestaan om nieuwe aliases toe te voegen. Zo kunnen modules dus hun eigen aliases definiëren die dan bijvoorbeeld aan een afwijkende layout kunnen gekoppeld worden.

De bedoeling is om zo grote flexibiliteit te voorzien met zo weinig mogelijk extra code.

Ook de meertaligheid zal voor een grotere complexiteit zorgen. Meertaligheid heeft verschillende gevolgen:

* Er kunnen grafische verschillen zijn tussen verschillende taalversies (tekst in grafics).
* Er kunnen tekstverschillen zijn (tekst in HTML-code)

Er moeten ook flexibeler mogelijkheden voorzien worden om constanten te defniëren en voor het uitvoeren van externe files in het template.

## Gebruikerssysteem

### Algemene doelstellingen

Het gebruikerssysteem zal gebruikt worden voor het aanbieden van sitespecifieke functionaliteit maar ook voor het updaten en onderhoud van de site. Door te kiezen voor 1 gebruikerssysteem voor zowel administrators als gewone gebruikers zal het mogelijk zijn om bij community-sites accounts van gebruikers aan te passen zodat zij kunnen meewerken aan de opbouw van de site.

Om controlcenter op zowel communities als voor gewone dynamische sites te gebruiken zal het gebruikerssysteem in grote mate aanpasbaar moeten zijn.

Er wordt dan in eerste instantie vooral aan volgende flexibiliteit gedacht:

* Op een community site moeten gebruikers zich kunnen registreren, ze zullen dus hun eigen account aanmaken. Op een bedrijfssite zullen alleen administrators een account hebben en deze zullen ze krijgen. Er zal dus een flag moeten voorzien worden die bepaalt of gebruikers zichzelf kunnen registreren.
* Niet iedere gebruiker zal dezelfde toegang hebben, er zal dus met een systeem van machtigingen moeten gewerkt worden. Die machtigingen moeten vlot aanpasbaar zijn. Bij grote groepen gebruikers zullen er eventueel gebruikersgroepen moeten worden kunnen aangemaakt om het beheer van de toegang te vergemakkelijken.
* Verschillende sites vereisen ook verschillende gegevens die moeten worden bijgehouden. Er zou dus een systeem moeten voorzien worden waarbij er gebruikersvelden kunnen worden toegevoegd door modules, zodat men bijvoorbeeld een adres kan bijhouden of het merk van gitaar van een gebruiker.
* Het systeem om in te loggen zal eventueel moeten kunnen geïntegreerd worden in het basistemplate van de site zodat een gebruiker vanop de hoofdpagina kan inloggen. Bij een bedrijfssite is dit uit den boze.
* De bedoeling is om zowel kleine sites als grote communities te bouwen met Controlcenter2. Om die reden moet het systeem gebruikersgroepen ondersteunen zodat grote groepen gebruikers gemakkelijk beheerd kunnen worden. Daarenboven moet het ook mogelijk zijn van een gebruiker geen lid te maken van een groep en manueel de machtigingen per gebruiker in te stellen.

### Community vs bedrijf

Probleem hier is dat we bij een community site enkele persoonlijke gegevens willen bijhouden. Bij een bedrijfssite zal het al snel beginnen op de zenuwen werken als je al teveel gegevens moet invoeren.

Hiervoor moet er dus een oplossing worden gezocht. Eventueel zou er kunnen geopteerd worden om bij het registratieproces gewoon naam, mail, gebruikersnaam en wachtwoord te vragen en bij eerste login kan er dan eventueel een procedure gestart worden voor het opgeven van die gegevens.

Het grote probleem is dat we niet weten welke gegevens er precies zullen worden bijgehouden. Wanneer je een site over tv-shows aandrijft zal er bijvoorbeeld een favoriete serie gevraagd worden. Bij een gitaarsite zal het eerder gaan over welk merk gitaar. Op zich is het geen probleem om modules toe te laten HTML toe te voegen aan de registratiepagina. Het probleem schuilt in de validatie van deze ‘custom fields’.

### Session Management

Een grote handicap van een webplatform is dat een HTML-pagina stateless is. Dus moet er een manier geïmplementeerd worden om bij te houden van een ingelogde gebruiker zonder dat dit een beveiligingsrisico met zich meebrengt. Hiervoor zal gebruik gemaakt worden van het ingebouwde session management van PHP5.

Wanneer een gebruiker zich inlogt zal een sessievariabele $\_SESSION[‘currentuser’] worden aangemaakt. Deze variabele zal overal moeten beschikbaar zijn.

## Beveiliging

Een systeem van machtigingen is natuurlijk nutteloos als er geen manier is om deze machtiging te controleren. Het controleren van de toegang moet zo simpel mogelijk zijn (bij voorkeur 1 regel code aan het begin van de file). De toegang kan gecontroleerd worden aan de hand van $\_SESSION[‘currentuser’]. Om de verwijzingen in de code simpeler te maken zou er bij het definiëren van de machtiging moeten kunnen gebruikgemaakt worden van woorden ipv id-nummers.

Het systeem dat werd gebruikt in Controlcenter1 kan hier als basis dienen maar zal moeten uitgebreid worden.

Het machtigingen systeem in Controlcenter1 werkte op moduleniveau: een gebruiker had toegang tot een module of had geen toegang. Dit blijkt echter niet in overeenstemming met de realiteit, daarom zullen er aparte permissions gedefinieerd worden die wel afhankelijk zijn van een module. Modules zullen dus permissions kunnen toevoegen.

## Configuratie

Modules die voor uiteenlopende doelstellingen zullen worden gebruikt moeten flexibel zijn. Het is bijvoorbeeld goed mogelijk dat een module voor het installeren van een webdirectory bij de ene site een betaling zal vereisen voor de site wordt toegevoegd en op de andere site niet.

Op een bepaalde site zal de voorgestelde url eerst gecontroleerd worden, bij andere directories niet.

Er zal dus nood zijn aan een manier om de modules te configureren. Op zich zou dit probleem naar de individuele modules kunnen verschoven worden maar dat zou de algemene structuur van Controlcenter2 niet ten goede komen. Beter een mogelijkheid die een configuratiescherm voorziet dat kan uitgebreid worden met parameters per module. Op die manier zal de interface simpeler worden voor het gebruik. De moeilijkheid zal de uitbreidbaarheid worden.

## User preferences

Gebruikers gaan moeten hun eigen voorkeuren kunnen doorgeven, ook dat moet mogelijk zijn.

## Logging

Vooral bij community sites maar ook bij bedrijfssites met meerdere administrators is een beperkte controle op misbruik nodig. Om die reden zal Controlcenter2 een logboekfunctionaliteit aanbieden.

De bedoeling is dat de acties van gebruikers kunnen worden bijgehouden. Er moet een systeem ontwikkeld worden om op een eenvoudige manier de acties van gebruikers te traceren.

Het logsysteem moet ook uitbreidbaar zijn, zodat modules er ook gebruik van kunnen maken.

Om de uitbreidbaarheid te garanderen en ervoor te zorgen dat het logboek gestructureerd blijft zouden er in de database Actions kunnen gedefinieerd worden. Dan kunnen modules zelf actions definiëren en gebruiken. In de weergave van de logs kan er dan mogelijkheid voorzien worden om op actions te sorteren.

## Modules

### Algemeen

Uiteindelijk draait Controlcenter2 rond het installeren van modules. Enkel met de basisfuncties van controlcenter2 kan er geen site worden gebouwd.

Om het gebruik van Controlcenter2 gemakkelijk en efficiënt te maken moet er dus een eenvoudig te gebruiken systeem worden voorzien om modules te maken en te integreren.

Eén van de functies die de modules ongetwijfeld het meest gaan gebruiken is $\_SESSION[‘currentuser’] om gebruikersgegevens te verwerken.

Het installeren van modules zal onvermijdelijk wijzigingen teweegbrengen in de database van Controlcenter2. Het is echter onverstandig om bij de installatie van een module gebruik te maken van enkel SQL scripts. Het gebruik van een SQL script kan eventueel toegestaan worden wanneer er extra tabellen moeten worden toegevoegd. Het is echter een onverstandige keuze bij het toevoegen van records aan de verschillende tabellen van de basisdatabase. Daarom wordt het gebruik van XML files voorgesteld als basis voor doorgeven van de module-specifieke parameters. Zoals in vorige rubrieken reeds vermeld zal de integratie van een nieuwe module vooral gebeuren door het toevoegen van records aan de tabellen. Hierdoor zal een module bij installatie volgende bewerkingen kunnen uitvoeren:

* Toevoegen van templatealiases
* Toevoegen van permissions
* Toevoegen van actions voor logging
* Toevoegen van constants
* Toevoegen van velden in gebruikersgegevens.
* Toevoegen van gebruikersvoorkeuren.
* Toevoegen van menuitems in Controlcenter menu, eventueel taalspecifiek

### De structuur van moddef.xml

Om de installeren van modules uniform te laten verlopen zal iedere module een file moddef.xml moeten hebben. Een voorbeeld van de eventuele structuur.

<MODULE>  
<NAME>Modulenaam in de database</NAME>  
<VERSION>1.01.32</VERSION>  
<MENUNAMES>  
 <MENUNAME lang=”dutch”>Web Directory</NAME>  
 <MENUNAME lang=”english”>Web Directory</NAME>  
</MENUNAMES>   
<TEMPLATEALIASES>  
<ALIAS>alias1</ALIAS>  
<ALIAS>alias2</ALIAS>  
</TEMPLATEALIASES>  
  
<PERMISSIONS>  
<PERMISSION>permission1</PERMISSION>  
<PERMISSION>permission2</PERMISSION>  
</PERMISSIONS>  
 <CONSTANTS>  
<CONSTANT name=”naam1”>Waarde</CONSTANT>  
<CONSTANT name=”naam2”>Waarde</CONSTANT>  
</CONSTANTS>  
  
  
</MODULE>

Codevoorbeeld 1: moddef.xml

### Upgrades

Het is mogelijk dat er voor modules die voor 1 specifieke site zijn ontworpen manuele updates gebeuren. Maar een feit is dat modules die op meerdere sites gebruikt worden ook gemakkelijk moeten onderhouden kunnen worden.

Dit betekent vooral dat wanneer er een nieuwe versie van een module beschikbaar is dat er dat er dan een upgrade moet kunnen worden uitgevoerd naar de nieuwe versie. Op bestandsniveau is er hier geen probleem: de bestanden die bij 1 module horen zullen in 1 directory geplaatst worden en daardoor zal het overschrijven van de bestanden ervoor zorgen dat de broncode geupdated is. Een groter probleem is echter de database. Wanneer een nieuwe versie een gewijzigde databasestructuur gebruikt zal de bestaande structuur moeten worden aangepast maar er zal mogelijk ook een conversie van gegevens moeten gebeuren.

Om deze problemen te kunnen oplossen moet het systeem op ieder moment kunnen nagaan wat de geïnstalleerde versie is van een module en wat de versie is van de code van de module. Bijgevolg zal er in de database bij de tabel modules een veld versie moeten voorzien worden. Op bestandsniveau kan dit probleem opgelost worden door in moddef.xml een tag mode te voorzien.

Uiteindelijk moet er een soort van masterserver worden gestart die bijhoudt wat de nieuwe versie is van iedere module en wat de compatibiliteitsstatus is van de verschillende versies van modules met de verschillende versies van controlcenter. Op die manier zou controlcenter automatisch kunnen nagaan of er updates beschikbaar zijn en of het installeren van een update voor de CORE problemen meebrengt voor de geïnstalleerde modules. Dit systeem zou onder de vorm van een maintenance script kunnen worden aangeboden. Wanneer het script merkt dat er inderdaad updates zijn dan zou een mail kunnen gestuurd worden naar de beheerder om die in te lichten dat er een mogelijkheid tot updaten is. De bedoeling is om in controlcenter2 een updater in te bouwen zodat updates automatisch gedownload en geïnstalleerd kunnen worden.

## Google Sitemaps

Vooral voor community sites, maar ook voor bedrijfssites is het belangrijk om gevonden te worden wanneer gebruikers via zoekmachines op zoek zijn naar informatie. Voor een groot stuk zal het succes bij zoekmachines afhankelijk zijn van de optimalisatie van het template en de modules. Het is echter zo dat Google als grootste zoekmachines een systeem van sitemaps in gebruik heeft. Het gaat om XML-files die je doorgeeft aan de zoekmachine met daarin een overzicht van alle pagina’s die actief zijn op de site.

Het opmaken van een dergelijke file kan eventueel door modules gebeuren maar bij verschillende modules zou je dan uiteindelijk ook verschillende sitemaps bekomen. Daarom is het beter om een Sitemap generator te implementeren in CORE. Het zal hier om een klasse gaan die kan aangeroepen worden om vanuit gelijk welke module pagina’s toe te voegen aan de sitemap.

# Structuur van de code

## Algemeen

Het volledige systeem moet Multi-tier worden gebouwd. De materie van de modules vormt hier echter wel een probleem want een module zal bijvoorbeeld ook verbinding moeten maken met de database. En ook daar zal rekening moeten gehouden worden met het type database dat gebruikt wordt. Om die reden zullen er meer lagen moeten worden gemaakt dan enkel de standaardlagen.

