Automatic plugin testing

Conceptontwikkeling voor het automatisch testen van plug-ins in Content Management Systemen

**Danny Gloudemans  
Eindhoven  
07 juni 2012**

**Danny Gloudemans  
07 juni 2012  
Eindhoven**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Gegevens stagebedrijf:** | |  | **Gegevens stagiair:** | |
| Naam: | ISAAC |  | Voorletters: | D.J.A.G. |
| Adres: | Marconilaan 16 |  | Naam: | Danny Gloudemans |
|  | 5621 AA Eindhoven |  | Adres: | Overschot 19 |
| Telefoonnummer: | +31 (0)40 290 89 79 |  |  | 5761EJ Bakel |
| E-mailadres: | info@isaac.nl |  | Telefoonnummer: | + 31 (0) 6 13 70 59 52 |
|  |  |  | E-mailadres: | danny.gloudemans@isaac.nl |
| **Bedrijfsmentor:** | |  |  |  |
| Naam: | Xander Steinnman |  | Studentnummer: | 2124331 |
| E-mailadres: | xander.steinmann@isaac.nl |  | Afstudeerrichting: | Software Engineering Voltijd |
| Telefoonnummer: | 040 215 53 66 |  | Opleiding: | Fontys Eindhoven |
|  |  |  |  |  |
| **Gegevens stagedocent:** | |  |  |  |
| Naam: | Coen Crombach |  | Begin stage: | 13 februari 2012 |
| E-mailadres: | c.crombach@fontys.nl |  | Einde stage: | 22 juni 2012 |
| Telefoonnummer: | + 31 8850 89241 |  | Werkdagen: | 85 dagen |

**Getekend voor gezien door bedrijf begeleider:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Datum getekend: | | |  | Naam en handtekening bedrijf begeleider: | |
|  |  | | |  |

# Voorwoord

Deze afstudeerscriptie heb ik geschreven naar aanleiding van mijn afstudeerstage als student software engineering aan de Fontys Hogeschool te Eindhoven. De afstudeerstage vond plaats in de periode februari 2012 tot en met begin juni 2012 bij ISAAC Software Solutions te Eindhoven. In het vervolg van dit document wordt er gesproken over ISAAC om het makkelijk leesbaar te houden.

Tijdens deze afstudeerstage heb ik een afstudeerproject uitgevoerd van 680 uur. Dit project is getiteld: *"Automatic Plugin Testing"* en bestaat uit twee onderdelen, namelijk het onderzoek en de ontwikkeling van de testomgeving voor het testen van plug-ins die gebruikt worden in dotCMS. De keuze voor de testomgeving en de inrichting daarvan worden gebaseerd op de resultaten van het onderzoek. Zowel het onderzoek als de ontwikkeling van de testomgeving komen uitgebreid aan bod in deze scriptie.

Graag wil ik in dit voorwoord van de gelegenheid gebruikmaken om het bedrijf ISAAC te bedanken voor de mogelijkheid om mijn afstudeerstage bij ISAAC te lopen. Hierbij bedank ik in het bijzonder mijn bedrijfsbegeleider Xander Steinmann voor het begeleiden tijdens het project. Ook wil ik mijn overige collega’s bij ISAAC bedanken voor hun tijd en het delen van hun ervaring en kennis tijdens de uitvoering van mijn project. Verder bedank ik mijn docentbegeleider Coen Crombach voor zijn ondersteuning tijdens deze afstudeerstage.

Danny Gloudemans

Eindhoven, juni 2012

Inhoudsopgave

[Samenvatting 5](#_Toc326751285)

[Summary 6](#_Toc326751286)

[Verklarende woordenlijst 7](#_Toc326751287)

[1. Inleiding 9](#_Toc326751288)

[2. ISAAC Software Solutions 10](#_Toc326751289)

[2.1 Algemene informatie ISAAC 10](#_Toc326751290)

[2.2 Bedrijfsstructuur 10](#_Toc326751291)

[2.3 Producten 11](#_Toc326751292)

[2.4 Locatie ISAAC 11](#_Toc326751293)

[3. De opdracht 12](#_Toc326751294)

[3.1 Probleemstelling 12](#_Toc326751296)

[3.2 Doelstelling 12](#_Toc326751297)

[3.3 Opdrachtomschrijving 12](#_Toc326751298)

[4. Aanpak project voor het onderzoek 13](#_Toc326751299)

[4.1 Kennismaking 13](#_Toc326751300)

[4.2 Project Initiation Document 13](#_Toc326751301)

[4.3 Het leren van dotCMS 13](#_Toc326751302)

[4.4 dotCMS plug-in aanpassen 13](#_Toc326751303)

[4.4.1 CustomLogin Plug-in 14](#_Toc326751304)

[4.4.2 CustomLogin aanpassing voor het onderzoek 14](#_Toc326751305)

[4.4.3 CustomLogin aanpassing buiten afstudeerproject om (1) 14](#_Toc326751306)

[4.4.4 CustomLogin aanpassing buiten afstudeerproject om (2) 15](#_Toc326751307)

[5. Onderzoek 17](#_Toc326751308)

[5.1 Onderzoeksvraag 17](#_Toc326751309)

[5.2 Deelvragen 17](#_Toc326751310)

[5.3 Uitwerking deelvragen 17](#_Toc326751311)

[5.3.1 Wat is een plug-in? 17](#_Toc326751312)

[5.3.2 Wat is een content management systeem? 17](#_Toc326751313)

[5.3.3 Wat is het testen van plug-ins? 18](#_Toc326751314)

[5.3.4 Wat is een automatische testomgeving 20](#_Toc326751315)

[5.3.5 Wanneer een automatische testomgeving te gebruiken? 25](#_Toc326751316)

[6. Uitvoering 27](#_Toc326751317)

[6.1 Automatisch testen plug-ins met batch script 27](#_Toc326751318)

[6.2 Inrichten van een testomgeving – Jenkins 29](#_Toc326751319)

[6.3 Inrichting testomgeving voor het testen op compile niveau (1) 29](#_Toc326751320)

[6.4 Inrichting testomgeving voor het testen op compile niveau (2) 31](#_Toc326751321)

[6.5 Inrichting testomgeving voor het testen op deploy niveau (1) 34](#_Toc326751322)

[6.6 Inrichting testomgeving voor het testen op deploy niveau (2) 39](#_Toc326751323)

[6.7 Unittesten 43](#_Toc326751324)

[6.8 Jenkins op een server draaien 43](#_Toc326751325)

[7. Conclusie en aanbevelingen 44](#_Toc326751326)

[Evaluatie 45](#_Toc326751327)

[Zelfreflectie 45](#_Toc326751328)

[Doelen 45](#_Toc326751329)

[Literatuurlijst 46](#_Toc326751330)

[A. Project Initiation Document](#_Toc326751331)

[B. Ontwikkelingen CustomLogin Plug-in](#_Toc326751360)

[C. Batch bestand voor automatisch testen deployen](#_Toc326751361)

# Samenvatting

In deze afstudeerscriptie wordt de uitvoering van mijn afstudeeropdracht bij ISAAC te Eindhoven besproken. ISAAC is een klein bedrijf dat zich specialiseert op het gebied van web applicaties. Deze scriptie geeft inzicht in de verschillende onderdelen van de afstudeeropdracht alsmede de uitwerking hiervan. Ook wordt het resultaat van de opdracht besproken.

ISAAC ontwikkeld websites voor klanten waarbij het gebruik maakt van een Content Management Systeem genaamd dotCMS. In dotCMS is het mogelijk om plug-ins toe te voegen, zodat de website op maat kan worden gemaakt voor de klant. Deze plug-ins maken gebruik van de dotCMS API en geven de website extra functionaliteit. De door de plug-ins gebruikte API maakt het mogelijk om functionaliteiten van dotCMS aan te roepen. Dit heeft vele voordelen, maar ook een bijkomstig nadeel. Als dotCMS namelijk een nieuwe versie uitbrengt, kan het probleem optreden dat de dotCMS API veranderd. Deze verandering maakt het mogelijk dat sommige plug-ins niet meer correct werken. Omdat ISAAC meer dan 60 plug-ins heeft, kost het veel werktijd om ze allemaal handmatig te testen. Naar aanleiding van dit probleem is het idee ontstaan om een automatische testomgeving op te zetten. Deze automatische testomgeving controleert of de plug-ins correct deployen en test de functionaliteiten van de plug-ins door middel van unittesten.

Alvorens het project te starten is er meer kennis verkregen over dotCMS en het ontwikkelen van een plug-in. Dit gebeurde met behulp van het boek “dotCMS From the Ground Up” van Michael Fienen. Het ontwikkelen van een plug-in betrof het verbeteren van de CustomLogin Plugin. Hierna is met een onderzoek gestart, waarin is onderzocht of het zinvol zou zijn om een testomgeving op te zetten. Bovendien is er ook bekeken wat de beste testomgeving voor ISAAC zou zijn. Uit dit onderzoek kon geconcludeerd worden dat het voor ISAAC zinvol is om een automatische testomgeving op te zetten. ISAAC heeft een dusdanig aantal plug-ins en dotCMS komt dusdanig vaak met een nieuwe versie uit dat het zinvol is om een testomgeving op te zetten. Verder is uit het onderzoek Jenkins als meest geschikte testomgeving voor ISAAC naar voren gekomen. Jenkins bezit dezelfde of zelfs meer functionaliteiten dan andere testomgevingen en bovendien is Jenkins gratis te gebruiken.

Het gebruik van Jenkins zou ISAAC dus, behalve de werktijd, geen extra geld kosten.

Nadat dit onderzoek was uitgevoerd, is het concept ook daadwerkelijk gerealiseerd. Het is mogelijk gemaakt om plug-ins automatisch te testen door middel van unittesten in Jenkins. Deze testen worden gestart zodra er een nieuwe versie van de plug-ins op Subversion wordt geplaatst. Hierdoor hoeft de gebruiker niets anders te doen dan naar de webpagina van Jenkins te gaan om het resultaat van de testen te bekijken.

# Summary

This thesis is a summary of all activities during my graduation assignment at ISAAC, Eindhoven. ISAAC is a small company that is specialized in websites. Some of the different parts of my assignment were learning about dotCMS, developing plugins for dotCMS and research about a continuous integration tool. These parts will be discussed in this thesis, as well as the implementation of the assignment.

ISAAC uses a Content Management System for developing websites. This system is called dotCMS. In order to customize the websites for their clients, ISAAC uses plug-ins. These plug-ins use the dotCMS API and give the site additional functionality. The dotCMS API makes it possible to use functions of dotCMS. This has many advantages, but also an inconvenient disadvantage: when dotCMS releases a new version of the Content Management System, the problem could occur that the dotCMS API has been changed. This change of the dotCMS API will cause some of the plug-ins to not work properly. ISAAC has over 60 plug-ins, so it would take too much time to test the functioning of the plug-ins manually. Because of this problem a solution was developed to implement an automatic test environment that will test if the plug-ins deploy properly. This test environment will also check the functionality of the plug-ins through unit testing.

Before the project started, there was more knowledge acquired about dotCMS and the developing of plug-ins. This was done by using the book "dotCMS From the Ground Up" by Michael Fienen. The development of a plug-in consisted of improving the CustomLogin Plugin. After learning more about dotCMS and developing the plug-in, the investigation could start. In the investigation the question was asked if it is useful to set up a test environment. It was also examined which test environment would be the best suitable for ISAAC. After finishing the investigation it could be concluded that it is useful for ISAAC to set up an automatic test environment. ISAAC has so many plug-ins and dotCMS comes with a new version of dotCMS so often that it is useful to set up a test environment. Jenkins was chosen as automatic test environment, because it has the same or even more functionality than other test environments, and it is free to use. The only costs for ISAAC would be the working hours.

At the conclusion of the investigation it was realized that the plug-ins are automatically tested by unit testing in Jenkins. These tests start automatically when a newer version of the plug-ins is placed into Subversion. The only thing what the user has to do is go to the webpage of Jenkins and watch the results of the tests.

# Verklarende woordenlijst

|  |  |
| --- | --- |
| **A** |  |
| API | API staat voor application programming interface. Dit is een manier om een opdracht uit handen te geven aan een externe bron. |
| **B** |  |
| Back-end | De back-end is het onderdeel van een programma, website of applicatie waarin het overgrote deel van de opdrachten wordt uitgevoerd. |
| **C** |  |
| Content | Content is de totale verzameling van informatie. Deze informatie is vaak in verschillende tekstvelden in te vullen. In de front-end is het mogelijk om deze informatie uit te lezen. |
| Content Management Systeem | Een content management systeem is een software applicatie waarmee mensen eenvoudig, zonder veel technische kennis, documenten en gegevens op internet kunnen publiceren. |
| **D** |  |
| dotCMS | dotCMS is een content management systeem gemaakt in Java. |
| **F** |  |
| Front-end | De front-end is een interface tussen de computer en de computergebruiker. De front-end maakt interactie tussen mens en machine mogelijk. |
| **I** |  |
| Identifier | De identifier is een unieke code die het content onderscheidt van het andere content binnen dotCMS. |
| **J** |  |
| JAR | JAR staat voor Java ARchive, wat een standaard archiefformaat is voor bestanden die in de programmeertaal Java zijn geschreven. |
| Java | Java is een objectgeoriënteerde programmeertaal voor het ontwikkelen van software. |
| JUnit | JUnit is een test framework voor Java. |
| **P** |  |
| Plug-in | Een plug-in (ook wel invoegtoepassing) is een aanvulling op een computerprogramma. Dit houdt in dat het originele programma een functionaliteit erbij krijgt die het van oorsprong niet had. |
| **R** |  |
| Refactoren | Het herstructureren van de broncode van een computerprogramma met als doel de leesbaarheid en onderhoudbaarheid te verbeteren of het stuk code te vereenvoudigen. |
| Reguliere expressie | Een reguliere expressie is een patroon waaraan een tekst moet voldoen, bijvoorbeeld dat er in een email adres een @, punt en drie gedeeltes tekst moeten zitten. |
| **T** |  |
| Trigger | Een trigger is een mechanisme dat een proces in gang zet. |
| **U** |  |
| Updaten | Het bijwerken van bijvoorbeeld een tekst aan de huidige omstandigheden. |
| **V** |  |
| Velocity | Apache Velocity is een software project van Apache Software Foundation. Dit wordt gebruikt in dotCMS om code van de plug-ins aan te roepen/gebruiken. |

# Inleiding

Het is voor een bedrijf van groot belang dat de software die ze ontwikkelt altijd goed werkt. Dit is ook essentieel voor een internetbedrijf zoals ISAAC te Eindhoven (voor meer informatie over het bedrijf ISAAC zie hoofdstuk 2), dat gespecialiseerd is in het ontwikkelen van web applicaties. Echter werkt de door ISAAC ontwikkelde software niet altijd naar behoren, wat overigens niet aan ISAAC ligt. ISAAC gebruikt namelijk al enkele jaren een content management systeem genaamd dotCMS. Voor dit content management systeem ontwikkelen ze plug-ins om extra functionaliteit in de front- en back-end van dotCMS te krijgen. Deze plug-ins maken gebruik van de dotCMS API. Deze API zorgt ervoor dat er op een simpele manier gebruik kan worden gemaakt van de dotCMS logica. Zo hoeft er bijvoorbeeld voor het toevoegen van content geen hele methode worden geschreven maar kan er door middel van de dotCMS API met een coderegel content worden toegevoegd.

Echter is het zo dat dotCMS bij een nieuwe release onaangekondigd de API wel eens verandert. Hierdoor werken soms de plug-ins niet meer zoals ze horen te werken, omdat de aangeroepen methodes in de API ineens zijn veranderd. Wanneer ISAAC de servers gaat updaten naar een nieuwere versie van dotCMS kan het voorkomen dat de plug-ins niet meer naar behoren werken. Daarom dient het bedrijf elke plug-in handmatig te gaan testen op de functionaliteiten. Omdat er meer dan 60 plug-ins binnen ISAAC zijn, kost dit veel werkuren. Dit is echter niet het enige nadeel, want degene die deze plug-ins gaat controleren moet ook de kennis hebben over de functionaliteiten van de plug-in.

Het idee dat vanuit ISAAC naar voren kwam, is om een automatische testomgeving op te zetten waarmee de plug-ins automatisch worden getest op een bepaalde trigger, zie voor de opdrachtomschrijving hoofdstuk 3. Voordat het onderzoek werd uitgevoerd, werd er eerst kennis opgedaan over het ontwikkelen van plug-ins en dotCMS. Hoe dat is gegaan, is terug te vinden in hoofdstuk 4. Vervolgens werd er een uitgebreid onderzoek over het testen van plug-ins in een Content Management Systeem gedaan. In dit onderzoek werd er eerst gekeken wat plug-ins en Content Management systemen zijn., waarna er werd gekeken wat het testen van een plug-in precies inhoudt. Nadat dit duidelijk was, werd er bekeken welke testomgeving het beste kan worden gebruikt en wanneer er een testomgeving gebruikt zou moeten worden. Voor het complete onderzoek zie hoofdstuk 5.

Nadat het onderzoek was uitgevoerd werd er aan de hand van de resultaten een testomgeving opgezet en ingericht. Hoe dit precies in zijn werk ging is te lezen in hoofdstuk 6. In hoofdstuk 7 wordt er een conclusie geformuleerd over het onderzoek en de uitvoering van het project. Ook worden er aanbevelingen gedaan voor verdere verbeteringen.

# ISAAC Software Solutions

In dit hoofdstuk volgt een toelichting over het bedrijf.

## Algemene informatie ISAAC

ISAAC is in 1999 opgericht binnen de Technische Universiteit van Eindhoven door drie studenten. Het bedrijf kreeg de naam ISAAC, wat staat voor Internet Strategy And Automation Company. Sinds het ontstaan van ISAAC heeft het bedrijf een enorme groei meegemaakt en vandaag de dag telt het bedrijf 37 werknemers.

## Bedrijfsstructuur

Aan de top van ISAAC staan drie aandeelhouders B.V.’s, namelijk: Management B.V. Haldorn, Management B.V. Quzick en Stichting ISAAC B.V. Haldorn en Quzick zijn ieder voor 45% eigenaar van ISAAC. De laatst genoemde, de Stichting ISAAC B.V. is voor 10% eigenaar. De stichting ISAAC B.V. bestaat uit enkele medewerkers van ISAAC.

Onder de drie aandeelhouders bevindt zich ISAAC Holding B.V. wat verdeelt is in twee aparte afdelingen, namelijk ISAAC Software Solutions B.V. en ISAAC Web Solutions B.V. ISAAC Software Solutions B.V. bestaat uit medewerkers die zich bezighouden met softwareontwikkeling. ISAAC Web Solutions is de afdeling die zich vooral bezighoudt op het gebied van website design en concepting. Deze afstudeerstage vond plaats binnen Software Solutions.

Figuur 1: Bedrijfsstructuur ISAAC

Management B.V. Haldorn (Mark)

Management B.V. Quzick (Max)

Stichting  
ISAAC B.V.

ISAAC Holding B.V.

ISAAC Software Solutions B.V.

ISAAC Web Solutions B.V.

**45% 45% 10%**

## Producten

Een blik op ISAAC’s portfolio laat zien dat ze zich onder andere bezighouden met webontwerp, Content Management Systemen, back-end systemen en webshops. Enkele technieken die bij ISAAC gebruikt worden, zijn: Java, HTML, CSS, Javascript en XML. Deze producten worden vooral gemaakt voor klanten met een middel tot groot bedrijf in Nederland. Enkele producten waar ISAAC mee werkt, zijn:

**dotCMS** – dotCMS is een open source Content Management Systeem (CMS) waarmee klanten zelf, eenvoudig en zonder technische kennis, hun website beheren en aanpassingen doorvoeren. ISAAC kan standaardoplossingen aanbieden voor het beheren van websites of een compleet pakket op maat maken. De op maat gemaakte pakketten worden vaak gemaakt met behulp van plug-ins. Een plug-in (ook wel invoegtoepassing) is een aanvulling op een computerprogramma. Voor de plug-ins van dit content management systeem wordt er een concept voor een testomgeving opgezet.

**Magento** – Magento is een open source e-commerce web applicatie. Dit wordt gebruikt voor de ontwikkeling van professionele webshops voor klanten. ISAAC maakt hier zelf aanpassingen en plug-ins voor, zodat de webshop aan de wensen van de klant voldoet.

## Locatie ISAAC

Momenteel is ISAAC werkzaam op de Marconilaan 16 te Eindhoven in het kantoorpand Obelisk (zie figuur 2). Ze bezetten de vierde en vijfde verdieping van het kantoorpand. Op de vijfde verdieping bevind zich de receptie, administratie en designers. En op de vierde verdieping bevinden zich voornamelijk de software engineers.

  
Figuur 2: Obelisk Kantoorpand

# De opdracht

In dit hoofdstuk wordt gekeken naar de probleemstelling van de opdracht. Daar­naast wordt er ingegaan op de doelstelling achter de opdracht, waarna de opdrachtomschrijving volgt.