Er moet duidelijk gesteld worden dat de code objectgeorienteerd zal worden opgebouwd. Alleen op die manier kan het systeem overzichtelijk blijven.

## Dataverbinding

Omdat de keuze van het databasesysteem een grote invloed heeft op de code moet er een manier gevonden worden om een abstractere vorm van databasecommunicatie te ontwikkelen. Het is zo dat er voor ieder ondersteund database systeem een DataAccess component moet ontwikkeld worden.

Het zal belangrijk zijn om een grote uniformiteit te hanteren bij het ontwikkelen van de DataAccess componenten zodat de andere lagen geen rekening hoeven te houden met de keuze van databasesysteem. Alleen op deze manier kan perfecte compatibiliteit met andere systemen dan MySQL gegarandeerd worden.

Eventueel zullen complexere modules ook Multi-tier moeten ontwikkeld worden zodat er ook daar een dataaccess component zal aanwezig zijn. Dit dataaccess component zal dan gebruik maken van het dataaccess component dat in controlcenter2 aanwezig is.

Het komt er dus eigenlijk op neer dat er een CORE zal worden ontwikkeld die alle basisfunctionaliteit van controlcenter2 zal bevatten. De CORE wordt volledig Multi-tier opgebouwd. Wanneer de modules ook verschillende layers gebruiken zullen de functies daar erven van de CORE-layers.

De functies in de verschillende dataaccess componenten zijn gelijk. De DataAccess selector zal 1 bepaald dataaccess component laden afhankelijk van de configuratie van Controlcenter. Op die manier zullen de functienamen in onderliggende layers dus gelijk blijven maar kan er wel vlot geschakeld worden tussen verschillende database types.

## Structuur van CORE

Het is dus eigenlijk zo dat Controlcenter2 eigenlijk rond de CORE-functionaliteit draait. Deze functionaliteit is aanwezig na installatie, ze kan dan worden uitgebreid met extra modules.

Het is dus zo dat er binnen de CORE-functionaliteit een Multi-tier architectuur zal bestaan. Als deze ook binnen de modules bestaat zal dit onder een abstractere vorm zijn, de dataaccess layer zal in een module geen database specifieke code bevatten. Maar enkel queries, koppelingen naar de CORE DataAccess, etc.



Figuur 1: Structuur van CORE

# Bestandsstructuur

Uit al het voorgaande kan er al een redelijke bestandsstructuur worden afgeleid die zal moeten worden gehanteerd:

/root

/core

/dataaccess

/entity

/fileaccess

/businesslogic

/presentation

/usermanagement

/logging

/…

/modules

/templates

/login

/backend

/frontend

# Algemene structuur

Door de flexibiliteit zal structuur van CORE relatief abstract zijn

## UML klassenmodel



Figuur 2: UML-model

### De klassen

#### Modules

De modules vormen de kern van Controlcenter. Een module zal geïnstalleerd worden door de bestanden die bij de module behoren te kopieren naar de server. Iedere module zal een moddef.xml file moeten bevatten waarin de basisgegevens staan voor het gebruik van de module. Op basis van deze gegevens zal de module geïnstalleerd worden. In de modules moet een onderscheid gemaakt worden tussen 2 soorten:

* Components: Deze modules bevinden zich binnen de Controlcenter CORE. Ze kunnen niet verwijderd worden
* Modules: Deze modules bevinden zich onder /modules. Het gaat om de functionaliteit die na het installeren van controlcenter wordt toegevoegd.

Wanneer het over de databasestructuur gaat zal er geen onderscheid gemaakt worden tussen beide types aangezien er praktisch gezien ook geen verschil is buiten de locatie van de bestanden. Om ervoor te zorgen dat de verschillende locatie geen problemen veroorzaakt zal er een selector M of C voorzien worden in deze tabel zodat scripts kunnen nagaan wat het type is van de module.

#### Permission

Een module zal verschillende taken kunnen uitvoeren het gaat dan om bijvoorbeeld een beheer voor gebruikers. Een dergelijk module zal bijvoorbeeld gebruikers kunnen toevoegen, wijzigen en verwijderen. Voor ieder van deze taken kan er een moduletask gecreëerd worden. Deze zal dan gebruikt kunnen worden om de toegangsrechten te bepalen van een gebruiker. Op die manier zal het beveiligingssysteem beter werken dan dat in controlcenter 1.x. In plaats van een gebruiker toegang te geven tot de volledige module of tot niks zal een gebruiker bijvoorbeeld berichten in een forum kunnen aanpassen maar niet verwijderen.

Het is wel zo dat een Permission niet noodzakelijk bij een module hoort. Er kunnen vanuit CORE Permissions gedefinieerd worden zonder module.

#### User

Gebruikers zullen gecreëerd worden voor alle betrokken partijen van de site. Het gaat dan om bijvoorbeeld administrators, editors, maar ook gewone bezoekers die op die manier hun voorkeuren willen bijhouden.

#### ModuleAction

Een moduleaction is een bewerking die kan uitgevoerd worden door een gebruiker. ModuleActions zullen gebruikt worden voor het loggingsysteem. Het verschil tussen een ModuleAction en een ModuleTask schuilt in hun gebruik. Een ModuleTask zal worden gebruikt voor het beveiligen van de site. Een ModuleAction wordt gebruikt voor het logboek.

#### UserLog

In bepaalde gevallen zal men willen bijhouden welke gebruiker bepaalde bewerkingen heeft uitgevoerd. Voor iedere bewerking van het logboek zal een UserLog moeten worden aangemaakt.

Door tussen ModuleAction en user deze associatieve klasse toe te voegen zullen acties in detail kunnen worden opgeslagen.

#### Usergroup

Omdat controlcenter ook voor grote communitysites moet kunnen worden gebruikt wordt een systeem van gebruikersgroepen voorzien. Het kan voor bepaalde projecten nuttig zijn om gebruikers in groepen in te delen zodat hun toegangsrechten collectief kunnen worden gewijzigd.

#### Parameters

Waarschijnlijk de meest abstracte klasse. De bedoeling van parameters is om modules configureerbaar te maken door beheerders. Een parameter geeft eigenlijk een standaardinstelling weer. Een gebruiker kan in zijn instellingen afwijken van deze instelling.

Een mogelijke parameter zou bijvoorbeeld SITE\_LANGUAGE kunnen zijn. Deze staat standaard op Engels. Dit betekent dat alle gebruikers die niet ingelogd zijn de site in het Engels te zien zullen krijgen maar als ze zich registeren en in hun instellingen de taal wijzigen dan wordt deze instelling gehanteerd.

Er moet hier duidelijk gesteld worden dat niet alle parameters door gebruikers kunnen beïnvloed worden.

#### UserPreference

Om ervoor te zorgen dat de instelling voor een bepaalde parameter voor een specifieke gebruiker kan worden bijgehouden wordt deze associatieve klasse gebruikt.

#### MenuItem

Om controlcenter bruikbaar te maken zal er een menustructuur moeten aanwezig zijn om de gebruiker een overzicht te geven van de functionaliteit die mogelijk is.

De menustructuur zal echter afhankelijk zijn van de geïnstalleerde modules. Om die reden moet de structuur in de database worden opgeslagen.

#### MenuItemType

Het controlcenter menu zal voor verschillende doelen gebruikt worden. Het is mogelijk dat dit op een community site gebruikt wordt om een aantal gebruikerspecifieke waarden te wijzigen. Mogelijk zal een gebruiker op een forum hier willen aanduiden dat hij een mail moet ontvangen wanneer er geantwoord wordt op zijn post.

De forum administrator zal dit menu gebruiken om forums toe te voegen.

Er zal dus om het overzicht te kunnen bewaren een onderscheid moeten gemaakt worden tussen de verschillende groepen menuitems. Hiervoor dient deze klasse.

#### Menutext

Omdat controlcenter meertaligheid moet ondersteunen kan de menutekst niet worden opgeslagen in het menuitem. Er kunnen door het gebruik van de klasse menutext verschillende klassen gekoppeld worden, voor verschillende talen.

#### Language

Omdat controlcenter meertaligheid ondersteund moet er een klasse aangemaakt worden per ondersteunde taal zodat er later naar verwezen kan worden.

### De Kardinaliteiten

#### User - Language

Een gebruiker kan eventueel een standaardtaal opgeven, maar dit is niet verplicht. Vanzelfsprekend kan er maar 1 taal worden als standaard opgegeven.

#### User-Parameters Via UserPreference

Een gebruiker kan een waarde definieren voor parameters die gedefinieerd zijn. Dit is echter niet verplicht, een parameter zal steeds een standaardwaarde bevatten die gebruikt zal worden wanneer er geen userpreference is.

#### User – Usergroup

Om het beheer van gebruikers gemakkelijker te maken is het zo dat de gebruikers in groepen kunnen worden ingedeeld. Op die manier kan de toegang van grote groepen gebruikers gemakkelijk bijgehouden en aangepast worden. Het is echter mogelijk dat een gebruiker niet tot een groep behoort. In een dergelijk geval kan de toegang per gebruiker worden ingesteld. Gebruikers kunnen lid zijn van verschillende groepen, de toegangsrechten van de groepen waar een gebruiker lid van is worden samengevoegd.

#### User – ModuleAction – via Userlog

Een gebruiker kan acties ondernemen die gelogd worden in dergelijk geval zal er in het script dat de logsequentie moet uitvoeren verwezen worden naar een moduleaction. Dit zal van pas komen voor het filteren van de logbberichten.

De associatieve klasse is nodig om te zorgen dat er voldoende details kunnen worden opgeslagen. Bij het forum kan er bijvoorbeeld een moduleaction ‘Delete User’ zijn, maar in de userlog klasse kan dan bijvoorbeeld een bericht ‘Deleted user Jax007’ van gemaakt worden.

#### Permission – Usergroup

Deze relatie zal bijhouden welke toegang een bepaalde usergroup heeft. Het is mogelijk dat een usergroup geen rechten heeft (al zal dit waarschijnlijk zelden voorkomen)

#### Permission – User

Omdat gebruikers niet noodzakelijk lid moeten zijn van een groep moet er een mogelijkheid voorzien worden om de toegang per gebruiker bij te houden. Hiervoor wordt deze relatie gebruikt.

#### Module –Permission

Iedere module heeft een aantal toegangsopties, voor iedere van deze toegangsopties moet een moduletask worden aangemaakt. Hoe meer er worden gedefinieerd hoe meer detail er in de toegang kan geprogrammeerd worden.

Het is zo dat er ook Permissions kunnen zijn zonder modules, deze kunnen echter enkel voor CORE gedefinieerd worden.

#### Module – Moduleaction

Een module zal vaak verschillende functionaliteiten hebben waarvan we de uitvoering willen bijhouden. Om ervoor te zorgen dat er geen kluwen van logberichten ontstaat worden de logberichten in verschillende moduleactions ingedeeld. Zo zal het opzoeken van een bepaalde actie makkelijker verlopen.

#### Module – Menuitem

Een module zal vaak een stuk backend functionaliteit bevatten die vanuit het controlcenter hoofdmenu moet toegankelijk zijn. Er zal daarom een menu moeten voorzien worden waar iedere module items aan kan toevoegen, een module zal zijn items opslaan in menuitems. Het is mogelijk dat er geen items zijn of meerdere items.

Een menuitem moet niet noodzakelijk aan een module gekoppeld worden, er zullen er een aantal zijn die geen koppeling hebben.

#### Module – Parameters

Om ervoor te zorgen dat modules breed bruikbaar zijn zullen er configuratieopties voorzien worden die bepalen hoe een bepaalde module werkt. Veel van deze parameters zullen tot een bepaalde module behoren. Een parameter hoeft echter niet altijd tot een module te behoren. Er zullen reeds een aantal parameters aanwezig zijn wanneer controlcenter2 geïnstalleerd is.

#### MenuItem – MenuText

Omdat er meertaligheid voorzien wordt zullen er meerdere teksten in verschillende talen gekoppeld worden aan een menuitem. Er zal altijd voor iedere aanwezige taal altijd 1 menutext moeten gekoppeld worden aan een menuitem. Een menutext hoort altijd tot strikt 1 menuitem

#### Menutext – Language

Om te kunnen definiëren bij welke taal een menutext hoort wordt deze relatie gebruikt. Een menutext zal altijd tot 1 taal behoren, iedere taal kan (en zal meestal) meerdere menuteksten hebben.

# Kernprocessen

Voor het ontwerpen kan beginnen moet vastgelegd worden welke functionaliteit controlcenter nu precies moet kunnen vervullen

* Verbinding maken met verschillende databasesystemen
* Door gebruik van templates de hoeveelheid HTML die moet worden geschreven drastisch inperken
* Gebruikers beheren
* Pagina’s beveiligen
* Modules toevoegen
* Acties van gebuikers loggen
* Door gebruik van parameters verschillende configuratiemogelijkheden aanbieden
* Een menu voorzien waarmee de functionaliteit gemakkelijk kan worden gebruikt.
* Meertaligheid aanbieden
* Mogelijkheid bieden tot opbouwen van sitemap

# Uitwerking

Om de ontwikkeling in de hand te kunnen houden wordt controlcenter2 incrementeel ontwikkeld via volgende incrementen

* Databaseverbinding
* Templatesystem
* Gebruikersbeheer
* Login
* Beveiliging
* Logging
* User Preferences
* Modules Installeren

## Exception handling

PHP5 is de eerste versie van PHP die exception handling ondersteund, al is het nog ruw…

Het is mogelijk om in functies en klassen exceptions the throwen als er iets misloopt. Die exceptions kunnen dan opgevangen worden in het script dat de functie aanroept.

Controlcenter2 maakt gebruik van de optie om eigen Exception klassen te definiëren. Het onderscheid met de normaal voorziene functionaliteit is dat de CC2Exception een boodschap heeft die weergegeven kan worden, maar ook een uitgebreidere foutmelding die in een later stadium gelogd kan worden.

Door nu een custom class te implementeren kan de exception handling later worden uitgebreid.

## Beveiligde bestanden

Omdat bepaalde gegevens variabel moeten zijn maar niet publiek mogen worden moet er een mogelijkheid voorzien worden om files op te slaan buiten de public\_html directory. Het pad naar deze file moet manueel ingesteld worden. Om die reden wordt er een file voorzien waarin de variabele $pathtoconfig wordt gedefinieerd. Deze variabele zal bij de installatie van Controlcenter moeten aangepast worden.

## Increment1: Databaseverbinding

Om ervoor te zorgen dat controlcenter compatibel kan zijn met verschillende databasesystemen moet er een extra laag ontwikkeld worden in de databasecommunicatie .

Om dit te kunnen toepassing zullen er voor ieder ondersteund databasesysteem DataAccess Components worden ontwikkeld.

Het idee is om de structuur van die componenten gelijk te maken zodat er afhankelijk van het databasesysteem een andere functiedefinitie kan worden geopend. Op die manier zullen bovenliggende lagen niet meer afhankelijk zijn van het gebruikte databasesysteem.

De problemen die moeten opgelost worden in dit increment zijn de volgende:

* Bepalen van de structuur van de DataAccess Components
  + Verbinding maken
  + Beveiliging
  + Queries uitvoeren + teruggave naar aanroepend script
    - Array met resultaten
    - Integer met aantal resultaten
    - …
  + NonQueries uitvoeren
  + Exception Handling
* Bouwen van een component voor MySQL
* Zorgen dat de oplossing connection pooling kan gebruiken
* Het data component selectiemechanisme

### Beveiliging

Het spreekt vanzelf dat de connectiegegevens voor de database niet in de database zelf kunnen worden opgeslagen. Het opslaan van deze gegevens in een bestand op de server houdt echter risico’s in. Bij controlcenter2 worden deze risico’s op 2 manieren ingeperkt:

* De verbindingsgegevens worden opgeslagen in een php-file, deze kunnen niet geopend worden in een browser omdat de webserver dit verhindert
* De verbindingsgegevens worden boven de public\_html directory opgeslagen waardoor ze in principe onbereikbaar worden van buitenaf.
* Er wordt geen gebruik gemaakt van constants zoals in controlcenter 1.20 want deze zijn in gans het scripts bereikbaar, dus ook op de presentation layer. Door gebruik te maken van een variabele kan deze binnen een object gehouden worden.
* Er wordt een mechanisme voorzien tegen SQL Injection

Dit neemt niet weg dat een hacker die FTP toegang kan verkrijgen toch de gegevens kan inlezen.

### Syntax van de configuratiefile

In de configuratiefile worden alle parameters bepaald die nodig zijn voor een verbinding met de database

* Databasetype
* Serveradres
* Database
* Gebruikersnaam
* Wachtwoord

<?php

##CONTROLCENTER2  
##DATABASE CONNECTION PARAMETERS

###DATABASE SYSTEM  
###Supported systems: MySQL

$\_DATABASE['type'] = "MySQL";

###CONNECTION DATA

$\_DATABASE['host'] = "localhost";  
$\_DATABASE['database'] = "controlcenter2";

$\_DATABASE['user'] = "test";  
$\_DATABASE['password'] = "test";

?>

Codevoorbeeld 2: Syntax van de database configuratiefile

### Datacomponent selector

Dit mechanisme moet ervoor zorgen dat het juiste component wordt ingeladen. Het is systeem is vrij simpel want het hangt volledig af van de waarde die opgegeven is bij $\_DATABASE[‘type’]

Het script moet inlezen welk type de database is en zal dan gaan zoeken in de directory /core/data/components/$\_DATABASE[‘type’]

Daar zal dan een file component.php geincluded worden.

### Data Componenten

De structuur van een datacomponent ziet er als volgt uit



Figuur 3: De klasse 'DataComponent'

De connectiegegevens worden als privates binnen het object gehouden om veiligheidsredenen. De verbinding wordt van binnen deze klasse geopend en ook het uitvoeren van de queries en het teruggeven van resultaten gebeurt in de klasse zelf.

In broncode ziet dit er als volgt uit

<?php  
class DataConnection  
{  
#De connectieparameters moeten beveiligd binnen de instantie blijven  
Private $parameters;  
 Function \_\_construct()  
 {  
 #Ophalen van de verbindingsgegevens  
 include "fileaccess/database.php";  
 $parameters = GetDataBaseParameters.php();  
 }

Private function Connect()  
 {

}

Public function setQuery($query)

{

}

Public function setAttribute($attributename,$value)

{

}

Public function ExecuteQuery()  
 {

}

Public function GetResultArray()  
 {

}

Public function GetScalar()  
 {

}

Public function GetNumRows()  
 {

}

Public function GetLastId()

{

}

}  
?>

Codevoorbeeld 3: DataComponent skeleton

De constructfunctie maakt gebruik van de functies in de fileaccess layer om de configuratiefile in te lezen.

Schematisch ziet de werking van een datacomponent er als volgt uit:



Figuur 4: Datacomponent constructor

#### setQuery($query)

Deze functie geeft de instantie een query om uit te voeren. De query wordt echter niet uitgevoerd door dit commando

#### getNumRows()

De functie getNumRows() geeft het aantal rijen terug dat aan de query voldoet

#### getResultArray()

De functie geeft een multidimensionale array terug met de resultaten van de query. Wanneer er geen resultaten zijn wordt false teruggegeven.

De resultaten kunnen dan verder verwerkt worden

foreach($result as $key=>$value)

{

list($id,$username) = $result[$key];

echo $id .",".$username;

}

Codevoorbeeld 4: Verwerken van de resultarray

#### getScalar()

Wanneer er maar 1 veld in het resultaat zit is het gemakkelijker en sneller om enkel dat ene veld op te vragen.

getScalar() geeft dit dan terug in de vorm van een variabele. Wanneer het resultaat meer dan 1 veld bevat wordt een exception teruggegeven.

#### getLastId()

Bij het toevoegen van records is het erg interessant om te weten op welk id een record werd toegevoegd. Deze functie geeft dit id terug.

#### Uitvoeren van een query

Om gebruik te maken van alle functionaliteit wat exception handling betreft maakt men best gebruik van volgende manier om queries uit te voeren:

Try  
{  
Require\_once $\_SERVER[‘DOCUMENT\_ROOT’].”/core/dataconnection/componentselector.php”;

$db = new dataconnection();  
$query = “Select \* from test”;  
$db->setQuery($query);  
$db->ExecuteQuery();  
$result = $db->getResultArray();  
}  
Catch(CC2Exception $ex)  
{  
#Exceptionhandling code…  
}

Codevoorbeeld 5: Uitvoeren van een query met datacomponents

#### MySQL Data Component

Voorlopig zal enkel MySQL ondersteund worden. Het datacomponent voor MySQL zal als richtlijn dienen voor het onwikkelen van bijkomende datacomponents.

### Beveiliging tegen SQL Injection

Om ervoor te zorgen dat SQL injection onmogelijk wordt zal een systeem gecreëerd worden zoals dat in .NET wordt gebruikt. De gegevens in die in variabelen worden aangeleverd worden gefilterd voordat de query wordt toegepast.

Dit vereist een licht gewijzigde werking. Een query die beveiligd is zal er als volgt uitzien:

SELECT \* FROM users WHERE users.id=@id

De @ duidt op een attribuut die opgegeven zal worden. Deze zal met de functie $this->setAttribute ingevuld worden en gefilterd worden. Concreet ziet de veilige toepassing er als volgt uit:

<?php

$db = new dataconnection();

$query = “SELECT \* FROM users WHERE users.id=@userid

$db->setQuery($query);

$db->setAttribute(“userid”,$userid);

$db->ExecuteQuery();

?>

codevoorbeeld 6: Uitvoeren van een query zonder risico op SQL injection

Wanneeer er geprobeerd wordt om een attributen in te vullen die niet in de Query voorkomt zal dit een exception opleveren. Wanneer niet alle attributen werden gedefinieerd zal dit eveneens een Exception opleveren.

### Samenvatting

Na het ontwikkelen van het eerste increment is er een interface die ervoor zorgt dat het dataverkeer op een gemakkelijke en abstracte manier zal kunnen verlopen.

## Increment2: Parameters

Nu we een werkende databaseverbinding hebben moet een systeem ontwikkeld worden om parameters te gebruiken. Het idee achter parameters is dat het een systeem is dat ze controlcenter configureerbaar zal maken. Bepaalde van die parameters zullen kunnen overschreven worden door de gebruiker.

Eerst en vooral zal er een klasse ‘parameter’ moeten worden gedefinieerd die gebruikt zal worden bij het werken met parameters.

Kenmerken van een parameter zijn:

* Naam van de parameter: hier verwezen naar de instelling die de parameter vertegenwoordigt. Om te zorgen dat de lijst van parameters wat overzichtelijker wordt krijgen de parameters een prefix. Alle parameters die met de core-elementen te maken hebben zullen de prefix ‘CORE\_’ krijgen.
* Waarde: De waarde van een parameter zal als string opgeslagen worden om maximale mogelijkheden te garanderen.
* Overridable: Deze waarde bepaalt of een parameter kan overschreven worden door een voorkeur van een gebruiker. Deze functionaliteit zal later pas geïmplementeerd kunnen worden wanneer het gebruikersbeheer ontwikkeld is. Er wordt nu echter al ruimte voorzien ervoor.



Figuur 5: Klasse 'parameter'

De klasse parameter zal aangemaakt worden zoals in bovenstaand schema. Merk op dat het id niet gewijzigd kan worden van buitenaf. Dit is een veiligheidsoverweging om te zorgen dat er geen fouten gebeuren bij het werken met parameters.

In code ziet dit er als volgt uit:

Increment1 zorgde voor een stabiele databaseverbinding en deze zal nu worden aangewend. De parameters worden opgeslagen in de database



Figuur 6: De databasetabel 'parameters'

De SQL-code voor het aanmaken van deze tabel:

CREATE TABLE parameters(

id INT(3) UNSIGNED ZEROFILL NOT NULL AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(20) NOT NULL,

value VARCHAR(20) NOT NULL,

overridable BIT NOT NULL);

CREATE UNIQUE INDEX search\_by\_name on parameters(name(20));

Codevoorbeeld 7: SQL-code voor declaratie van de parameters-tabel

Er wordt een index aangemaakt op name omdat dit criterium veel gebruikt zal worden. Name wordt niet als primary key gedefinieerd omdat het geen nummer is.

### De code

Bij dit increment wordt eigenlijk vooral aandacht besteed aan de DataAccess code van de parameters. De code in andere lagen zal vaak verschillend bij de verschillende parameters en zal dus in volgende incrementen worden uitgewerkt.

Bovenstaande code is de code die in de DataAccess laag wordt gebruikt voor het ophalen, wijzigen en toevoegen van parameters. Alle functies werken waar mogelijk met de parameter-klasse.

### Samenvatting

Aan het einde van dit increment is er een manier voor definiëren, aanpassen en toevoegen van parameters. Deze functionaliteit zal nog verder moeten worden uitgebreid eenmaal het gebruikersbeheer geïmplementeerd is.

De parameterfunctie zal een aantal belangrijke toepassingen krijgen:

* Bepalen of de database en bestandsversie van controlcenter overeenkomst => update nodig?
* Standaardtaal van de site die door controlcenter aangedreven wordt
* …

## Increment3: Templatesystem

Een goed werkende site engine moet natuurlijk ook kunnen gebruik maken van templates om snel HTML-pagina’s te kunnen aanmaken. Controlcenter2 template system zal gebaseerd zijn op dezelfde principes van hetgeen werd gebruikt in Controlcenter1.x

De functionaliteit wordt echter gevoelig uitgebreid. Volgende aspecten moeten aangehaald worden:

* Inlezen van templates
* Weergeven van templates
* Aanwenden van templates in presentation layer
* Meertaligheid
* Mogelijkheid om de site te disablen
* Importeren van variabelen in templates
* Importeren van parameters in templates
* Importeren van bestanden in templates
* Implementatie van Aliases
* Implementatie van addins

### Structuur van het templatesystem

Om ervoor te zorgen dat het systeem dat moet worden ontwikkeld met al zijn functionaliteit zo bruikbaar mogelijk blijft zal er een klasse ‘htmlpage’ worden gemaakt. Dit is de klasse die alle templatefunctionaliteit bevat en die dus op de presentationlayer zal aangeroepen worden.