## Probleemstelling

ISAAC ontwikkelt websites voor klanten, waarvoor het een Content Management Systeem gebruikt, genaamd dotCMS. Om deze websites in dotCMS aan de wens van de klant te laten voldoen, dienen deze aangepast te worden. Dit doet ISAAC door middel van plug-ins.

Echter is het zo dat wanneer dotCMS een nieuwe versie uitbrengt van hun product het vaak voorkomt dat de API is veranderd. API staat voor application programming interface. Dit is een manier om een opdracht uit handen te geven aan een externe bron. Hierdoor is het mogelijk dat een plug-in die gebruik maakt van de dotCMS API niet meer juist functioneert.

Momenteel wordt er bij plug-ins na het uitkomen van een nieuwe versie van dotCMS handmatig getest of alle functionaliteiten nog correct werken. Dit heeft behalve dat handmatig testen van plug-ins veel tijd kost, ook het nadeel dat de tester alle functionaliteiten van de plug-in dient te kennen. Omdat de huidige werkwijze veel werktijd kost, is het nuttig om naar andere opties te kijken.

## Doelstelling

De doelstelling tijdens deze afstudeerstage is om te onderzoeken wat de beste manier is om een geautomatiseerde testomgeving op te zetten voor het testen van plug-ins. Tijdens het testen van een plug-in moet er gecontroleerd worden of de plug-in nog werkt in een nieuwe versie van dotCMS en of deze functioneel hetzelfde doet als voorheen. Wanneer het onderzoek is afgerond is het de bedoeling dat er een concept testomgeving wordt opgezet, waarbij de gebruiker in zo min mogelijk stappen een nieuwe plug-in aan de testomgeving kan toevoegen.

## Opdrachtomschrijving

Er moet een testomgeving opgezet worden voor het testen van plug-ins. Hierin wordt getest of de plug-ins werken in een nieuwe versie van dotCMS en of de plug-ins nog naar behoren functioneren. Voordat deze testomgeving wordt opgezet, wordt er eerst geleerd hoe dotCMS in elkaar steekt. Dit wordt gedaan door middel van het boek 'dotCMS From the Ground Up' van Michael Fienen. Wanneer er voldoende kennis over dotCMS is opgedaan, is de volgende stap om kennis op te doen over het ontwikkelen van een plug-in. Dit gebeurt naar aanleiding van een uitbreiding op de CustomLogin plug-in, die door de vorige stagiair is ontwikkeld. Wanneer het ontwikkelen en uitbreiden van de plug-in succesvol is voltooid, wordt er begonnen met het onderzoek. Dit onderzoek zal uitwijzen welke methodes het beste kunnen worden gebruikt voor het opzetten van een automatische testomgeving. Het resultaat zal worden verwerkt in een concept voor ISAAC.

# Aanpak project voor het onderzoek

In dit hoofdstuk wordt het procesverloop van het opstarten van het project beschreven.

## Kennismaking

De eerste dag van de afstudeerstage was op maandag 13 februari 2012. Deze dag begon met het kennismaken met de werknemers van ISAAC en een korte rondleiding binnen het bedrijf. Hierna werd de werkplek in orde gemaakt. Dit hield in dat de PC en het beeldscherm op de werkplek gezet moesten worden en daarop de benodigde software werd geïnstalleerd en geconfigureerd. Ook diende administratieve werkzaamheden te worden verricht, zoals het doornemen en ondertekenen van de afstudeerovereenkomst van Fontys en een overeenkomst van ISAAC.

## Project Initiation Document

Het eerste deel van de afstudeerstage was het definiëren van het project in het Project Initiation Document, met behulp van de projectomschrijving een definitieve opdrachtomschrijving te maken en deze in te leveren bij de docentbegeleider. Het schrijven van het Project Initiation Document was een terugkerend proces binnen de opleiding, waardoor een raamwerk met vaste onderdelen reeds beschikbaar was. Dit raamwerk vormt de basis voor het opzetten van het Project Initiation Document. De invulling van het Project Initiation Document is volledig zelfstandig gebeurd. Hierna werd het verstuurd naar de stagebegeleider, met hem zijn enkele onderdelen besproken en waar nodig aangepast.

## Het leren van dotCMS

Naast dat er in het begin werd gewerkt aan het Project Initiation Document, was het ook belangrijk dat er kennis werd gemaakt met dotCMS. Deze kennis is noodzakelijk om een kwalitatief goed onderzoek naar Content Management Systemen uit te voeren. Door deze kennis was het mogelijk om de basisprincipes van dotCMS en plug-ins toe te passen in het onderzoek. Dit gebeurde aan de hand van het boek: dotCMS From the Ground Up geschreven door Michael Fienen. Naast de toelichting op de basis van dotCMS wordt er ook dieper op het gebruik van code ingegaan.

## dotCMS plug-in aanpassen

Nadat er genoeg kennis was over het gebruik van dotCMS, was de volgende stap om meer ervaring met de plug-ins voor dotCMS op te doen. Deze kennis over het ontwikkelen van plug-ins is essentieel voor het onderzoek. Hierdoor ontstaat er een beter beeld van wat er precies getest moet worden in plug-ins en hoe deze in elkaar zitten. De plug-in die moest worden aangepast, was de CustomLogin plug-in die door de vorige stagiair was ontwikkeld. Hierna worden de drie versies beschreven die tijdens deze stage zijn gemaakt. De eerste aanpassing van de CustomLogin viel onder de afstudeeropdracht en is ontwikkeld voordat het onderzoek begon, dit wordt beschreven in 4.4.2 CustomLogin aanpassing voor het onderzoek. Tijdens de afstudeerstage is de plug-in verder ontwikkeld buiten het afstudeerproject om. Hoe dit in zijn werking ging is te vinden in secties 4.4.3 en 4.4.4.

### CustomLogin Plug-in

Om een goed beeld te krijgen van wat er nou in de plug-in is aangepast en wat de problemen waren, wordt eerst de functionaliteit van de CustomLogin plug-in beschreven. Met de CustomLogin plug-in kan er een losse gebruiker worden aangemaakt die wordt gekoppeld aan een dotCMS gebruiker. Hiermee kan er met een niet dotCMS gebruiker worden ingelogd in de front-end. Met deze gebruiker zijn er meer mogelijkheden dan met de standaard gebruiker waar dotCMS gebruik van maakt. De plug-in heeft als functionaliteiten:

|  |  |
| --- | --- |
| * Creëren van een account * Wijzigen van een account * Inloggen met een account | * Wachtwoord vergeten * Wachtwoord wijzigen d.m.v. een sleutel |

### CustomLogin aanpassing voor het onderzoek

De plug-in was nu zo gebouwd dat er bij het deployen van de plug-in twee tabellen werden aangemaakt. Waarbij er in de CustomUser tabel de gebruiker werd aangemaakt en in de dotCMSUser tabel de koppeling werd gelegd tussen de gebruiker en een dotCMS gebruiker. Tijdens deze versie was het de bedoeling dat de koppelingstabel werd verwijderd. Hierdoor moest er een veld worden toegevoegd aan de CustomUser tabel waarin het emailadres van de dotCMS gebruiker stond vermeld. Ook werd er een veld toegevoegd waarin een identifier stond van een automatisch door de plug-in aangemaakte content. De identifier is een unieke code die het content onderscheidt van het andere content. Hiermee kon er extra informatie over de gebruiker worden opgeslagen. Als de gebruiker dan was ingelogd kon er door middel van Velocity en de identifier de content velden worden uitgelezen. Hierbij ontstond het probleem dat het content niet rechtstreeks in de database kon worden toegevoegd maar dat er van de dotCMS API gebruik moest worden gemaakt. Achteraf bleek het veel makkelijker om gebruik te maken van de dotCMS API dan dat alles rechtstreeks in de database werd gezet. De laatste stap was dat er een reguliere expressie toegevoegd moest worden, deze controleert of het wachtwoord voldoet aan de eisen. Een reguliere expressie is een patroon waaraan een tekst moet voldoen, bijvoorbeeld dat er in een email adres een @, punt en drie gedeeltes tekst moeten zitten. Er was echter nog geen ervaring met reguliere expressies, waardoor er uitgezocht moest worden hoe deze het beste opgebouwd konden worden. Nadat de reguliere expressie gemaakt en getest was, werd deze geïmplementeerd in Java. Het controleren op de reguliere expressie gebeurde met behulp van een Pattern en een Matcher. Voor de exacte implementatie van de reguliere expressie zie bijlage B.

### CustomLogin aanpassing buiten afstudeerproject om (1)

Tijdens de tweede aanpassing van de CustomLogin plug-in was het de bedoeling dat alle gegevens die nu in de database stonden werden omgezet naar content. Hierdoor is het mogelijk dat de gegevens die in het content staan in de front-end worden gebruikt. Het probleem waar hierbij tegenaan werd gelopen is dat het eerst niet mogelijk leek om content te updaten. Na de hulp van Xander ingeroepen te hebben, bleek het zo te zijn dat de Inode van het content gezet moest worden. Echter was dit een update van een al bestaande content, die al eerder was opgehaald, waardoor er een lege string aan de Inode moest worden meegegeven. Een string is een tekenreeks in de programmeerwereld, in dit geval een lege tekenreeks.

Nadat deze aanpassingen succesvol waren toegevoegd is er samen met Xander door de code gelopen. Tijdens het doorlopen van de code zijn er tips gegeven, onnodige code verwijderd, methodes duidelijkere namen gegeven en code gerefactored. Refactoren houdt in dat code wordt geherstructureerd met als doel dat de code vereenvoudigd, makkelijker onderhoudbaar of duidelijker leesbaar wordt. Het was erg leerzaam om eens met iemand uit het bedrijfsleven door de code te lopen en zijn mening daarover te horen.

Het refactoren met Xander heeft de volgende leerzame punten opgeleverd:

* Het aantal regels van een methode moet nooit meer dan +/- 10 regels bedragen, zodat het goed leesbaar blijft.
* Consequent in elke class dezelfde opmaak gebruiken, zodat alle code leesbaar blijft.
* Het is belangrijker om een methode een goede naam te geven dan te zorgen dat de documentatie van de methode in orde is. De documentatie wordt namelijk vaak vergeten aan te passen wanneer de methode wordt aangepast.
* Geen afkortingen gebruiken, dit voorkomt onduidelijkheden in de methode/class.
* In een catch moeten de fouten altijd afgehandeld worden, dit kan anders problemen veroorzaken wanneer dit niet gebeurt.