### Template aliases

Eén van de grote problemen met het templatesysteem van Controlcenter1.x is dat er onvoldoende flexibel gewisseld kon worden met templates. De reden hiervoor was dat er in de code verwezen werd naar een directory op de server. Dit had tot gevolg dat de bestanden steeds in die directory moesten geplaatst worden.

**Voorbeeld:**Stel dat een site bijvoorbeeld gebruik maakt van een template ‘main’. De beheerders van de website willen voor de feestdagen een lichte aanpassing doen aan de site om die wat feestelijker te maken. Bij controlcenter 1 betekende dit dat de HTML-code in de directory templates/main moest vervangen worden door nieuwe broncode. Wanneer men dan terug naar het oude template wilde moest de oude HTML-code weer naar templates/main worden gekopieerd. Dit zorgt ervoor dat het overschakelen naar een nieuwe look veel tijd in beslag neemt en ook risico’s met zich meebrengt. Stel dat na het opladen blijkt dat de site niet correct werkt met het nieuwe template dan moet in allerijl het oude template weer worden opgeladen.

Er is dus nood aan een manier om vlotter te kunnen schakelen tussen templates. Om deze vraag te beantwoorden wordt een systeem van templatealiases geïntroduceerd vanaf controlcenter 2. Het idee achter templatealiases is dat deze, via de database, de koppeling maken met een directory op de webserver.

Als we teruggaan naar ons voorbeeld zou dit concreet betekenen dat er in de code verwezen wordt naar een templatealias ‘main’. In de database bestaat er een tabel templatealiases



Figuur 7: de databasetabel templatealiases

Het templatesysteem zoekt automatisch op welke directory er gekoppeld is aan main en maakt gebruik van het template dat in die directory is opgeslagen. In een dergelijk geval kunnen er bijvoorbeeld 2 templates bestaan in /templates, namelijk ‘xmas’ en ‘regular’. Door in de databasetabel de waarde bij directory te veranderen van ‘regular’ in ‘xmas’ kan er een omschakeling tussen de verschillende templates gebeuren. Als dan blijkt dat er problemen zijn dan kan er op dezelfde manier terug overgeschakeld worden naar het oorspronkelijke template.

### Template Addins

Anders dan in vorige versies van controlcenter zal er bij het gebruik van het templatesysteem nadruk gelegd worden op een verdere splitsing tussen PHP-code en HTML-code. Om ervoor te zorgen dat het niet langer noodzakelijk is om HTML-code en PHP-code te vermengen in één file wordt er een systeem van templateaddins geïntroduceerd.

Een templateaddin is een stuk HTML-code dat door het templatesysteem kan ingelezen worden en toegevoegd aan de HTML-code. Kenmerkend voor addins is dat ze variabelen bevatten die door het templatesystem automatisch worden vervangen door variabelen in die door de PHP-code zijn gedeclareerd. Onderstaand voorbeeld brengt meer verduidelijking:

**Voorbeeld:**We willen een pagina weergeven waarop een template wordt toegepast en waarop de melding “Welkom $gebruikersnaam” wordt weergegeven. De verdeling tussen de verschillende componenten zien er in de nieuwe indeling als volgt uit:

* PHP-code: Deze code zal de gebruikersnaam moeten ophalen (evt via bovenliggende layers) en toewijzen aan een variabele
* Template addin: Hier wordt de HTML-code opgeslagen die gebruikt wordt voor de weergave van de gegevens
* Template: Hier wordt de HTML opgeslagen die voor alle pagina’s gebruikt wordt

PHP-code:

<?php

…

//Onderstaande functie haalt via Logic en DataAccess layers degebruikersnaam op  
$gebruikersnaam = getUserName();

//Het templatesystem wordt aangeroepen, er zal gebruik gemaakt worden van het template dat aan de alias ‘controlcenter’ gekoppeld is.  
$html = new htmlpage(“controlcenter”);

//De templateaddin Showuser wordt aangeroepen  
$html->LoadAddin(“addins/showuser.tpa”);

//De variabele <[USERNAME]> Die in showuser.tpa wordt gedeclareerd krijgt nu door het templatesysteem de waarde van $gebruikersnaam toegewezen.  
$html->setVariable (“username”,$gebruikersnaam);

//Nu is de HTML-code volledig en kan deze afgedrukt worden op het scherm van de gebruiker  
$html->PrintHMTL();

?>

Codevoorbeeld 8: Het gebruik van het templatesysteem in combinatie met addins

Showuser.tpa:

<h1>Welcome, <[USERNAME]></h1>

<h3>We hope to see you again soon!!</h3>

Codevoorbeeld 9: Voorbeeld van een simpele template addin

Template code:

<html>

<head>

<title>Contolcenter 2</title>

<!—PageContent-->

</head>

</html>

Na verwerking door het templatesysteem zal de HTML-code die geproduceerd is er als volgt uit zien:

<html>

<head>

<title>Contolcenter 2</title>

<h1>Welcome, DragonEye</h1>

<h3>We hope to see you again soon!!</h3>

</head>

</html>

Alhoewel deze manier van werken ervoor zorgt dat de code over verschillende files wordt verspreid zal dit ook tot gevolg hebben dat de code gemakkelijker onderhouden kan worden. De verschillende componenten van de presentation layer worden immer gescheiden.

### Implementatie van meertaligheid

Reeds van bij het begin werd de eis gesteld dat controlcenter2 sites moet kunnen aandrijven in verschillende talen waarbij gebruikers kunnen schakelen tussen verschillende talen en waarbij ook alle menu’s voor beheerders in verschillende talen beschikbaar zijn.

Aangezien het templatesysteem voor iedere weergave zal worden gebruikt is het logisch dat de meertaligheid op dit niveau moet worden geïmplementeerd.

Er zijn verschillende aspecten waarmee rekening gehouden moet worden:

Eerst en vooral zullen er verschillen zijn in de templates. Wanneer men bijvoorbeeld een knop ontwerpt waarop de tekst ‘Register Now!!’ staat en die wordt in jpeg formaat gebrukt in het template dan zal dit template niet gebruikt kunnen worden voor een Nederlandstalige versie van dezelfde site. In een dergelijk geval zullen er 2 template gemaakt worden, één met de ‘Register Now!!’ op de knop en één met ‘Registreer Nu!!’ op de knop. Dit betekent dat het templatesysteem dus rekening zal moeten houden met de taal van de gebruiker wanneer een template wordt geopend.

In de praktijk zal de taal waarin de site standaard wordt weergegeven in de parameter CORE\_LANGUAGE worden opgeslagen. De waarde van CORE\_LANGUAGE zal dan het id zijn van een taal die in de tabel languages is gedefinieerd is. CORE\_LANGUAGE zal mogelijk overridable zijn waardoor gebruikers deze waarde kunnen overschrijven met hun eigen geselecteerde taal.

Het overschrijven van parameters met eigen voorkeuren is materie voor de volgende incrementen , nu gaan we ervanuit dat CORE\_LANGUAGE de juiste waarde bevat. De selectie van de templatefile zal als volgt gebeueren:



Figuur 8: Selectie van template.htm

Een tweede aspect van meertaligheid is de tekst die in de HTML-code voorkomt. Deze zal natuurlijk ook moeten vertaald worden. Een probleem hier is echter dat we niet weten hoeveel talen en welke talen er zullen ondersteund worden, dat wordt door de beheerders van de site beslist. Ook wat modules betreft weten we niet welke er al dan niet geïnstalleerd worden.

Om toch meertaligheid in tekst te kunnen aanbieden onder veranderlijke omstandigheden wordt er geopteerd om gebruik te maken van language files. In deze language files worden variabelen gedeclareerd die verwijzen naar stukken tekst in een bepaalde taal.

In de file Dutch.language.php zal \_LANG\_WRONGLOGIN bijvoorbeeld de tekst “De gebruikersnaam en/of wachtwoord zijn niet correct”. Dezelfde variabele zal in English.language.php de waarde ‘Your username and/or password are incorrect”. In de code kan er dan verwezen worden naar de constante \_LANG\_WRONGLOGIN, ongeacht de taal.

Het taalsysteem zal modulair moeten zijn om ervoor te zorgen dat iedere module zijn eigen languagefiles kan toevoegen aan het systeem. Wanneer niet telkens alle taalconstanten moeten worden ingeladen zal dit ook voordelen hebben wat performantie betreft.

Concreet zullen er voor ieder component (gebruikersbeheer, logging, etc.) taalfiles moeten worden ontwikkeld. Voor de meer algemene teksten zal er voor iedere taal een algemen taalfile worden voorzien die altijd wordt ingeladen.

De bestanden zullen als volgt worden opgeslagen:

* De algemene taalfiles zullen worden opgeslagen onder /core/languages/$taal.language.php
* De taalfiles voor de componenten die tot Controlcenter CORE behoren zullen worden opgeslagen in /core/presentation/gebruikersbeheer/languages/$taal.language.php
* De taalfiles voor de modules zullen worden opgeslagen onder /modules/modulenaam/languages/$taal.language.php

Het templatesysteem zal aan de hand van CORE\_LANGUAGE de juiste language file zoeken in /core/languages en deze importeren.

Voor het importeren van de andere files zal een iets complexere manier van werken moeten worden gebruikt. We moet immers niet altijd alle files importeren. Voor het opzoeken van de juiste templatefiles zal er gebruik worden gemaakt van de url van het aanroepende script. Stel bijvoorbeeld dat een gebruiker een nieuwe gebruikersaccount wil aanmaken. Hij zal dan bijvoorbeeld <http://www.dragoneyehosting.be/core/presentation/usermanagement/adduser.php> openen. Door deze url te analyseren kunnen we te weten komen welke languagefile er geopend moet worden. Het zal de languagefile zijn die behoord bij het CORE component usermanagement. De languagefiles voor de componenten worden opgeslagen in een subdirectory languages. De languagefile bevindt zich dus onder /core/presentation/usermanagement/languages/.

Concreet zal volgende werkwijze gebruikt worden:



Figuur 9: Selectie van languagefiles

### Structuur van templatefiles

De templatefiles zullen aan een bepaalde structuur moeten beantwoord om ervoor te zorgen dat het templatesysteem ze correct kan verwerken. De bedoeling is natuurlijk om zo weinig mogelijk beperkingen op te leggen.

Een template wordt opgeslagen onder de directory /templates/templatenaam, deze directory bevat alle files die nodig zijn om het template te gebruiken.

De belangrijkste file is degene die de HTML-code bevat. Dit is immers de code die door het templatesysteem zal toegevoegd/uitgebreid worden aan de pagina die de gebruiker op zijn scherm krijgt.

De templatefile bevat ENKEL HTML. Bovenop de HTML-code zijn er een aantal custom tags die door het templatesysteem geïnterpreteerd en verwerkt kunnen worden. Een basistemplate ziet er als volgt uit:

<html>

<head>

<title>Contolcenter 2</title>

<!—PageContent-->

</head>

</html>

Codevoorbeeld 10: Een simpele templatefile

In bovenstaand voorbeeld valt onmiddellijk de custom tag <!—PageContent--> op. Dit is de belangrijkste custom tag die door het templatesystem wordt gebruikt. Dit is namelijk het punt waar het templatesysteem inhoud kan toevoegen.

Het templatesystem zal de file bij verwerking zodanig aanpassen dat de addins die worden ingeladen vanuit het aanroepende script op deze plaats ingevoegd worden.

### Variabelen

Het is mogelijk dat er in een template of in een addin variabelen voorkomen die door het aanroepende script moeten worden ingevuld.

Aangezien we gebruik maken van een klasse moeten deze variabelen binnen de instantie aangemaakt worden.

Voor de aanmaak van variabelen wordt de functie $this->setVariable($varname,$value) gebruikt.

Binnen de klasse is er een array $variables, voor iedere variabele wordt een onderdeel aan deze array toegevoegd zodat dit gebruikt kan worden.

### Custom Tags

Om ervoor te zorgen dat het mogelijk is om het templatesysteem te gebruiken met de addins zullen er custom tags moeten worden gedefinieerd zodat er voldoende functionaliteit ontstaat

Dit is de tweede herziening van het systeem van het systeem van de custom tags dat nieuw wordt ingevoerd vanaf controlcenter2. Het probleem bestaat erin dat er gebruik zal moeten worden gemaakt van PCRE functies voor het analyseren van patronen in de HTML-code van zowel templates als addins. Omdat deze PCRE functies zeer complex en dus ook veeleisend zijn moeten ze tot een minimum beperkt worden.

Om een minimumaantal analyzes te kunnen aanwenden zullen alle tags volgens een welbepaald patroon moeten worden gedefinieerd:

<!cc *tagnaam* *parameter>…[<!*end cc *tagnaam*>]

Er moeten structuren geprogrammeerd worden die enkel een starttag hebben maar ook structuren met start-en eindtags. Er zal dus moeten rekening gehouden worden met het feit dat er eventueel geneste custom tags mogelijk zullen zijn.

De verwerking zal als volgt moeten verlopen:



Figuur 10: Verwerking van custom tags

### Overzicht van de custom tags

#### <!CC Addin ‘padnaarfile.tpa’>

Deze tag staat voor een addin die moet geïmporteerd worden in de HTML-code. Er wordt in de tag verwezen naar een pad. Dit pad begint vanaf $\_SERVER[‘DOCUMENT\_ROOT’].

#### <!CC LOOP [variabele]>…<!CC END LOOP [variabele]>

De LOOP tag zorgt dat bepaalde HTML-code kan herhaald worden aan de hand van een array die vanuit het aanroepende script wordt gedefinieerd.

Binnen de <!CC LOOP><!CC END LOOP> tags zullen er loopvariabelen worden gedefinieerd met behulp van de constructie <!CC LVAR [varname]>.

Volgend voorbeeld maakt de werking van de loops duidelijk:

*Aanroepend script:*

<?php

$html = new htmlpage(“test”);

$userlist[0][‘username’] = “user01”;

$userlist[0][‘country’]=”USA”;

$userlist[1][‘username’]=”user02”;

Userlist[1][‘country’] = “Belgium”;

$html->setVariable(“userarray”,$userlist);

$html->PrintHTML();

?>

*Template test:*

<HTML>

<HEAD></HEAD>

<BODY>

<CENTER>Userlist</CENTER>

<TABLE>

<!CC LOOP [userarray]>

<TR><TD><!CC LVAR [username]></TD><TD><!CC LVAR[username]></TD></TR>

<!CC END LOOP [userarray]>

</TABLE>

</BODY>

</HTML>

Bij het verwerken van deze code zal het templatesysteem voor ieder item in de array $userlist een lijn <TR><TD>… aanmaken en de waarden op $userlist[i][‘username’] en $userlist[i][‘country’] invullen.

#### <!CC IFSET [variabele]><!CC END IFSET [variabele]>

De IFSET tag zorgt ervoor dat html niet wordt afgedrukt als er geen waarde is toegewezen aan de variabele in de parameter.

<HTML>

…

<BODY>

<!CC IFSET [test]>

Deze tekst wordt niet afgedrukt als test geen waarde heeft..

<!CC END IFSET [test]>

</BODY>

</HTML>

#### <!CC VAR [variabele]>

De VAR tag zorgt voor het invoeren van variabelen die vanuit het aanroepende script kunnen worden gedeclareerd.

*Voorbeeld:*

*Aanroepend script:*

<?php

$html = new htmlpage(“test”);

$html->setVariable(“testvar”,”waarde1”);

$html->PrintHTML();

?>

*Template:*

<HTML>

…

<BODY>

<CENTER><H1><!CC VAR [testvar]></CENTER>

</BODY>

</HTML>

In bovenstaand geval zal de <!CC… tag vervangen worden door de waarde “waarde1”. Stel dat er bij het uitvoeren van PrintHTML() geen waarde werd opgegeven voor testvar dan zal dit geen exception opleveren.

#### <!CC LANG [langconstant]>

De LANG-tag zal de tekstinvoegen die aan de constant is gekoppeld volgens de taal die op dat moment van toepassing is. Deze tag zorgt ervoor dat meertaligheid in templates geen problemen oplevert.

Stel dat er op het moment van uitvoeren 2 talen zijn gedefinieerd:

* 1: English
* 2: Nederlands

De gebruiker is niet ingelogd => CORE\_LANG zal bepalen welke taal er gebruikt wordt.

<HTML>

….

<BODY>

<H1><!CC LANG [CORE\_WELCOME]></H1>

</BODY>

</HTML>

Deze HTMLcode zal een Nederlandstalige welkomtekst opleveren wanner CORE\_LANG=2 maar een engelstalige tekst wanneer CORE\_LANG=1.

### De Klasse HTMLPage

De klasse HTMLPage vormt de kern van het templatesysteem. Deze klasse zal gebruikt worden in de aanroepende scripts om het templatesysteem te activeren.

De structuur is de volgende



Figuur 11: De klasse HTMLPage

Bij de constructie van een instantie wordt een alias meegegeven die dan zal bepalen welk template er gebruikt zal worden. Er kunnen variabelen gedeclareerd worden en addins geladen waarin dan variabelen zijn opgenomen.

### Templatesystem Revisie1

Bij het ontwikkelen van increment5 werd vastgesteld dat de aangeboden functionaliteit met betrekking tot lussen ontoereikend is.

Het probleem stelt zich bij multidimensionale arrays. Concreet willen we het volgende doen:

We definieren een array met volgende waarden:

$permissionlist[0]

$permissionlist[0][modulename] = “Forum”

$permissionlist[0][moduletasks][0][name] = “Add forums”

$permissionlist[0][moduletasks][0][id] = 1

$permissionlist[0][moduletasks][1][name] = “Edit posts”

$permissionlist[0][moduletasks][1][id] = 2

We willen hieruit volgende HTML genereren met behulp van het templatesysteem:

<TABLE>

<TR><TD COLSPAN=2>Forum</TD></TR>  
<TR><TD>1</TD><TD>Add forums</TD></TR>  
<TR><TD>2</TD><TD>Edit posts</TD></TR>

</TABLE>

Hier hebben we dus eigenlijk een Lus binnen een lus. Op zich geen probleem, ware het niet dat er hier een verband is tussen de 2 lussen: ze zitten in dezelfde array.

De huidige manier van verwerken kan hier geen oplossing bieden: De loops worden verwerkt in de second parse. Er wordt gezocht naar het patroon <!CC NAAM [PARAMETER]><!CC END NAAM [PARAMETER]>. Wanner het patroon ontdekt wordt wordt er afhankelijk van de waarde van NAAM een bepaalde functie uitgevoerd. Het zoeken naar het patroon gaat verder tot er geen matches meer zijn. Dit systeem werkt zolang er geen verband is tussen de verschillende lussen. Op dit moment wordt iedere gedetecteerde lus apart behandeld.

Een eventuele oplossing schuilt erin dat we op zoek gaan daar de lussen die geen parent lus hebben. Voor iedere dergelijke lus wordt er code uitgevoerd die ook de geneste lussen behandelt. Er zal wel een manier nodig zijn om het verband tussen de lussen aan te duiden en voor iedere LVAR zal duidelijk moeten zijn tot welke lus die behoort. Concreet wordt volgende oplossing voorgesteld:

<!CC LOOP [parent.child]><!CC END LOOP [parent.child]>

<!CC LVAR [parent.child.variable]>

Om dit systeem te doen werken moeten ere en aantal aanpassingen gebeuren aan de bestaande code:

* Het verschil tussen een lus op het eerste niveau en een geneste lus is dat er een punt zal voorkomen in de naam van de variabele => **In de secondparse wordt er een aanpassing gedaan aan het patroon zodat er geen matches meer gebeuren op loops met een punt in de variabelenaam.**
* Het patroon wordt UNGREEDY => dit wil zeggen dat er niet meer geprobeerd wordt om zo weinig mogelijk hits te genereren maar er wordt indien er keuze is geopteerd voor meerdere hits van kleinere patronen.
* Om ervoor te zorgen dat er een onbeperkt aantal niveaus mogelijk is zal er nood zijn aan een verwijzing naar de functie binnen de definitie van de functie?
* Aanpassingen van de structuur van ParseLoop()

### Samenvatting

Het derde increment vormt het startpunt van het templatesysteem. Op dit moment werd enkel de hoogstnoodzakelijke functionaliteit ontwikkeld.

De structuur van de code is zodanig dat er nog extra custom tags kunnen worden toegevoegd wanneer dit nodig blijkt. Nu reeds staat vast dat er verder in het ontwikkelingsproces een revisie zal moeten gebeuren van het templatesysteem om ook functies in verband met toegang van de actieve gebruiker te regelen. Deze functionaliteit kan op dit moment echter nog niet geïmplementeerd worden omdat het systeem van gebruikerstoegang nog niet bestaat.

Ook zal moeten blijken of de manier van werken zoals in controlcenter 1.x (waarbij de HTML in het aanroepend script werd opgenomen) zich opdringt. Bij voorkeur zal er gebruik gemaakt worden van addins om te zorgen dat de HTML gescheiden wordt van de PHP code.

## Increment4: Aanmaken/wijzigen van gebruikers

### Inleiding

Nu de parameters, de databaseverbinding en het templatesysteem voldoende ontwikkeld zijn kan gestart worden met de ontwikkeling van het eerste CORE component.

Het gebruikersbeheer vormt de kern van de Controlcenter-functionaliteit en er zal dan ook veel aandacht aan worden besteed.

Zoals in de inleiding gesteld moet controlcenter2 inzetbaar zijn voor sites waar gebruikers een eigen account kunnen krijgen als voor sites waar accounts enkel voor beheerders zijn.

Er zal om die reden met 2 scripts gewerkt worden die instaan voor de registratie:

* Intregistration zal instaan voor de registratie van gebruikers vanuit het backend deel. Voor het gebruik van dit deel zal een gebruiker de nodige toegang moeten krijgen van een beheerder. Bij het aanmaken van een gebruiker met internal registration kan de gebruiker kiezen tot welke gebruikersgroep de nieuwe account behoort
* Extregistration zal instaan voor de registratie van gebruikers die nog geen toegang hebben tot het backend. Concreet zal dit script gebruikt worden voor gebruikers die zichzelf registreren. Externe registratie moet geactiveerd worden door een beheerder. Controlcenter zal de parameter CORE\_USER\_EXT\_REGISTRATION gebruiken als indicator om te weten of externe registratie is toegestaan. Administrators zullen dus in de configuratie van controlcenter kunnen kiezen welk systeem van registratie zij wensen te gebruiken. Bij externe registratie kan de gebruiker niet kiezen onder welke gebruikersgroep hij terecht komt. De administrator zal de parameter CORE\_USER\_EXT\_USERGROUP de waarde geven van de gebruikersgroep waar nieuwe gebruikers aan toegewezen worden. Bij externe registratie zullen ook een aantal veiligheidsmaatregelen moeten worden in acht genomen.   
    
  Zo zal een beheerder met behulp van parameter CORE\_USER\_CONFIRMATION\_REQ kunnen bepalen of een gebruiker zijn activatie moet bevestigen. De activatie heeft 2 stappen: eerst wordt er een mail gestuurd naar de gebruiker met een activatielink, op deze link gebeurt dan nog een visuele bevestiging.

In extreme gevallen zal er nadat de gebruiker zijn account heeft bevestigd ook nog een manuele controlegebeuren van de gegevens door een administrator alvorens de account bruikbaar wordt. Dit is enkel het geval wanneer de parameter CORE\_USER\_ADMIN\_CONTROL gelijk is aan 1

### De klasse USER

Om de code overzichtelijk te houden zal een klasse user worden gebruikt voor het werken met gebruikers. De klasse zal alle gegevens bevatten die voor een gebruiker worden bijgehouden. In 2.2.2 Community vs bedrijf werd reeds aangehaald dat er eventueel custom gegevens zouden kunnen worden bijgehouden van gebruikers.

De mogelijkheid tot het invoeren van deze custom fields zal later moeten worden onderzocht, in dit increment wordt enkele gewerkt rond de gegevens die van iedere gebruiker in ieder geval van belang zullen zijn:

* Gebruikersnaam (UNIEK)
* Wachtwoord (encryptie!!)
* Voornaam
* Familienaam
* Mailadres (UNIEK)
* Website
* Nationaliteit
* Taal
* Activatiecode (activatie nodig? <= CORE\_USER\_CONFIRMATION\_REQ)

Voor het ontwerp van de klasse user zal het wachtwoord niet worden opgenomen. Het wachtwoord zal slechts gecontroleerd worden bij het inloggen en het zou een beveiligingsrisico zijn om dit in de klasse bij te houden.

Wachtwoorden en gebruikersnamen zullen niet case-sensitive worden geprogrammeerd om het systeem gebruiksvriendelijk te houden.

Voor het samenstellen van de klasse user (onder /core/entity/user.php) zal enkel rekening gehouden worden met de gegevens die vaak gebruikt zullen worden wanneer er met gebruikers gewerkt wordt:

* Gebruikersid
* Gebruikersnaam
* Naam
* Voornaam
* Mailadres
* Website
* Nationaliteit
* Taal

Voor de andere gegevens die eventueel nodig zullen zijn zullen er aparte functies voorzien worden.

De klasse gebruiker zal er dus als volgt uitzien:



Figuur 12: De Klasse User

Een aantal parameters zoals Id en Username kunnen niet gewijzigd worden. Dit is om verkeerde bewerkingen te voorkomen. Een gebruikersid wordt toegewezen bij registratie en zal dus niet veranderd moeten worden. Een gebruikersnaam zou technisch gezien wel kunnen veranderen maar voor een systeem dat ook communities moet aandrijven is dit niet opportuun.

### De database

Voor het opslaan van gebruikers zal er een tabel moeten worden voorzien in de database. Deze zal er als volgt uitzien:



Figuur 13: De databasetabel users

Voor de landaanduiding zal een tabel country worden toegevoegd aan de database. Deze tabel zal slechts 1 veld bevatten namelijk countrycode. De bedoeling is om de gebruikers in de database een countrycode toe te wijzen. De effectieve landnamen zullen in de CORE languagefiles worden opgenomen.



Figuur 14: tabel countries

De codes in de tabel countries zijn deze van de ISO 3166 lijst.