### CustomLogin aanpassing buiten afstudeerproject om (2)

De derde en laatste aanpassing van de CustomLogin plug-in was noodzakelijk, zodat de plug-in kon worden gebruikt in meerdere projecten. De eerste stap was dat de overbodige velden in het content verwijderd moesten worden. Dit waren de volgende velden:

* Website, het veld website werd alleen gebruikt voor het opzoeken van de gebruiker. Maar de gebruiker kan ook gevonden worden wanneer het emailadres uniek is.
* Gebruikersnaam, aangezien het emailadres uniek is gemaakt, is het overbodig om ook een gebruikersnaam te gebruiken voor het inloggen. Hiervoor wordt nu het unieke emailadres gebruikt.
* dotCMSUser, de dotCMSUser wordt voor elke gebruiker hetzelfde, waardoor de dotCMSUser net zo goed in de configuratie kan worden gezet.
* Salt, de salt wordt, evenals de dotCMSUser, voor elke gebruiker hetzelfde, waardoor deze net zo goed in de configuratie kan worden gezet.
* Passwordreset, deze optie blokkeerde de gebruiker als er om een nieuw wachtwoord was gevraagd. Hierdoor kon er niet meer ingelogd worden als het oude wachtwoord weer bekend werd. Het moet echter wel mogelijk zijn om in te loggen wanneer het oude wachtwoord weer bekend is, daarom werd deze optie overbodig.

Het verwijderen van de overbodige velden leverde geen problemen op. Het kosten wel tijd om alle scenario’s goed te testen, aangezien er een aantal essentiële velden niet meer nodig waren. De volgende stap was dat de plug-in gebruik ging maken van de ISAAC Mail Service. Voordat dit werd gedaan, moest er eerst worden uitgezocht hoe de ISAAC Mail Service exact werkt. Aangezien er nog geen ervaring met Mail Services was, werden er eerste enkele testapplicaties gemaakt, waarin verschillende scenario’s werden getest aan de hand van de handleiding. Nadat de handleiding succesvol was doorlopen werd de ISAAC Mail Service toegevoegd aan de CustomLogin plug-in.

Een volgende aanpassing was dat de CustomLogin plug-in de mogelijkheid krijgt, dat er een gebruiker kan worden toegevoegd door middel van de ‘voeg content toe’ knop in dotCMS. Dit was alleen niet zomaar mogelijk, aangezien het wachtwoord geencrypt moet worden en het emailadres moet worden gecontroleerd of deze uniek is. Na overleg met Xander bleek hiervoor een class te zijn in de dotCMS API, genaamd ContentletAPIPreHookAbstractImp. Als eerste moest er worden uitgezocht welke methodes dotCMS gebruikt om content op te slaan*.* Dit werd gedaan door alle methodes uit de ContentletAPIPreHookAbstractImp class te implementeren en aan elke methode een log regel toe te voegen. Nadat er een gebruiker was toegevoegd als content, werd in het log bestand duidelijk welke methode er gebruikt werd. Door deze hook was het mogelijk om het ingevulde wachtwoord te hashen. Het hashen van een wachtwoord houdt in dat deze door middel van een wiskundige algoritme wordt versleuteld, zodat het niet meer leesbaar is.

Een andere eis was dat enkele velden in het content onzichtbaar moesten zijn. Dit was alleen met de instellingen van dotCMS niet mogelijk. Omdat het wachtwoord gehasht werd, was het mogelijk dat de gehashte tekst zolang was dat het niet in de hidden field paste. Waardoor gebruik van de textarea noodzakelijk was, echter was het niet mogelijk om de textarea onzichtbaar te maken. De oplossing werd geboden door Xander, die al eerder een plug-in had gemaakt, waarmee het mogelijk was om elk veld onzichtbaar te maken.

De laatste aanpassing het dynamisch maken van het aantal keer dat er foutief ingelogd kan worden. Momenteel stond dit hardcoded in de Java code, waardoor er geen invloed op was. Door een kleine aanpassing, werd het aantal keer dat er foutief ingelogd mocht worden in de configuratie gezet. Hierdoor is het voor de beheerder mogelijk om in de configuratie in te stellen hoe vaak een gebruiker foutief mag inloggen voordat het account wordt geblokkeerd.

Meer informatie over het ontwikkeling van de CustomLogin plug-in is te vinden in bijlage B.

# Onderzoek

In dit hoofdstuk staan de resultaten van het onderzoek dat tijdens de afstudeerstage is uitgevoerd.

## Onderzoeksvraag

Het doel van het onderzoek is om erachter te komen hoe het beste een plug-in kan worden getest in een automatische testomgeving. De onderzoeksvraag die naar voren is gekomen naar aanleiding van de opdracht, is de volgende:

Hoe kan het beste een plug-in van een content management systeem automatisch worden getest?

## Deelvragen

Om de onderzoeksvraag meer kracht bij te zetten zijn er deelvragen opgesteld. Deze deelvragen geven samen het antwoord op de onderzoeksvraag.

1. Wat is een plug-in?
2. Wat is een content management systeem?
3. Wat is het testen van plug-ins?
4. Wat is een automatische testomgeving?
5. Wanneer een automatische testomgeving te gebruiken?

## Uitwerking deelvragen

In deze sectie worden de deelvragen uitgewerkt.

### Wat is een plug-in?

Een plug-in is een aanvulling op een computerprogramma, in dit geval dus een aanvulling op het Content Management Systeem. Het zorgt ervoor dat het software programma een functionaliteit krijgt die het op dat moment nog niet bezit. Zo kan er een Adobe Acrobat plug-in worden toegevoegd aan Mozilla Firefox zodat er in de browser pdf bestanden kunnen worden gelezen. Terwijl Mozilla Firefox deze functionaliteit van zichzelf niet ondersteunt.

### Wat is een content management systeem?

Een Content Management Systeem, afgekort CMS, is een systeem dat het proces van beheren, presenteren en publiceren van content voor internet en intranet sites automatiseert. Het belangrijkste kenmerk van een content management system is dat deze een scheiding realiseert tussen het content zelf en de presentatie van het content. De vormgeving en structuur van een site en de presentatie van informatie worden bij een content management system vooraf eenmalig vastgelegd in templates. Het beheer van de site bestaat daarna enkel nog uit het toevoegen, wijzigen of verwijderen van het content. Het content management system zorgt ervoor dat het content automatisch op de juiste plaats en in de juiste vorm op de site gepubliceerd wordt.

### Wat is het testen van plug-ins?

Het testen van software (plug-ins) houdt in dat er wordt gecontroleerd of de applicatie datgene doet wat er volgens de requirements wordt verwacht. Dit is noodzakelijk omdat het praktisch gezien onmogelijk is om foutloze code te schrijven voor de complexe software applicaties van deze tijd. Het probleem van testen is alleen dat het tijd en geld kost, daarom moet er, voordat er wordt begonnen met testen, goed worden nagedacht wat er allemaal getest moet worden. Zo moet de test niet te klein zijn, want dan kunnen niet alle requirements worden aangetoond. Maar de test kan ook niet te uitgebreid zijn want dan kost het te veel tijd en dus geld.

Toch kan een bedrijf zich afvragen waarom er dan getest moet worden als het zoveel geld kost. Er wordt getest om fouten uit de software te halen. Als de maker van het software de software test heeft de maker het bewijs dat het gemaakte software volgens de geteste scenario’s foutloos zijn. Dit voorkomt onnodige kosten, aangezien het goedkoper is om tijdens de ontwikkelingsperiode de code nog aan te passen dan dat het al in productieomgeving is. Echter is het in dit geval zo dat de software zo goed is als de kwaliteit van de gemaakte testen.

**Teststrategieën**Software test methodes zijn oorspronkelijk verdeeld in twee manieren, namelijk whiteboxtest en blackboxtest. Deze twee manieren beschrijven vanuit welk oogpunt de tester de testen maakt en uitvoert.

**Whiteboxtest**De whiteboxtest is een strategie waarbij de tester kennis heeft van de code van het te testen programma of systeem. Tijdens het testen met behulp van deze strategie mag de tester de code van het programma aanpassen. Vaak worden er bij het constateren van een fout print-statements toegevoegd aan de code. Dit wordt gedaan zodat de tester elke stap kan volgen dat in het software programma wordt doorlopen. Hiermee is het voor de tester gemakkelijker om te zien waar het verkeerd gaat en waarom dit is. Hierbij worden zoveel mogelijk scenario’s doorlopen zodat het zo goed als zeker is dat de geschreven code klopt. Whitboxtesten wordt vaak toegepast bij unittesten en integratietesten.

**Blackboxtest**Bij blackboxtesten wordt er door een tester een systeem getest waarbij er geen of beperkte kennis van de code is. Bij deze test probeert de gebruiker allerlei input te geven wat zowel goed als fout is. De goede maar ook foute input hoort het programma goed te verwerken. Bijvoorbeeld in een rekeningprogramma waarbij er een bedrag moet worden ingevuld, mogen er maar maximaal twee decimalen aanwezig zijn. Toch test de gebruiker niet alleen op de waarde 1,99 maar ook op de waardes 1,20,11, ABC en 0,00001. Alleen in het geval met de waarde 1,99 mag het goed gaan. In de overige gevallen hoort er een nette foutmelding te komen dat er onjuiste input is gegeven. Deze manier van testen wordt vaak toegepast bij systeemtesten en acceptatietesten.

**Testlevels**Er zijn een aantal levels voor het testen van een software applicatie/systeem. De volgende testlevels zullen in deze scriptie worden besproken: unittesten, integratietesten, systeemtesten en acceptatietesten.

**Unittesten**Bij unittesten worden er stukjes code (units) afzonderlijk van elkaar getest. Voor het testen van iedere unit worden er meerdere tests ontwikkeld. Hierbij worden dan verschillende scenario’s doorlopen om alle facetten van de unit mee te maken. Vaak worden hiervoor mockobjecten gemaakt om zo de verschillende tests gescheiden te houden. Een mockobject is een tijdelijk object wat dezelfde eigenschappen heeft als een echt object om de functionaliteiten daarvan te simuleren. Het uitvoeren va de unittesten zijn de taak van de softwareontwikkelaar of een testteam.

**Integratietest**Bij de integratietest worden de individuele units verbonden en als geheel getest. Het doel van de integratietest is zekerheid te krijgen over de functionaliteit, performance en betrouwbaarheid die van de units gevraagd wordt. Voordat deze fase aan bod komt is het wel vereist dat de individuele units de unittesten succesvol hebben doorstaan.

**Systeemtest**  
Bij de systeemtest wordt het volledig geïntegreerde systeem getest, waarbij wordt gekeken of het voldoet aan de requirements. Om deze test uit te voeren is er geen kennis vereist van de code van het te testen systeem. Het doel is om de onderdelen gezamenlijk te testen, hier wordt vooral gecontroleerd wat in de vorige test nog niet aan de orde is gekomen.