### Wachtwoordbeveiliging

Net zoals in controlcenter 1.x wordt er gebruik gemaakt van het MD5 encryptiealgoritme. Er wordt deze keer echter een stap verder gegaan. Met behulp van zogenaamde rainbowtables zijn MD5 wachtwoorden te kraken. De enige oplossing is ervoor te zorgen dat er een zogenaamde ‘salt’ toegevoegde wordt. Het principe achter dit systeem is dat de salt toegevoegd wordt aan de wachtwoorden voor deze met MD5 gehasht worden en voor de naar de database worden geschreven. Bij het inloggen wordt de salt opnieuw toegevoegd voor MD5-encryptie.

Dit biedt de uitgelezen kans om de encryptie van de ene controlcenter site anders te maken dan die van de andere controlcenter site. Belangrijk is dat de salt slechts kan veranderd worden voordat de eerste gebruiker wordt toegevoegd, anders kan er niet meer ingelogd worden.

De saltvariabele wordt opgeslagen in buiten de public html in de controlcenter configuratiedirectory.

### Validatie van gegevens

Het toevoegen van gebruikers is de eerste bewerking waarbij een input van de eindgebruiker zal moeten worden gevalideerd. Er moet dus naar een goeie manier gezocht worden om de validatie volgens de multitier regels te laten verlopen.

Bijkomend probleem is dat er ook realtime zal worden gevalideerd door middel van AJAX-technologie. Er moet dus een manier gezocht worden om de gegevens op met dezelfde code zowel op de AJAX als op de traditionele manier te valideren (voor browsers die geen AJAX ondersteunen of gebruikers die deze functie uitzetten).



Figuur 15: Validatie van gegevens

Concreet is het zo dat er een php script wordt aangemaakt waarin het formulier wordt geladen via een addin. In die addin staat het verzendadres van het formulier naar het aanroepende script => Het formulier post eigenlijk naar zichzelf.

Aan het begin van het script wordt er altijd gecontroleerd of de variabele $\_POST waardes heeft. Op die manier kan er nagegaan worden of de gebruiker het formulier voor het eerst opent of niet. Wanneer blijkt dat er gegevens zijn dan moeten deze gevalideerd worden via functionaliteit die in de BusinessLogic layer zit. Wanneer die validatie slaagt worden de wijzigingen weggeschreven, wanneer de validatie niet slaagt moet de gebruiker het formulier opnieuw krijgen met de waarden die hij/zij heeft ingevuld en een foutmelding.

Om ervoor te zorgen dat het valideren van gegevens zo uniform mogelijk is wordt er gebruik gemaakt van een abstracte klasse Validator die zich in de entity-laag bevindt. Deze klasse heeft volgende structuur:

<?php

##De abstracte klasse validator wordt gebruikt voor de validatie van gegevens

abstract class Validator

{

private $validatorname;

public function \_\_construct($validatorname)

{

#Het enige wat bij de constructie gebeurt is het definiëren van de variabele $this->validatorname

$this->validatorname = $validatorname;

}

public function GenerateXMLResponse($fieldname,$message)

{

##Deze functie genereert een XMLfile die eventueel teruggegeven kan worden voor AJAX-functionaliteit

$domdocument = new DomDocument();

$maintag = $domdocument->CreateElement("response");

$fieldnametag = $domdocument->CreateElement("fieldname");

$fieldnametag->appendChild($domdocument->CreateTextNode($fieldname));

$messagetag = $domdocument->CreateElement("message");

$messagetag->appendChild($domdocument->CreateTextNode($message));

$maintag->appendChild($fieldnametag);

$maintag->appendChild($messagetag);

return $domdocument->SaveXML();

}

public abstract function ValidateObject($object);

public abstract function ValidateField($field,$value,$id);

}

?>

Deze klasse zorgt ervoor dat validators een vaste structuur hebben. Het is zo dat er bij de validatie van een volledig object (zal normaal voorkomen bij de traditionele manier van valideren) gebruik gemaakt wordt van de functie ValidateObject($object). Bij AJAX-validatie zal er slechts 1 veld tegelijk worden gevalideerd met behulp van de functie ValidateField($field,$value). Om ervoor te zorgen dat het XML bericht dat wordt weergegeven identiek is bij alle validators wordt de functie voor het genereren ook hier gedefinieerd en ingevuld.

Een belangrijke opmerking bij het gebruik van ValidateObject is dat dit een probleem zal opleveren bij het gebruik voor validatie van gebruikers. Het wachtwoord wordt immers niet in het object opgeslagen, bijgevolg zal er in de functie die de validatie regelt in de Logic-layer een aanroep moeten gebeuren van ValidateObject en van ValidateField voor het wachtwoord.

Het XML-bericht dat teruggegeven wordt bij AJAX ziet er als volgt uit:

<response validator=validatorname>

<fieldname>*naam van het veld in het aanroepend form*</fieldname>

<message>*teruggegeven bericht, leeg als de validator slaagt*</message>

</response>

Om ervoor te zorgen dat het gebruik van AJAX-validatie evenveel tijd kost als de traditionele manier van valideren worden er een aantal javascript functies voorzien

#### ValidateField(control,validator,identifier)

Deze functie neemt een DOM-element als attribuut (normaal gezien een textbox) en een validator (url naar een php script).

De functie doet het nodig voor het opzetten van een AJAX-request en voor het weergeven van de errors. Concreet betekent dit dat als je bij het laden van een pagina ValidatieField(this,”url naar validator’) koppelt aan bijvoorbeeld het onchange event dat je dan een werkend ajax systeem hebt.

De functie gaat voor het weergeven van de fouten op zoek naar de FORM-tag die boven het opgegeven element ligt en maakt daar een errorbox aan. De foutmeldingen worden gegroepeerd per validatorname, op die manier kan deze techniek gebruikt worden op verschillende formulieren op eenzelfde pagina.

Om ervoor te zorgen dat de code hergebruikt kan worden bij het wijzigen van gebruikers wordt een identifier toegevoegd. Deze zal verwijzen naar het id van de gebruiker: -1 als het om een nieuwe gebruiker gaat of het idnummer als het om een bestaande gebruiker gaat.

#### showError(fieldname,message,validator,formid)

Deze functie geeft een foutmelding weer. Als er nog geen fouten worden weergegeven voor die validator dan wordt een nieuwe errorbox aangemaakt, wanneer er wel een box is dan wordt de foutmelding toegevoegd.

Met deze functies kunnen dus ook foutmeldingen worden weergegeven zonder gebruik te maken van ValidateField. De methode heeft namelijk ook zijn nadelen: door de structuur van de functie en de validator zijn bepaalde controles niet mogelijk zoals bijvoorbeeld vergelijking van een veld wachtwoord en de bevestiging van het wachtwoord. Een oplossing hiervoor is een functie te schrijven in javascript voor de validatie van die twee velden en daar gebruik te maken van showError() zodat de foutmeldingen toch gegroepeerd kunnen worden.

Bij deze functie moet ook opgemerkt worden dat er één foutmelding kan worden weergegeven per id.

#### function removeError(fieldname,validator)

Deze functie verwijdert een foutmelding

#### function getLanguageConstant(type,itemname,constantname)

Wanneer er geen gebruik gemaakt wordt van servervalidatie bij controle van de invoer duikt het probleem van meertaligheid terug op. Als we zoals hierboven proberen 2 wachtwoorden te vergelijken dan moet de foutmelding ook in de actieve taal zijn. Vanuit javascript kunnen de languagefiles echter niet geopend worden. Daarom werd deze functie geschreven. Deze functie haalt via een synchrone ajax-transactie de waarde van een languageconstant op. Probleem is dat we in het geval van deze functie het antwoord op de functie nodig hebben voor we verder kunnen, daarom neemt dit tijd in beslag.

### Externe registratie

Op dit punt hebben we eigenlijk alle functionaliteit ontworpen die gemeenschappelijk is aan beide manieren van registreren. Het is zo dat zowel bij externe als interne registratie de gegevens die nu worden doorgestuurd zullen moeten worden opgegeven en gecontroleerd worden.

Om die reden kunnen we eigenlijk aan de externe registratie beginnen met een kopie van de interne registratie waar we dan later specifieke code aan kunnen toevoegen.

### Samenvatting

Aan het eind van dit increment kunnen we gebruikers toevoegen. Maar er werd veel meer gerealiseerd dan enkel dat. Er werd gezorgd voor een vaste structuur voor de validatie van gegevens op volgende manieren:

* Er werd een abstracte klasse Validator gemaakt aan serverzijde. Deze klasse heeft ingebouwde functionaliteit voor het maken van XML-responses. Ook de structuur van de klasse wordt vastgelegd met uniformiteit tot doel
* Er werden Javascript klasses voorzien aan cliëntzijde om ervoor te zorgen dat gegevens via AJAX realtime gevalideerd kunnen worden. Ook voor het weergeven van foutmeldingen werd herbruikbare code geschreven.
* Gebruikers kunnen toegevoegd en gewijzigd worden met gebruik van dezelfde validatorcode.



Figuur 16: Overzicht van AJAX-functionaliteit

Bijkomend werd een systeem van password salts uitgewerkt waarmee iedere controlcenter2 server een afwijkende manier zal hebben van wachtwoorden encoderen.

Op dit moment wordt er nog geen onderscheid gemaakt tussen interne en externe registratie omdat het systeem van permissions er nog niet is.

De externe registratie is op dit moment een kopie van de interne registratie. Het enige dat zij gemeenschappelijk hebben is de addin commonfields.tpa waarin de formuliervelden zitten opgeslagen die door beide scripts worden gebruikt.

## Increment5: Gebruikersgroepen en Permissions

### Inleiding

In dit increment moet er werk gemaakt worden van het systeem van gebruikersgroepen en van permissions.

Gebruikers worden standaard lid van één of meerdere gebruikersgroepen. De verschillende gebruikersgroepen hebben toegang tot verschillende controlcenter functies. De bedoeling is om met het systeem van gebruikersgroepen te zorgen voor een manier waarop de toegang van grote groepen gebruikers kan geregeld worden.

Er moet echter ook de mogelijkheid geboden worden op gebruikers manueel een bepaalde toegang te geven.

Doelstellingen van dit increment zijn;

* Uitbreiden van de database zodat gebruikersgroepen en permissions kunnen worden gebruikt
* Integreren van de permissions bij de registratie van gebruikers:
  + Bij interne registratie manueel gebruikers aan groepen toevoegen of manueel de toegang bepalen.
  + Bij externe registratie een gebruiker automatisch een bepaalde toegang te geven.
* Functionaliteit voor het ophalen van gebruikersgegevens

### Uitbreiding van de database

Bij deze uitbreiding komen er verschillende tabellen bij in de database. Volgende klassen worden in dit increment aan de database toegevoegd:

* Usergroups
* ModuleTasks

Meer info over deze klassen is te vinden bij 5.1.1 De klassen.

Om dat er verschillende veel-op-veel relaties aangeduid zijn in het uml-model zullen er tussentabellen moeten worden aangemaakt:

* UsergroupMembers: Tussentabel tussen Users en Usergroups: Een gebruiker kan lid zijn van meerdere groepen.
* UsergroupPermissions: Tussentabel tussen Usergroups en ModuleTasks waarin de toegang van de verschillende gebruikersgroepen wordt opgeslagen.
* UserPermissions: Tussentabel tussen Users en ModuleTasks waarin de toegang wordt opgeslagen van de gebruikers die tot geen enkele groep behoren.

#### De tabel Usergroups

De structuur van deze tabel ziet er als volgt uit:



Figuur 17: structuur van de tabel usergroups

#### De tabel Permissions

Deze tabel bevat alle mogelijke taken die met controlcenter kunnen worden uitgevoerd. Aan deze tabel zullen taken toegevoegd worden door modules die worden geïnstalleerd. Taken worden gekoppeld aan een module, er zijn ook permissions die niet gekoppeld zijn aan een module, het gaat om de zogenaamde CORE-permissions.

De structuur van deze tabel ziet er als volgt uit:



Figuur 18: structuur van de tabel moduletasks

Iedere moduletask krijgt automatisch een id toegewezen. Dit lijkt misschien vreemd omdat het id bij iedere controlcenter installatie zal verschillend zijn afhankelijk van de volgorde waarop de modules zij geïnstalleerd. De reden hiervoor is dat er op die manier naar de ids kan verwezen worden in de tussentabellen om de snelheid van het systeem te bevorderen.

#### De tabel UsergroupMembers

Deze tabel houdt bij van welke groepen een bepaalde gebruiker lid is.

De structuur van de tabel ziet er als volgt uit:



Figuur 19: structuur van de tabel usergroupmembers

Beide velden maken deel uit van de sleutel omdat het de combinatie is van de factoren die uniek zal zijn.

#### De tabel UsergroupPermissions

Deze tabel bevat alle toegangsrechten van een bepaalde groep. De structuur van deze tabel is als volgt:



Figuur 20: structuur van de tabel usergrouppermissions

Het veld usergroup zal gebruikt worden voor het id van de gebruikersgroep

#### De tabel UserPermissions

Deze tabel wordt gebruikt voor gebruikers waarvan de toegang manueel werd bepaald.

De structuur is gelijkaardig aan die van de usergrouppermissions:



Figuur 21: structuur van de tabel UserPermissions

Het veld user zal gebruikt worden voor het id van de gebruiker.

#### De tabel Modules

Deze tabel zal de modules bevatten die geïnstalleerd zijn op de server. In deze fase van de ontwikkeling worden enkel de noodzakelijke velden voor het toegangsbeheer gedefinieerd. Dit wordt later nog aangepast.