**Acceptatietest**Het doel van de acceptatietest is om te kijken of het systeem acceptabel is. Vaak wordt deze test dan ook uitgevoerd door de eindgebruiker om te kijken of hij het systeem accepteert. Dit kan door de eindgebruiker worden gedaan omdat de tester geen kennis hoeft te hebben van het te testen systeem. Tijdens de acceptatietest komen soms ook zaken naar boven die door de unittesten niet werd opgevangen. Dit is dan ook de laatste test om te zien of alles naar behoren functioneert en of de gebruiker het systeem accepteert.

**Belangrijk bij het testen van een plug-in**Om een plug-in te testen wordt er onderscheid gemaakt op twee niveaus. Namelijk het testen op compile niveau en op deploy niveau. Het testen op compile niveau houdt in dat de plug-in wordt getest op de onafhankelijk code, dus door middel van unittesten. Het testen op deploy niveau houdt in dat de plug-in in het content management systeem wordt gedeployed waarbij de testen vooral zullen draaien om de methodes die gebruiken maken van de libraries die in het content management systeem zitten. Op compile niveau wordt er getest met unittesten en integratietesten en op deploy niveau wordt er getest met systeemtesten.

### Wat is een automatische testomgeving

Om het automatisch testen eenvoudiger te maken, zijn er verschillende mogelijkheden om een automatische (test)build omgeving op te zetten. Er is echter wel een verschil tussen automatische build tools en automatische build omgevingen (ook wel continuous integration tools genoemd). De automatische build tools zorgen ervoor dat het project automatisch wordt gebuild door middel van bijvoorbeeld batch of Ant bestanden. De continuous integration tool daarentegen loopt door enkele events wanneer deze wordt getriggerd. De events kunnen uit allerlei stappen bestaan, zoals het uitvoeren van Ant en batch bestanden, het verwerken van gegevens in een grafiek en het automatisch starten van een volgende job. Zo kan er gebruik worden gemaakt van het automatisch ophalen van het project via een versiesysteem, zoals Subversion. Hiermee kan de continuous integration tool het project op een gewenste plaats uitchecken en daar zijn job op laten draaien. Deze job kan bestaan uit het uitvoeren van een ANT Task, batch command, shell command etc. Zo zou de job dus ook door middel van een Ant Task unittesten uit kunnen laten voeren en daarvan een JUnit testrapport kunnen genereren. Nadat deze taak is uitgevoerd kan er in de job worden ingesteld waar het rapport zich bevindt zodat deze automatisch worden verwerkt in een grafiek. De resultaten van de job kunnen per e-mail ontvangen worden, zodat er niet elke keer naar de server hoeft worden gegaan. En zo zijn er nog tal van andere mogelijkheden die een continuous integration tool biedt. Vooral door de plug-ins zijn de mogelijkheden bijna eindeloos. De stappen die in een continuous integration tool worden uitgevoerd, kunnen ook worden gemaakt in een batch of Ant bestand. Echter is hier veel meer kennis van deze scripttaal nodig, dan wanneer er een continuous integration tool wordt gebruikt.  
  
Hieronder worden de vier meest toonaangevende continuous integration tools besproken. De keuze voor deze vier continuous integration tools is gebaseerd op de CI Feature Matrix en het artikel Continuous Integraton Server Comparison 2008 van Chris Read. De tools die in dit onderzoek besproken worden, zijn: TeamCity van JetBrains, Bamboo van Atlassian, CruiseControl van ThoughtWorks en het open source Jenkins wat is gesplitst van Hudson.

**TeamCity**TeamCity is een commercieel product dat gemaakt is door JetBrains. TeamCity is te verkrijgen in drie edities, namelijk Professioneel, Enterprise en open-source. De professionele editie heeft de beperking dat er maar 20 gebruikers en 20 build configuraties kunnen worden aangemaakt. Bij het product zijn er drie build agents beschikbaar, echter kunnen deze nog worden bijgekocht. Voor de Enterprise editie geldt dat er geen beperkingen zijn voor het aantal gebruikers en build configuraties. Echter kost deze Enterprise editie 1899 euro. De open-source versie bevat alle functionaliteiten gratis, echter verstrekt JetBrains alleen open-source licenties aan bedrijven die het gebruiken voor niet-commerciële software projecten. TeamCity bevat, zoals verwacht mag worden van een commercieel product, alle basisfuncties die een continuous integration server hoort te hebben. Het is eenvoudig in te richten zonder dat er door allerlei ingewikkelde stappen hoeft te worden gelopen.

**Bamboo**Bamboo is een commercieel product dat gemaakt is door Atlassian. Het eerste wat opvalt, is dat er voor deze continuous integration server geen gratis versie is. De goedkoopste versie kost 10 dollar in de maand, daar krijgt de koper10 jobs en één agent om de jobs uit te voeren voor. De versie die zou voldoen om plug-ins te testen kost 50 dollar in de maand waarbij er onbeperkt jobs aangemaakt kunnen worden die te gebruiken zijn door één agent. Echter is er een 30 dagen trial versie te verkrijgen waarmee getest is wat de functionaliteiten zijn van de Bamboo continuous integration server. Om gebruik te maken van de trial versie dient men zich eerst te registreren om de trial versie sleutel aan te vragen. Andere nadelen van Bamboo zijn dat de werkdirectory voor alle jobs hetzelfde is en om een job te configureren moeten er meerdere tabbladen worden doorlopen. Al met al is de conclusie over Bamboo dat het ingewikkeld is om een job aan te maken in tegenstelling tot andere continuous integration servers. Daarbij komt ook nog eens kijken dat er voor deze CI-server moet worden betaald, waar er betere en goedkopere opties zijn.

**Jenkins**Jenkins is een open source continuous integration server dat is begonnen als een hobby van Kohsuke Kawaguchi terwijl hij bij Sun werkte. Jenkins is open source, dus er zijn geen kosten aan verbonden. Verder heeft het geen beperkingen wat betreft het aantal jobs en heeft het een groot aantal plug-ins waardoor het eigenlijk alles kan doen waarvoor een continuous integration server is gemaakt. Het enige wat Jenkins niet heeft tegenover Bamboo is de integratie met confluence, dit is echter niet van toegevoegde waarde voor het testen van plug-ins. Omdat Jenkins kosteloos is en een grote hoeveelheid van plug-ins heeft is het de ideale continuous integration server om te gebruiken voor het testen van plug-ins.

**CruiseControl**CruiseControl is een open source continuous integration server ontwikkeld door een groep vrijwilligers. CruiseControl viel oorspronkelijk onder ThoughtWorks, deze is echter verder gegaan met een betaalde versie genaamd Cruise. CruiseControl heeft speciale versies voor .NET applicaties genaamd CruiseControl.NET en voor Ruby applicaties genaamd CruiseControl.rb. CruiseControl beschikt net als de ander genoemde continuous integration servers over alle functionaliteiten die een continuous integration server zou moeten bezitten. Echter is de online handleiding van CruiseControl erg vaak niet bereikbaar waardoor het lastig is informatie op te zoeken bij het inrichten van de omgeving. Dit is dan ook de reden waarom CruiseControl niet wordt meegenomen in de vergelijkingstabel. Aangezien support belangrijk is voor een bedrijf die een testomgeving wil gebruiken, waarbij de handleiding misschien nog wel de aller belangrijkste support is.

**Vergelijkingstabel - continuous integration servers**  
In figuur 3 worden er enkele functionaliteiten vergeleken van de drie bovenstaande continuous integration servers, Bamboo, Jenkins en TeamCity. Let wel op dat deze lijst is gemaakt op basis van 1 maart 2012, het kan dus zijn dat de status van een van deze functies is veranderd. Verder worden alleen de functionaliteiten besproken die relevant zijn voor het testen van plug-ins.

Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif - Bevat de functie http://confluence.public.thoughtworks.org/images/icons/emoticons/error.gif - Bevat de functie niet

|  |  |  |  |
| --- | --- | --- | --- |
|  | Bamboo | Jenkins | TeamCity |
| Open Source / Gratis | http://confluence.public.thoughtworks.org/images/icons/emoticons/error.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/error.gif |
| Subversion | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| SCM filtering\* | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Parrallel bouwen | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Gebruikersbeveiliging | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| LDAP integratie | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Publiceer Email | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Publiceer RSS feed | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Ant builder | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Commandline builder | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Maven | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/error.gif |
| Confluence integratie | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/error.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/error.gif |
| JIRA integratie | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | Beschrijving: http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| JUnit resultaten | http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |
| Selenium resultaten | http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif | http://confluence.public.thoughtworks.org/images/icons/emoticons/check.gif |

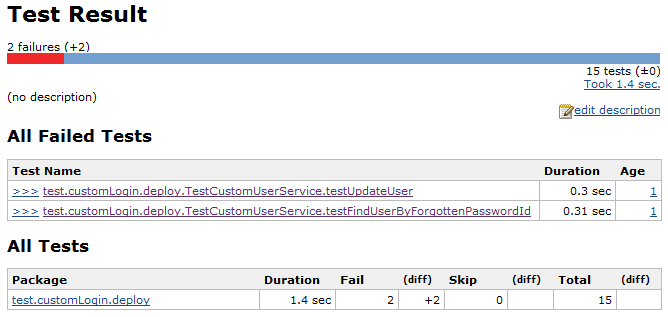
Figuur 3: Vergelijkingstabel van de continuous integration servers: Bamboo, Jenkins en TeamCity  
\* SCM filtering is het automatisch builden wanneer er een nieuwe versie is van het project

Op basis van de bovenstaande tabel kunnen we stellen dat de keuze zal gaan tussen Bamboo en Jenkins. Deze twee continuous integration servers hebben de meeste functionaliteiten, echter kost Bamboo geld en is Jenkins open-source en gratis. Verder heeft Jenkins een gebruiksvriendelijker interface en de installatie is verreweg het makkelijkste van de bovengenoemde continuous integration servers. Dus Jenkins is het meest geschikt voor dit project. Daarom vervolgen we dit onderzoek met enkele voorbeelden van plug-ins die te gebruiken zijn in Jenkins.

**Plug-ins binnen Jenkins**De kracht van Jenkins zit hem vooral in de plug-ins die voor Jenkins beschikbaar zijn. Met deze plug-ins komen er meer statistieken beschikbaar van de acties die zijn uitgevoerd. Er worden hier enkele plug-ins besproken om een indicatie te geven wat er verwacht kan worden van de plug-ins binnen Jenkins.

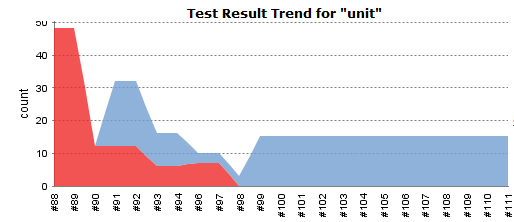
**JUnit Attachments Plug-in**De JUnit Attachments Plug-in zorgt ervoor dat de XML bestanden van de uitgevoerde JUnit testen binnen Jenkins getoond kunnen worden. De resultaten worden getoond in zowel een grafiek als in een rapport.

In het onderstaande figuur 4 staan de resultaten van de gefaalde testen. De naam van de test geeft aan in welke klasse dit was en hoe deze methode heet. Bij het klikken op de naam van de test toont deze de exacte fout, het resultaat wat er werd verwacht en het resultaat wat eruit is gekomen. Daaronder staat de package waarin de tests zitten die zijn uitgevoerd, hoeveel tijd dit heeft gekost, het aantal gefaalde testen en het totaal aantal testen wat er getest is. Hierdoor komt er een duidelijk overzicht wat er precies is misgegaan en vooral waar het is misgegaan.



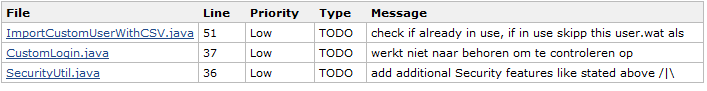
Figuur 4: Unittestresultaten in Jenkins

Zoals in het begin van deze sectie aangegeven kan er door middel van een Ant Task unittesten worden uitgevoerd en daar automatisch een testrapport van laten genereren. Dit rapport kan Jenkins uitlezen en de resultaten tonen in een grafiek zoals in figuur 5.



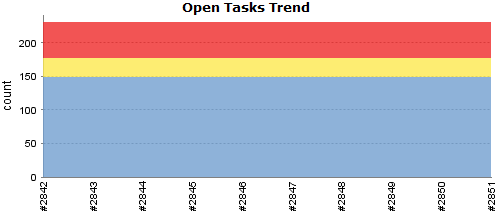
Figuur 5: Unittestresultaten verwerkt in een grafiek binnen Jenkins

**Task Scanner Plug-in**De Task Scanner Plug-in scant het project op open tasks. Tasks zijn taken die nog moeten gebeuren in het project. Zo wordt de plug-in geconfigureerd op prioriteiten waaraan een woord wordt gelinkt. Zo is er de hoge prioriteit: FIXME, normale prioriteit: XXX en bij lage prioriteit: TODO. De gevonden resultaten worden in een tabel, zoals in figuur 6, gezet waarbij wordt aangegeven in welk bestand de prioriteit is, welke lijn in het bestand, wat de prioriteit is, het type prioriteit en het bericht wat bij de prioriteit hoort.



Figuur 6: Open tasks resultaten in Jenkins

Alle resultaten worden ook weergegeven in een grafiek, zoals in figuur 7. Waarbij het rode gedeelte de hoge prioriteiten aangeeft, het gele de normale prioriteiten en het blauwe de lage prioriteiten. Het is echter wel zo dat er niet meer dan 200 hoge prioriteiten zijn in dit geval maar 60 (230-170). Want de totale hoogte van de grafiek zijn alle prioriteiten bij elkaar opgeteld.



Figuur 7: Open tasks trend resultaten in Jenkins

**Jenkins Workspace Cleanup Plugin**Deze plug-in zorgt ervoor dat het project, in dit geval de plug-in voor het content management systeem, automatisch wordt verwijderd na de job. Dit is in het geval voor het testen van plug-ins erg handig. Gezien het builden van plug-ins in het content management systeem niet per plug-in gebeurt maar alle plug-ins direct worden gebuild die op dat moment in de plug-ins directory staan. Mocht er dan één plug-in niet juist builden dan zullen de daar op volgende jobs ook foutief verlopen.

### Wanneer een automatische testomgeving te gebruiken?

Voor ICT bedrijven is het duidelijk dat handmatig testen vaak meer tijd kost dan dat er beschikbaar is. Daarom kan een automatische testomgeving de effectiviteit en efficiëntie verhogen. In dit gedeelte wordt uitgelegd wanneer het nuttig is om een testomgeving te gebruiken en wat de voor- en nadelen zijn van een testomgeving.

**Wanneer is het hebben van een automatisch testsysteem nuttig?**Deze zijn alleen nuttig in de volgende gevallen:

* Als er software is ontwikkeld voor meerdere versies. Hiervoor hoeft maar éénmalig een testscript worden geschreven die in de toekomst bij nieuwere versies wederom kan worden gebruikt om te kijken of de software ook in de nieuwe versie werkt.
* Elke keer als er een wijziging in de code is gemaakt en de testers moeten handmatig testen of de overige code hierdoor nog juist functioneert.
* Geautomatiseerde testen kunnen voor meer nauwkeurigheid zorgen bij het testen. Zo kan er een klein kleurverschil in de GUI over het hoofd gezien worden, terwijl de automatische testen deze wel op zal pikken.
* In een complexe omgeving waar veel scenario’s moeten worden onderzocht, kan geautomatiseerd testen veel sneller verschillende scenario’s ondergaan met een simpel testscript.

**De voordelen van een automatisch testsysteem**Mocht het ICT bedrijf aan één van de voorgaande gevallen voldoen dan moet er worden gekeken naar de voor- en nadelen van automatisch testen. De voordelen van automatisch testen zijn als volgt:

* Voert test cases sneller uit: Automatisch testen kan een test in sommige gevallen met wel 95% versnellen.
* Eenmaal een test script gemaakt voor een automatische testomgeving dan kan deze worden opgeslagen en worden hergebruikt.
* Bij een automatische testomgeving wordt er uiteindelijk een rapport gemaakt met daarin de testresultaten. Hierin is precies te zien welke testen succesvol zijn verlopen en bij welke test een fout is opgetreden en wat deze fout was.
* Er is minder tijd nodig om de testen uit te voeren, ook nadat er een bug moest worden opgelost, kunnen er nogmaals de automatische testen worden gebruikt om te controleren of het werkt.

**De nadelen van een automatisch testsysteem**Maar waar er voordelen zijn, zijn er ook nadelen voor het automatisch testen van software. De nadelen van automatisch testen zijn als volgt:

* Ook al kan automatisch testen voor veel applicaties worden gebruikt, het dekt niet alle code. Want er zijn in sommige gevallen erg complexe testcases die niet kunnen worden getest door automatische testen. Hierdoor zal het handmatig testen blijven bestaan.
* Geautomatiseerde testtools kunnen kostbaar zijn, dit is afhankelijk van de grootte, support en functionaliteit.
* Sommige automatische testtools kunnen zo complex zijn dat daar een speciale training voor nodig is. De testers zullen niet alleen kennis moeten hebben van de tool maar ook van het test proces. Dit kost het bedrijf extra tijd en geld.
* Het succes van geautomatiseerde testen is afhankelijk van duidelijke eisen en zorgvuldig ontwikkelde testcases. Want elk scenario en applicatie is anders en een automatische test tool schrijft zelf geen testen. Dit blijft een handmatig proces.

Uiteindelijk is het voor een ICT bedrijf belangrijk om deze voor- en nadelen tegen elkaar af te wegen. Gezien automatisch testen in vele gevallen erg duur kan zijn, moet het wel rendabel zijn om de plug-ins te gaan testen met een automatische testomgeving.

# Uitvoering

In dit hoofdstuk wordt de uitvoering van het project na het onderzoek beschreven.

## Automatisch testen plug-ins met batch script

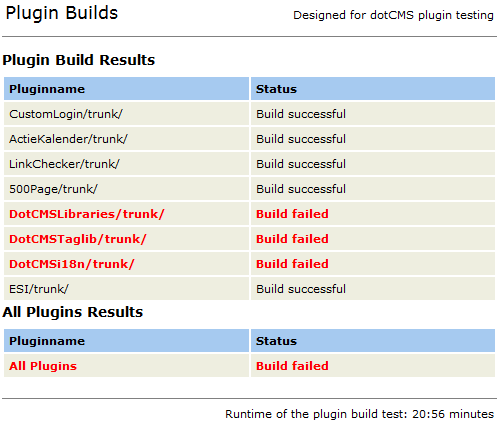
De eerste stap van de uitvoering was het schrijven van een batch bestand voor het automatisch testen of een plug-in correct was gedeployed. Deze stap is er bewust tussen gezet, zodat er precies uitgezocht kon worden wat er allemaal nodig is voor het deployen en verwijderen van een plug-in. Eerst moest er worden uitgezocht wat een batch bestand precies is en hoe deze is opgebouwd. Het bleek dat er veel informatie over batch bestanden te vinden was. Waardoor er ook genoeg mogelijkheden waren om een goed script te schrijven voor het testen of een plug-in deployed.

Ten tweede moest er een script worden ontwikkeld waarbij de plug-in van Subversion werd gehaald. Deze plug-ins moesten automatisch in de plugins directory van dotCMS worden gezet. Dit bleek echter een probleem te zijn, omdat er geen script is waarbij direct de plug-in van Subversion kan worden gehaald. Hiervoor moest een aparte Java class worden geschreven (zie figuur 8) wat samen met de svnkit library wordt aangeroepen in het batch bestand. Aan de main methode wordt de URL meegegeven van de plug-in locatie in Subversion. Deze URL’s staan in een apart bestand onder elkaar en worden in het batch bestand één voor één uitgelezen. Van deze URL wordt het stuk eruit gehaald waar de naam van de plug-in instaat. Deze wordt gebruikt om de locatie waar de plug-in moet komen te staan te maken. Hierna wordt er gecontroleerd of de ingevulde gebruikersnaam en wachtwoord correct zijn. Vervolgens vindt de checkout van de plug-in plaats.

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **try** {  String url = args[0];  String[] links = url.split("/");  File dstPath = **new** File(System.getenv("DOTCMS\_HOME")+ “/plugins/"+links[5]);  SVNURL urlSVN;  urlSVN = SVNURL.parseURIEncoded(url);  ISVNAuthenticationManager authManager = . SVNWCUtil.createDefaultAuthenticationManager("gebruikersnaam”, "wachtwoord");  SVNUpdateClient uc = **new** SVNUpdateClient(authManager , **null**);  uc.doCheckout(urlSVN, dstPath, SVNRevision.UNDEFINED, SVNRevision.HEAD, . SVNDepth.INFINITY, **true**);  } **catch** (SVNException e) {  e.printStackTrace();  }  } |

Figuur 8: Java code voor het ophalen van plug-ins via Subversion.  
  
De volgende stap was het deployen van de plug-in. Dit gebeurde door het batch bestand aan te roepen wat dotCMS zelf gebruikt voor het deployen van plug-ins. Na deze stap moest er worden gecontroleerd of de plug-in correct was gedeployed. Dit gebeurde door middel van het doorzoeken van het dotCMS log bestand. In dit bestand staat of de plug-in succesvol is gedeployed of dat het deployen is gefaald. Deze stap werd uitgevoerd met de Findstr methode, aan deze methode wordt een stuk tekst meegegeven wat de methode gaat zoeken in het log bestand. Het gevonden resultaat wordt weergegeven.