### De Klasse usergroup

Om ervoor te zorgen dat werken met usergroups niet te complex is wordt gebruik gemaakt van een klasse usergroup die alle gegevens over die gebruikersgroep moet gaan bijhouden:

* Id
* Naam
* Toegang verbonden aan die groep



Figuur 22: De klasse Usergroup

#### Usergroup::setName(string)

Deze functie verandert de naam van de groep

#### Usergroup::getName()

Deze functie haalt de naam van de groep op

#### Usergroup::setPermissions($permissionlist[])

Deze functie accepteert een array met 1 dimensie als parameter met daarin de id’s van de tasks waar de groep toegang toe heeft.

#### Usergroup::getPermissions()

Deze functie geeft een array terug met daarin de id’s van de tasks waar de gebruiker toegang toe heeft.

### Toevoegen en wijzigen van usergroups

Naar analogie van het toevoegen en verwijderen van gebruikers wordt er ook hier gebruik gemaakt van de werkwijze met validators. Op die manier kunnen de controles die wordt uitgevoerd gebruikt worden bij zowel het toevoegen als het wijzigen van een usergroup.

Naast de toegangsrechten kan ook de naam van een usergroup gewijzigd worden. Er mag echter geen naam gekozen worden die al gekoppeld is aan een andere usergroup.

### Samenvatting

Het usergroup systeem zorgt ervoor dat grote groepen gebruikers kunnen beheerd worden. De rechten worden opgeslagen in de database en kunnen op ieder moment gewijzigd worden.

## Increment6: Koppeling tussen gebruikers en gebruikersgroepen

### Doelstelling

Door het werk dat verricht werd in de vorige incrementen hebben we nu een systeem om gebruikers te maken en te bewerken en is er een systeem om gebruikersgroepen te maken en te bewerken.

Nu moet de mogelijkheid gecreëerd worden om gebruikers lid te maken van één of meerdere usergroups.

### Toevoegen van gebruikers aan een gebruikersgroep bij creatie

Bij het aanmaken van gebruikers werd een onderscheid gemaakt tussen interne en externe registratie. Eén van de voornaamste redenen hiervoor is dat er bij interne creatie bepaald zal worden tot welke gebruikersgroepen deze gebruiker zal horen.

Bij externe creatie is vooraf bepaald bij welke gebruikersgroep een gebruiker wordt ingedeeld. Eerst en vooral moet er een aanpassing gebeuren aan het user-object zodat dit ook de gebruikersgroepen kan bevatten. Gebruikers kunnen lid zijn van meerdere groepen, er zal dus met een private array gewerkt worden. Er worden 2 functies toegevoegd aan het userobject: setUsergroups() en getUsergroups(). In de businesslogic en dataaccess layers wordt er een functie toegevoegd die een overzicht geeft van de gebruikersgroepen. Om ervoor te zorgen dat die functies niet alle permissions ook moeten inladen wordt er hier geen gebruik gemaakt van een usergroup object maar wordt een array gebruikt die de naam van de gebruikersgroep bevat en het databaseid.

### Gebruikersgroepen en externe registratie

Gebruikers die zelf een account registreren (op sites waar dit toegelaten is) worden automatisch ingedeeld bij 1 gebruikersgroep die door de admin vastgesteld wordt.

Dit systeem maakt gebruik van een parameter CORE\_USER\_EXT\_USERGROUP. Deze parameter krijgt de waarde die overeenkomt met het id van de gebruikersgroep in kwestie.

### Samenvatting

Aan het einde van increment6 begint het gebruikerssysteem vorm te krijgen. Gebruikers kunnen aangemaakt worden (zowel intern als extern). Er kunnen gebruikersgroepen aangemaakt worden die bepaalde toegang hebben. De toegang van deze groepen kan achteraf gewijzigd worden en de gebruikerskunnen aan 1 of meerdere gebruikersgroepen gekoppeld worden.

## Increment7: Verificatie van extern geregistreerde gebruikers

Eén van de grootste problemen die zich voordoen op community websites is spam. Het is daarom erg belangrijk dat controlcenter2 een doeltreffend systeem heeft om spammers buiten te houden.

Een 100% werkend systeem is onmogelijk te implementeren, maar het is de bedoeling een systeem uit te werken dat voor zo weinig mogelijk last zorgt bij echte gebruikers en dat toch minstens 90% van de spammers op afstand houdt. De andere 10% zal dan door manuele controle moeten gefilterd worden.

Er worden verschillende parameters gedefinieerd die de eigenaar van de site toelaten om zelf te bepalen hoe strikt de controle op nieuwe gebruikers is.

Om de visuele controle vlot te laten verlopen wordt gebruik gemaakt van bestaande captcha-technologie van reCaptcha ([www.recaptcha.net](http://www.recaptcha.net)). Deze captcha kan visueel gecustomised worden zodat hij perfect bij het formulier behoort.



Figuur 23: Activatiescherm

De gebruiker moet zijn gebruikersnaam en wachtwoord ingeven en de tekst die in de captcha wordt weergegeven om zijn account te activeren.

Afhankelijk van de waarde van de parameter CORE\_USER\_ADMIN\_ACTIVATION zal een administrator dan de account nog manueel moeten nakijken vooraleer deze bruikbaar wordt.



Figuur 24: Controlcenter anti-spam systeem

### De Klasse Email

Dit is de eerste keer dat Controlcenter2 mails zal moeten versturen. Omdat de mailfunctie in PHP niet bepaald gebruiksvriendelijk is wordt een klasse Email geïmplementeerd. Deze klasse zal het versturen van mails gemakkelijker maken.

Bijkomend zal dit ook toelaten om HTML-mails te genereren met behulp van het templatesysteem. De klasse Email zal er als volgt uitzien:



Figuur 25: De klasse Email

De Emailklasse wordt niet geïmplementeerd in één van de bestaande lagen. Net zoals het templatesysteem wordt ze buiten de normale CORE-lagen gehouden.

De klasse mail zal gebruikt worden voor alle mails die door controlcenter verstuurd moeten worden. Om ervoor te zorgen dat zoveel mogelijk mails van eenzelfde mailadres worden gestuurd zal de afzender van de mail standaard de waarde krijgen van de parameter CORE\_SERVER\_MAILADRESS.

E-mail die met deze klasse verstuurd wordt heeft altijd HTML-opmaak.

Voor het gebruik van het mailsysteem wordt er een nieuwe standaar templatealias voorzien. De alias mail zal standaard gebruikt worden als er bij de constructor geen andere alias wordt meegegeven.

### Templatesystem revisie 2

Templatesystem R1 wordt op dit moment gebruikt door [www.detoverlantaarn.be](http://www.detoverlantaarn.be) en [www.3musketiers.be](http://www.3musketiers.be) en heeft daar op dit moment niets dan voordelen opgeleverd. Alle stabiliteitsproblemen van de beginfase lijken achter de rug.

Om het templatesysteem ook te kunnen gebruiken voor het versturen van E-mail zal het echter nodig zijn om een aantal kleine extra aanpassingen door te voeren die de bruikbaarheid van het templatesysteem verder uitbreiden.

#### $htmlp age->getHTML()

Deze functie geeft de HTMLcode terug zoals deze op het scherm zou worden afgedrukt met PrintHTML().

Op deze manier kan de HTML-code opgeslagen worden of gebruikt bij andere functies.

#### $htmlpage->insertCode($codestring)

Deze functie zal op de plaats van de PageContent-tag code invoegen die aangeleverd wordt via $codestring. Dit zorgt ervoor dat de HTML die ingevoegd wordt op het template niet noodzakelijk meer uit .tpa files hoeft te komen. Toch blijft het voor HTML die moet weergegeven worden sterk aan te raden gebruik te maken van addins om de code overzichtelijk te houden.

### Wijzigingen aan de databasetabel users

Om de verschillende statuswijzigingen van een extern aangemaakte gebruikersaccount te kunnen opvolgen zullen er een aantal wijzigingen nodig zijn in de databasetabel users.

Er worden 2 kolommen toegevoegd:

* De kolom userconfirmation: Deze kolom kan de waarde 0 of 1 krijgen. Wanneer een externe account wordt aangemaakt en CORE\_USER\_SELF\_ACTIVATION is 1 dan zal deze kolom de waarde ‘N’ krijgen en krijgt de gebruiker een mail met uitleg over hoe hij zijn account moet activeren. Wanneer CORE\_USER\_SELF\_ACTIVATION 0 is of wanneer het om een intern aangemaakte account gaat dan zal deze kolom onmiddellijk op Y komen.
* De kolom adminconfirmation kan de waarden 0 of 1 krijgen. Wanneer een externe account wordt aangemaakt en CORE\_USER\_ADMIN\_ACTIVATION is 1 dan zal deze kolom de waarde ‘N’ krijgen. Van zodra een beheerder de account heeft goedgekeurd zal deze waarde gewijzigd worden naar ‘Y’ en kan de gebruiker inloggen met zijn account. Wanneer CORE\_USER\_ADMIN\_ACTIVATION gelijk is aan 0 of het gaat om een intern aangemaakte account dan zal deze kolom automatisch de waarde ‘Y’ krijgen.

### Wijzigingen aan het userobject

Omdat het object ook moet kunnen gegevens bevatten over de status van een account moeten er wat aanpassingen gebeuren:

* Bij de constructor wordt er ruimte voorzien om de adminactivation en useractivation waarden door te geven aan het object
* Er worden 2 publieke functies geïmplementeerd:
  + getUserConfirmationStatus(): Deze functie geeft true of false terug afhankelijk van de waarde van userconfirmation in de database
  + getAdminConfirmationStatus(): Deze functie geeft true of false terug afhankelijk van de waarde van adminconfirmation in de database.

### Versturen van mail bij het activatieproces

In eerste instantie zal controlcenter2 de gebruiker een mail sturen met een activatielink. Op die manier kan de gebruiker zijn account bevestigen. Deze mail wordt enkel verstuurd wanneer gebruiker zijn account moet bevestigen.

De mails voor de bevestiging moeten in de taal zijn van de site. Dit heeft tot gevolg dat er in de addin voor de bevestigingsmail gewerkt zal moeten worden met language constanten.

In de activatiemail krijgt de gebruiker een link die verwijst naar /core/presentation/usermanagement/accounts/activate.php?id=xxx

Xxx wordt vervangen door het id van de nieuwe gebruiker.

### Integratie van reCaptcha in Controlcenter2

reCaptcha wordt aangeleverd in de vorm van een klasse die in ieder PHP-script kan gebruikt worden om controles uit te voeren. Het is echter zo dat er bij de captcha gebruik gemaakt wordt van een publieke en private sleutel.

Standaard wordt er een universele sleutel gebruikt die op gelijk welk domein werkt. Om er echter voor te zorgen dat de beveiliging optimaal kan worden aangewend is het zo dat er voor de publieke en private sleutel parameters worden aangemaakt zodat er eventueel een andere sleutel kan worden gebruikt per server.

### Toegevoegde functies

#### checkUserPassword($username,$password)

Deze functie controleert de gebruikernsaam-wachtwoord combinatie en geeft true of false terug.

#### showMessage($title,$body,$link=’’,$linktext=’’,$alias=’frontend’)

Deze functie heft niet rechtstreeks iets te maken met het huidige increment maar de noodzaak van deze functionaliteit werd wel duidelijk bij de ontwikkeling van dit increment.

showMessage creëert een simpele HTML pagina met een titel, een boodschap en eventueel een link.

## Samenvatting

Aan het einde van dit increment kunnen gebruikers toegevoegd worden door de beheerder en kunnen gebruikers zichzelf ook registreren.

Er zijn mogelijkheden om het gebruikerssysteem aan te passen op maat van de site die aangedreven wordt:

* Er kan bepaald worden dat gebruikers zichzelf niet kunnen registreren zodat enkel admins accounts kunnen aanmaken
* Er kan bepaald worden hoe strict de activatieprocedure is:
  + Activatie door de gebruiker
  + Activatie door de beheerder
* Er kunnen verschillende gebruikersgroepen gedefinieerd worden met elk verschillende toegangen
* Een gebruiker kan toegevoegd worden aan één of meerdere groepen
* Er kan bepaald worden tot welke groep gebruikers die zichzelf registreren zullen behoren.

Er zijn echter ook nog een héél aantal ToDo’s:

* De bevestigingsschermen van de registratie nakijken en zonodig aanpassen
* Bepalen dat gebruikers hun wachtwoord moeten wijzigen
* Gebruikers moeten hun eigen info kunnen bewerken
* Functionaliteit voor vergeten wachtwoorden

Deze tekorten houden de verdere ontwikkeling van het systeem echter niet tegen en zijn voor de huidige Controlcenter-sites van minder groot belang. Ze zullen dan ook later geïmplementeerd worden.

# Increment 8: inloggen en beveiliging

## Inloggen

Nu de functionaliteit rond creëren van accounts en activeren van accounts voldoende ontwikkeld is kan er begonnen worden het het ontwikkelen van login en beveiliging.

De bedoeling is dat bepaalde pagina’s enkel toegankelijk zijn voor bepaalde groepen gebruikers. Dit houdt in dat als bepaalde pagina’s geopend worden zonder ingelogd te zijn dat de gebruiker dan om gebruikersnaam en wachtwoord gevraagd wordt.