Het was echter de bedoeling dat de resultaten in een rapport kwamen te staan. Maar omdat er geen standaard testmethode wordt gebruikt, zoals bijvoorbeeld JUnit, dient er een html rapport te worden gecreëerd in het batch bestand. Hierin werd letterlijk het html gezet wat er uiteindelijk uit moest komen. Alleen de resultaten van het controleren of het deployen correct was verlopen is dynamisch, zie figuur 9.

   
Figuur 9: Deel van het HTML rapport gecreëerd door het batch bestand

Nadat het rapport is gecreëerd, is de volgende stap het undeployen van de plug-in. Deze stap is net zoals het deployen van de plug-in, het aanroepen van het al reeds bestaand batch bestand in dotCMS. Na deze stap wordt de plug-in verwijderd. Dit wordt voor alle plug-ins gedaan die in het bestand onder elkaar staan.

Nadat dit voor elke plug-in was gedaan, worden alle plug-ins tegelijkertijd van Subversion gehaald en in de juiste folder gezet. Waarna alle plug-ins worden gedeployed en er wederom wordt gecontroleerd of dit correct verliep.   
  
Het volledige batch bestand is te vinden in bijlage C. Echter wordt dit batch bestand niet meer gebruikt, omdat Jenkins verbeterde functionaliteiten heeft. Het kan hierdoor voorkomen dat het batch bestand niet meer correct werkt of up-to-date is.

## Inrichten van een testomgeving – Jenkins

Nadat er voldoende kennis was over het automatisch deployen van een plug-in, kon er verder worden gegaan met de volgende stap in de planning. Echter staat er in de planning aangegeven dat er eerst testen worden geschreven voordat er wordt begonnen aan de inrichting van de testomgeving. Dit is echter na overleg met Xander en Jan-Willem besloten om dit om te draaien. Dit had als reden dat de verwachting was dat het inrichten van de testomgeving problemen op zou leveren, aangezien er nog maar kortstondig met Jenkins was gewerkt tijdens de periode op school. Terwijl het maken van JUnit testen geen probleem op zou moeten leveren, aangezien dit al vaker was gedaan in het verleden. Daarnaast moest er worden gekeken naar het formaat van testrapporten wat Jenkins accepteert, waardoor het zonde zou zijn als er in eerste instantie een verkeerde formaat testrapporten wordt gebruikt. Echter zijn er in de plug-ins wel JUnit testen aanwezig, deze bevatten alleen geen echte testen, maar stub testen. Dit houdt in dat deze testen alleen een resultaat geven zonder daadwerkelijk methodes te testen, zodat er tijdens het inrichten van de testomgeving toch getest kon worden met testen.

Het installeren en configureren van Jenkins was een goed te volgen proces. Voor het installeren was geen of nauwelijks kennis over continuous integration servers nodig, waardoor de Jenkins omgeving in zeer korte tijd al lokaal op de PC draaide.

Voor het testen van de plug-ins moesten er twee soorten testen komen. Deze werden geconfigureerd in zogenoemde jobs. De eerste job is voor het testen op compile niveau. In deze job wordt er gecontroleerd of de plug-in kan builden en er worden testen uitgevoerd waarbij de testen geen gebruik maken van de dotCMS libraries, voor meer informatie zie secties 6.3 en 6.4. De tweede job is voor het testen op deploy niveau. Tijdens de job wordt er gecontroleerd of de plug-in kan deployen in dotCMS. Deployen betekent het invoegen van de plug-in in dotCMS. Verder worden de testen uitgevoerd die gebruik maken van de dotCMS libraries, voor meer informatie over de deploy jobs zie secties 6.5 en 6.6.

## Inrichting testomgeving voor het testen op compile niveau (1)

Voor het inrichten van de testomgeving voor het compileren, werd er eerst met Xander en Jan-Willem besproken wat deze inrichting precies inhield. De stappen die voor het testen van de plug-in op compile niveau moesten worden uitgevoerd zijn weergegeven in figuur 10 en worden vervolgens uitvoerig behandeld. Echter voordat de plug-in in de testomgeving komt te staan dienen er eerst JUnit testen te worden geschreven. De JUnit testen controleren of stukken code naar verwachting werken, dit is onderdeel van de unit- en integratietesten. Deze testen komen in de source map in de test/naam\_plugin/compile folder.   
Figuur 10: De inrichting van de testomgeving voor het testen van de plug-in op compile niveau

Testresultaten publiceren

Ophalen plug-in

Builden plug-in

Testen uitvoeren

Verwijderen plug-in

**Ophalen plug-in:** Het ophalen van de plug-in gebeurt door middel van de Subversion Plugin. In de configuratie wordt door middel van een URL aangegeven waar de plug-in staat in Subversion en waar de plug-in neer moet worden gezet, in dit geval de plugins directory in dotCMS. Subversion is een versiebeheersysteem, waarbij elke versie van het project wordt bewaard.   
  
**Builden plug-in:** Voor het builden van de plug-in wordt er de Ant Task compile aangeroepen die al is geconfigureerd in dotCMS.   
 **Testen uitvoeren:** Voor het uitvoeren van de testen op compile niveau moest er een Ant script worden geschreven. Dit was in eerste instantie lastig, aangezien er geen kennis en ervaring was wat betreft Ant scripts. De eerste stap die moest worden gezet was het uitzoeken hoe Ant scripts zijn opgebouwd. Nadat er meer kennis was opgedaan over Ant scripts, werd er begonnen met het maken van een script voor het testen op compile niveau. Hieruit is het script gekomen wat in figuur 11 te zien is. In dit Ant script wordt gezocht naar de zojuist gebuilde testclasses. De gevonden testclasses worden uitgevoerd en verwerkt in een JUnit rapport wat in de map van de plug-in komt te staan onder Reports.

|  |
| --- |
| **<property name="test.classes" value="\*/build/classes/"/>**  **<property name="test.reports" value="\*/Report"/>**  **<target name="test-compile">**  **<mkdir dir="${test.reports}"/>**  **<junit fork="no" printsummary="no" haltonfailure="no">**  **<batchtest fork="no" todir="${test.reports}">**  **<fileset dir="${test.classes}">**  **<include name="test/\*/compile/\*.class"/>**  **</fileset>**  **</batchtest>**  **<formatter type="xml"/>**  **<classpath id="test.classpath">**  **<pathelement location="${test.classes}"/>**  **<pathelement location="C:\Configuratie\junit-4.10.jar"/>**  **</classpath>**  **</junit>**  **</target>** |

Figuur 11: Ant script voor het testen op compile niveau  
  
**Testresultaten publiceren:** De testresultaten worden in Jenkins automatisch in een nette grafiek weergeven door middel van de Publish JUnit test result report optie. Het enige wat hiervoor moet worden geconfigureerd is de locatie waar het testrapport moet komen te staat.  
 **Verwijderen plug-in:** Het verwijderen van de plug-in is noodzakelijk, aangezien deze in de plugins directory van dotCMS staat. Wordt deze niet verwijderd dan wordt er bij het deployen van een andere plug-in deze automatisch mee gedeployed. Voor het verwijderen van de plug-in wordt er de Jenkins Workspace Cleanup Plugin gebruikt. Bij het configureren van deze plug-in kan er exact worden aangegeven welke folders en bestanden er verwijderd moeten worden.

## Inrichting testomgeving voor het testen op compile niveau (2)

De eerste versie voor het testen van plug-ins op compile niveau werd gepresenteerd aan het dotCMS expertiseteam binnen ISAAC. Uit deze meeting met het dotCMS expertiseteam kwamen enkele voorstellen en verbeterpunten naar voren. Hieronder zijn de voorstellen opgesomd die volgens het dotCMS expertiseteam prioriteit hadden. Deze voorstellen zijn verwerkt in de nieuwe versie, waardoor het uiteindelijke proces voor het testen van plug-ins op compile niveau er als volgt uitziet:   
  
  
Figuur 12: De verbeterde inrichting van de testomgeving voor het testen van de plug-in op compile niveau   
  
**Plug-in op eigen Jenkins workspace compileren**Een ander punt dat naar voren kwam tijdens de meeting met het expertiseteam was dat de algehele test sneller zou zijn als de plug-in op een eigen werkplaats zou staan. Hierdoor hoeft de plug-in maar eenmalig uitgecheckt te worden. Mocht er dan een nieuwere versie van de plug-in worden gecommit, dan hoeven alleen de aangepaste classes van Subversion te worden gehaald.  
  
**Automatisch starten van de testen bij het comitten op Subversion**In Jenkins is er de mogelijkheid om te controleren of er veranderingen plaats hebben gevonden in een specifieke plug-in, dit wordt ook wel pollen genoemd. Door deze optie te gebruiken wordt de plug-in automatisch getest wanneer er een nieuwere versie van de plug-in beschikbaar is in Subversion.

**Ant script Compile**

Ophalen / updaten plug-in

Builden plug-in

Testen uitvoeren

Testresultaten publiceren

Project controleren op open taken

Poll SCM

Elke 5 minuten controleren

**Testen lostrekken van de source folder**Tijdens de meeting met het expertiseteam is ook het idee geopperd om de testen en testlibraries buiten de source code te zetten. Dit zou als voordeel hebben dat de testen en testlibraries niet mee deployen bij een plug-in op productieniveau. Zou dit wel mogelijk zijn, dan zou het in theorie kunnen voorkomen dat er testen worden aangeroepen op productieniveau. Hierdoor zouden er problemen kunnen ontstaan. Na dit verder uitgezocht te hebben, bleek het inderdaad voordeliger te zijn om de testen in een aparte source folder te zetten. Om dit te realiseren moest de locatie van de testen in de plug-in worden aangepast. Verder diende alle scripts en configuratie instellingen die iets met de testen te maken hadden te worden veranderd.

**Algemeen Ant script schrijven voor het compileren**In een meeting met Jan-Willem werd de huidige opzet van het testen van de plug-in op compile niveau besproken. Uit deze meeting kwam naar voren dat het verstandiger was om één groot Ant script te schrijven in plaats van meerdere stappen te hebben in de configuratie van een job. Omdat alle jobs op compile niveau dezelfde stappen doorlopen, is het handiger wanneer er iets moet worden veranderd aan die stappen dat dit op één centrale plek gebeurt, in plaats van dat er bij meer dan 60 plug-ins handmatig iets aan de configuratie moet worden aangepast.