In andere gevallen waar een gebruiker wel ingelogd is maar niet tot de juiste gebruikersgroep behoort zal er gemeld worden dat de gebruiker geen toegang heeft.

Het is zo dat het loginsysteem zoveel mogelijk wordt losgekoppeld van de HTML en eigenlijk als een soort van doorgang fungeert:



Figuur 26: login.php als tussenstap

Login.php verwerkt de gegevens die via een andere pagina worden aangeleverd en verstuurt de gebruiker vervolgens naar een andere pagina: een bestemmingspagina of een loginpagina.

## Login.php

Om het systeem besproken in 8.1 te kunnen realiseren moeten er een aantal parameters gedefinieerd worden die aangeleverd worden naar login.php zodat het script zijn werk kan doen.

De volgende parameters worden gedefinieerd:

* u: username
* p: password
* d: destination: de pagina waar de gebruiker (bij succesvol inloggen) naartoe verwezen wordt. Dit zou een facultatieve parameter moeten worden, wanneer deze waarde niet gedefinieerd wordt dan wordt de gebruiker gewoon teruggestuurd naar de referral page.

De parameter referral zou eigenlijk moeten automatisch bepaald worden, maar in eerste instantie zal die aangeleverd worden.

Wanneer een login niet succesvol is wordt de gebruiker doorverwezen naar een login-pagina. Het is namelijk zo het loginformulier vaak in de navigatie wordt verwerkt en dat de foutmeldingen in dergelijk geval moeilijk kunnen weergegeven worden in de navigatie omdat die iedere keer anders is.

Om die reden wordt de gebruiker naar een inlogformulier verwezen waar de foutmelding wordt weergegeven.

#### Logic: Login($username,$password)

Deze functie gaat via DataAccess layer na of de gebruikersnaam en wachtwoord correct zijn. De functie haalt het userobject op en creëert een sessievariable $\_SESSION[‘currentuser’].

De functie controleert ook of de activatieprocedure werd doorlopen en zal pas de gebruiker inloggen wanneer dit effectief is gebeurd.



Figuur 27: inlogprocedure

## CheckPermission

De checkpermission vormt de sleutel tot het beveiligen van pagina’s. De bedoeling is om pagina’s te kunnen beveiligen met slechts 1 regel code die toegevoegd wordt aan het begin van de broncode.

CheckPermission(“forum”,”moderator”);

Met bovenstaande code zou gekeken worden of de ingelogde gebruiker het toegangsrecht tot de taak “moderator” op de module “forum” heeft.

Om deze functie te laten werken wordt er gekeken naar de gegevens in $\_SESSION[‘currentuser’]. In deze sessievariabele staan de gebruikersgroepen opgeslagen waar de gebruiker bij hoort. (zie Figuur 12: De Klasse User). Er wordt dan nagekeken welke toegangsrechten bij de groep horen en op die manier kan er afgeleid worden of de gebruiker toegang heeft of niet.



Figuur 28: Werking CheckPermission()

## Gedwongen wachtwoordwijziging

Om een optimale privacy te garanderen kan het zijn dat een administrator een account aanmaakt met een voorlopig wachtwoord. De gebruiker moet dit wachtwoord dan wijziging van zodra hij/zij de volgende keer inlogt.

Hiervoor moet er een aanpassing gebeuren in het userobject (zie Figuur 12: De Klasse User).

De functie getPasswordChangeRequired zal true of false teruggeven waardoor het systeem kan bepalen of het wachtwoord gewijzigd moet worden. Bij het inloggen wordt een gebruiker doorverwezen naar een pagina het oude wachtwoord en een nieuw wachtwoord (2 maal) moet worden ingegeven.

Deze pagina zal kunnen hergebruikt worden voor de functie ‘wachtwoord vergeten’ en voor het geval een gebruiker zelf zijn wachtwoord wil wijzigen.

## Samenvatting

Aan het einde van dit increment kan er ingelogd worden. Er wordt bij het inloggen ook gecontroleerd of de account actief is.

Er zijn echter nog ToDo’s:

* Remember me functie waardoor de gebruiker niet telkens zijn login opnieuw moet geven.
* Er wordt vooral gebruik gemaakt van functies die al geïmplementeerd waren waardoor er soms teveel queries worden uitgevoerd. Het zou efficiënter kunnen, maar dat is nu geen prioriteit.

# Increment 9: Beveiligen van de eerste 8 incrementen

## Inleiding

Nu Controlcenter2 beschikt over een flexibel beveiligingssysteem moet dit geïmplementeerd worden in de reeds beschikbare functionaliteit.

## Permissions

Op dit moment zijn er enkel functies beschikbaar die te maken hebben met het usermanagement systeem. Voor dit systeem moeten er nu permissions gedefinieerd worden. Deze permissions zullen steeds behoren tot het core component ‘usermanagement’.

Volgende permissions worden gedefinieerd:

* Manage usergroups
* Add user
* Edit user

## Standaardinstellingen

De bedoeling is dat de finale release van Controlcenter2 vlot geïnstalleerd kan worden. Om die reden is het nodig om bij installatie basisinstellingen te voorzien die geschikt zijn voor beginnende sites. Met behulp van de verschillende functionaliteiten zullen administrators de permissions dan kunnen aanpassen volgens de noden van het project.

Er wordt een standaard gebruikersgroep ‘usermanagement’ aangemaakt bij installatie. Er wordt 1 gebruiker voorzien die toegang heeft tot deze groep en alle andere groepen. De gebruikersnaam en het wachtwoord van deze gebruiker wordt gekozen tijdens de installatieprocedure.

# Increment10: AJAX functionaliteit

## Inleiding

Bij het ontwikkelen van het gebruikersbeheerd werd een manier uitgewerkt om ajax transacties te gebruiken in controlcenter. Het ging echter op een aantal losse functies. Achteraf bleek het ook een stuk extra werd om effectief AJAX te gaan implementeren met deze functies.

Om die reden werd de manier waarop AJAX gebruikt wordt nog eens opnieuw bekeken

## Filosofie

De bedoeling is om een manier uit te werken waarbij in slechts een aantal lijnen code een transactie opgezet kan worden zonder dat er een al te grote wijziging nodig is in de manier van programmeren.

Om dit te werkstelligen wordt er met 2 objecten gewerkt:

* Het object ajaxTransaction() in javascript: dit object zal alle clientside bewerkingen voor zijn rekening nemen. In het bijzonder:
  + Het bijhouden en structureren van de data die naar de server moet verstuurd worden
  + Verbinden met de server
  + De link leggen met de functionaliteit die moet uitgevoerd worden nadat de bewerking op de server voltooid is
  + De gegevens terug ontvangen, structureren en doorgeven aan een andere javascriptfunctie die uitgevoerd moet worden
* Het object ajaxResponse in php:
  + Dit object ontvangt de request van de clientside en verwerkt de parameters die met de request werden meegestuurd
  + Het object maakt de link met de serverside functionaliteit die moet uitgevoerd worden.
  + Het object verzamelt en structureert de data die terug moet gestuurd worden naar de client. Dit gebeurt via XML

## Het protocol

De communicatie van client naar server gebeurt via de POST method: het ajaxTransaction object bouwt de requeststring op aan de hand van de data die werd doorgegeven

De communicatie van de server naar de client gebeurt via XML. Er zijn 2 verschillende structuren mogelijk.

### Wanneer er geen fouten voorkomen

Deze manier van communiceren wordt gebruikt wanneer alles goed verlopen is. Er kunnen zich 2 scenarios voordoen:

* De server bevestigt gewoon dat alles ok is
  + Vb: gebruiker is verwijderd, mail is verstuurd, etc.
* De server bevestigt dat alles ok is en stuurt data terug
  + Vb: Er wordt via ajax een lijst opgevraagd van de gebruikers

De structuur ziet er als volgt uit:

<response>

<status>ok</status>

<datareturn>

<datarow>…variabele data</datarow>

<datarow>…variabele data</datarow>

</datareturn>

</response>

Wanneer er geen data teruggegeven wordt zal de datareturn tag niet aangemaakt worden en krijg je dus gewoon de response en status tags.

### Wanneer er fouten voorkomen

Wanneer er zich fouten voordoen zal de server foutmeldingen moeten teruggeven die aan de gebruiker getoond moeten worden. Het gaat niet om exceptions maar om warden die door de gebruiker werden ingevoerd en die na controle door de logic layer niet blijken te kunnen.

De structuur van de XML wijzigt in de geval naar het volgende:

<response>

<status>error</status>

<errorlist>

<error><parameter>*parameternaam*</parameter><message>*foutboodschap*</message>

…

</errorlist>

</response>

Deze structuur geeft de mogelijkheid aan het ajaxTransaction object om de foutmeldingen aan te leveren maar ook de parameter die de foutmelding veroorzaakt, het zou hier bijvoorbeeld kunnen gaan om de naam van het formulierveld waarin de foute waarde zich bevindt. Dit kan dan weer gebruikt worden voor de errorhandling

## Gebruik van het ajaxTransactionobject

Het object werd geprogrammeerd als Javascript klasse om alle functionaliteit te centraliseren.

### Instantiering

Aangezien het om een klasse gaat moeten we eerst een nieuwe instantie opmaken van de klasse. Wanneer we dit doen creëert het object alle nodige verbindingen met de server.

Var transactie = new ajaxTransaction();

De constructor aanvaardt geen argumenten (voorlopig).

### Data aanleveren om door te sturen naar de server

Wanneer de een opvraging doen bij de server zal dit niet gebruikt worden, maar bij validatie van gegevens zullen deze gegevens doorstuurd worden naar de server. Dit gebeurt via de addData method:

Transactie.addData(parameter,waarde);

Hier worden de waarden doorgegeven met hun parameter (kan de naam van een veld zijn). De bedoeling is dat wanneer er iets verkeerd loopt dat de errormessage ook de parameter zal krijgen van de data die de fout heeft veroorzaakt.

### Activity indicator

Vaak willen we de gebruiker tonen dat er een request aan de gang is. Op die manier wordt duidelijk dat de browser niet is vastgelopen maar dat de browser wacht op een antwoord van de server.

ajaxTransaction biedt de mogelijkheid om de progressindicator automatisch te tonen en te verbergen. Wanneer de request verstuurd is zal ajaxTransaction de indicator zichtbaar maken, wanneer het antwoord ontvangen wordt zal ajaxTransaction deze automatisch opnieuw verbergen.

Om de functionaliteit te gebruiken moet de method setIndicator(id) gebruikt worden. Als parameter moet de naam van het object aangeleverd worden dat als progressindicator gebruikt wordt.

Transactie.setIndicator(‘indicatorelementnaam’);

### onComplete

ajaxTransaction moet zo flexible mogelijk zijn. Omdat we niet weten wat er zal gebeuren met de data die terugkomt zal die functionaliteit wel nog manueel geprogrammeerd moeten worden voor een specifieke pagina. Om dat mogelijk te maken werd het public property onComplete geïmplementeerd. Hier gebeurt de koppeling aan een functie die buiten het ajaxTransaction object wordt gedefinieerd.

Transactie.onComplete = function() { alert(‘ok’);};

### executeRequest

Wanneer de data is aangeleverd kan de request uitgevoerd worden.

Transactie.executeRequest();

## Gebruik van het ajaxResponse php object

BIJLAGE A: Configuratieparameters

|  |  |  |
| --- | --- | --- |
| PARAMETER | MOGELIJKE WAARDEN | UITLEG |
| CORE\_LANG | Integer die verwijst naar een id in de tabel languages | CORE\_LANG bepaalt welke de standaardtaal is die van toepassing is op de site. |
| CORE\_USER\_EXT\_REGISTRATION | 0 of 1 | Deze parameter bepaalt of de gebruikers zichzelf kunnen registreren. |
| CORE\_USER\_EXT\_USERGROUP | Integer die verwijst naar een id in de tabel usergroups | Deze parameter bepaalt bij welke gebruikersgroep de gebruikers worden ingedeeld wanneer ze zich registreren. |
| CORE\_USER\_SELF\_ACTIVATION | 0 of 1 | Deze parameter bepaalt of een gebruiker die zich registreert zijn account moet bevestigen alvorens deze geactiveerd wordt. |
| CORE\_USER\_ADMIN\_ACTIVATION | 0 of 1 | Deze parameter bepaalt of gebruikers manueel moeten goedgekeurd worden door een administrator |
| CORE\_SERVER\_MAILADRESS | Mailadres | Deze parameter is het mailadres dat door de mailklasse standaard als afzender adres wordt gebruikt |
| CORE\_SITE\_NAME | String | Bevat de naam van de site zoals die door controlcenter weergegeven wordt. |
| CORE\_RECAPTCHA\_PUBLIC | String | Publieke reCaptcha sleutel die wordt gebruikt bij controle van input |
| CORE\_RECAPTCHA\_PRIVATE | String | Private reCaptcha sleutel die wordt gebruikt bij controle van input |
| CORE\_NOACCESS\_URL | String | url waar de gebruiker naar wordt verwezen als hij ingelogd is maar onvoldoende toegangsrechten heeft. |

BIJLAGE B: Databaseschema