Het script voor het testen van de plug-in op compile niveau had één moeilijkheid. Het dotCMS expertiseteam had namelijk aangegeven dat het verstandiger zou zijn om de plug-in op zijn eigen werkplek neer te zetten in plaats van in de plugins directory van dotCMS. Hierdoor kon er geen gebruik worden gemaakt van het originele script dat normaal gesproken wordt gebruikt voor het compileren, omdat deze er vanuit gaat dat de plug-in in de plugins directory staat. Om het script zo te maken dat deze er vanuit gaat dat de plug-in op de eigen werkplek staat, diende er enkele variabele aangepast te worden.

Zo wordt er gebruik gemaakt van de workspace environment variabele uit Jenkins. Deze variabele geeft aan wat de werkplek locatie van de plug-in is. Door deze variabele voor elke property te zetten die iets te maken heeft met de locatie van de plug-in, was het probleem van de locatie opgelost. Alleen maakte het compile script ook gebruik van enkele mappen uit dotCMS, deze mappen kon het script alleen niet meer vinden omdat het script vanuit de werkplaats van de plug-in werkt. Daarom moest de volgende regel worden toegevoegd: basedir="C:\dotcms". Deze regel geeft aan dat het script vanuit C:\dotcms, de rootfolder van dotCMS, moet werken. Deze laatste toevoeging zorgde ervoor dat het script naar behoren werkt, echter moet het aanmaken van de variabelen in de goede volgorde gebeuren. Wordt er eerst gebruik gemaakt van een variabele voordat deze wordt aangemaakt, dan werkt deze variabele niet op de correcte wijze.

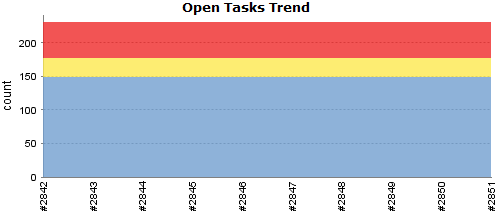
|  |  |
| --- | --- |
| Het Ant script wordt aangeroepen door middel van één Invoke Ant methode in Jenkins, zoals hier rechts te zien is in figuur 13. Aan deze methode wordt mee gegeven welke target uitgevoerd moet worden en wat de locatie van de build file is. In dit geval is er in de configuratie van Jenkins een variabele “configuratie” aangemaakt, waardoor er maar op één plaats staat aangegeven waar de configuratie bestanden staan. Dit zorgt ervoor, mits de naam van de build file niet veranderd, dat er nooit iets aan de build configuratie hoeft worden veranderd. | Figuur 13: Invoke Ant voor het starten van de compile.xml |

**Project controleren op open taken**De laatste verandering die is doorgevoerd tijdens deze fase is het controleren op open taken. Hiervoor wordt gebruik gemaakt van de Scan workspace for open tasks plugin. Het controleren op open taken gebeurt door middel van het zoeken naar drie specifiek geconfigureerde woorden. Elk van deze woorden hoort bij een bepaalde prioriteit. Deze woorden zijn als volgt geconfigureerd:

* Hoge prioriteit – FIXME
* Normale prioriteit – XXX
* Lage prioriteit – TODO

In de configuratie moet bovendien ook worden aangegeven welke bestanden er gecontroleerd moeten worden. In dit geval zijn het alle Java bestanden die in de plug-in zitten. De Scan workspace for open tasks plugin publiceert automatisch het resultaat in een grafiek zoals weergeven in figuur 14, waarbij de volgende kleuren voor de volgende prioriteiten staan:

* Rood – Hoge prioriteit
* Geel – Normale prioriteit
* Blauw – Lage prioriteit

Figuur 14: Totaal aantal open tasks verwerkt in een grafiek

## Inrichting testomgeving voor het testen op deploy niveau (1)

Voor het inrichten van de testomgeving voor het testen op deploy niveau, werd er net als bij het testen op compile niveau eerst met Xander en Jan-Willem besproken hoe de inrichting precies zou worden aangepakt. Voor het testen van de plug-in op deploy niveau moesten de in figuur 15 schematisch weergegeven stappen worden uitgevoerd. De testen op deploy niveau bevatten vooral integratietesten en testen op systeemniveau. De JUnit testen voor het testen op deploy niveau komen in de source folder test/naam\_plugin/deploy te staan.   
Figuur 15: De inrichting van de testomgeving voor het testen van de plug-in op deploy niveau   
  
**Ophalen plug-in:** Het ophalen van de plug-in gebeurt door middel van de Subversion Plugin. In de configuratie van de Subversion Plugin dienen twee zaken te worden geconfigureerd, namelijk de URL waarvan de plug-in afgehaald moet worden en de locatie waar de plug-in neergezet moet worden, in dit geval is dat de plugins directory van dotCMS.

Plug-ins verwijderen

Testresultaten publiceren

Undeployen Plug-ins

Wachten   
 (3 minuten)

Selenium server starten en naar URL

dotCMS afsluiten

Wachten   
(3 minuten)

dotCMS starten

Deployen Plug-ins

Ophalen TestEnabler plug-in

Ophalen plug-in

**Ophalen TestEnabler plug-in:** De TestEnabler plug-in is ontwikkeld voor het testen van plug-ins op deploy niveau. Deze plug-in zorgt ervoor dat de testen van andere plug-in die zijn gedeployed in dotCMS worden uitgevoerd. Dit gebeurt als volgt:

In de TestEnabler plug-in wordt er een methode aangeroepen die door middel van een filter wordt geactiveerd, zie figuur 16. Deze filter haalt met behulp van de dotCMS API alle plug-ins op die op dat moment zijn gedeployed in dotCMS. De namen van de plug-ins, behalve die van een standaard dotCMS plug-in en de TestEnabler zelf, worden doorgegeven aan de GenerateReportForPlugin methode in de GenerateReportService class.

|  |
| --- |
| @Override  **public** **void** doFilter(ServletRequest req, ServletResponse res, FilterChain arg2) **throws** IOException, ServletException {  GenerateReportService generateReportService = **new** GenerateReportService();  PluginAPI pluginAPI = APILocator.*getPluginAPI*();  List<String> pluginsList = pluginAPI.getDeployedPluginOrder();    **for**(String pluginName : pluginsList){  **if** (!"com.dotcms.config".equals(pluginName) && . !"TestEnabler".equals(pluginName)) {  generateReportService.GenerateReportForPlugin(pluginName);  }  }  } |

Figuur 16: De filter van de TestEnabler plug-in

De GenerateReportForPlugin maakt als eerste de “Reports” directory aan in de plug-in. Dit gebeurt door middel van de dotCMS systeemvariabele en de naam van de plug-in. Hierna worden de testclasses opgehaald door middel van de findClasses methode in de GetClassFromJarService class, zie Figuur 18. Met behulp van de Ant library worden de testen uitgevoerd. De resultaten die uit de testen zijn voortgekomen worden in een XML rapport verwerkt en in de zojuist aangemaakte map geplaatst.

|  |
| --- |
| **public** **void** GenerateReportForPlugin(String pluginName) {  **try** {  String pathToReports = System.*getenv*("DOTCMS\_HOME")+"\\plugins\\"+pluginName+"\\Reports";    Class<?>[] arrayList = GetClassFromJarService.*findClasses*(pluginName);    Project project = **new** Project();  **new** File(pathToReports).mkdir();  JUnitTask task = **new** JUnitTask();    project.setProperty("java.io.tmpdir",pathToReports);  task.setProject(project);    FormatterElement.TypeAttribute type = **new** FormatterElement.TypeAttribute();  type.setValue("xml");    FormatterElement formater = **new** FormatterElement();  formater.setType(type);  task.addFormatter(formater);    **for** (Class<?> testClass: arrayList) {  JUnitTest test = **new** JUnitTest(testClass.getName());  test.setTodir(**new** File(pathToReports));  task.addTest(test);  }    task.execute();  } **catch** (Exception e) {  Logger.*error*(GenerateReportService.**class**, "Fout bij het aanroepen van de testen! . dotCMS locatie: " + System.*getenv*("DOTCMS\_HOME") +" Foutmelding: " . + e.toString());  }  } |

Figuur 17: De methode in de TestEnabler die het de testen uitvoert en een rapport genereerd

De findClasses methode in de GetClassFromJarService haalt door middel van de private findClasses methode de testclasses op. Dit gebeurt op basis van de directory waar de testen zich in bevinden, in dit geval moet de directorynaam “/deploy/” bevatten. Echter worden hier alleen de namen van de classes opgehaald. Daarom wordt er in de convertToClasses methode de naam omgezet naar een class. Waarna de testclasses worden teruggegeven aan de GenerateReportForPlugin methode in de GenerateReportService.

|  |
| --- |
| **private** **static** String *jarFile* = "";    **public** **static** Class<?>[] findClasses(String pluginName) {  *jarFile* = System.*getenv*("DOTCMS\_HOME") + "/dotCMS/WEB-INF/lib/plugin- . "+pluginName+".jar";  List<String> classFiles = **new** ArrayList<String>();  *findClasses*(classFiles);  List<Class<?>> classes = *convertToClasses*(classFiles);  **return** classes.toArray(**new** Class[classes.size()]);  }  **private** **static** **void** findClasses(**final** List<String> classFiles) {  JarFile jf;  **try** {  jf = **new** JarFile(*jarFile*);  **for** (Enumeration<JarEntry> e = jf.entries(); e.hasMoreElements();) {  String name = e.nextElement().getName();    **if** (name.contains("/deploy/") && name.endsWith(".class") && . !name.contains("$")) {  classFiles.add(name.replaceAll("/", ".").substring(0, . name.length() - 6));  }  }  } **catch** (IOException e) {  **throw** **new** RuntimeException(e);  } **finally** {  jf.close();  }  }  **private** **static** List<Class<?>> convertToClasses(**final** List<String> classFiles) {  List<Class<?>> classes = **new** ArrayList<Class<?>>();  **for** (String name : classFiles) {  Class<?> c;  **try** {  c = Class.*forName*(name);  }  **catch** (ClassNotFoundException e) {  **throw** **new** AssertionError(e);  }  **if** (!Modifier.*isAbstract*(c.getModifiers())) {  classes.add(c);  }  }  **return** classes;  } |

Figuur 18: De class GetClassFromJarService in de TestEnabler die de testclasses ophaalt vanuit de plug-ins.

**Deployen Plug-ins:** Het deployen van de plug-ins gebeurt door middel van de invoke Ant build methode in Jenkins. De methode roept het al bestaande deploy batch bestand van dotCMS aan.

**dotCMS starten:** dotCMS starten gebeurt door middel van de buildstap “Execute Windows batch command” binnen Jenkins. Deze stap werkt hetzelfde als een batch bestand. In deze buildstap worden twee stappen uitgevoerd, zie ook figuur 19. De eerste stap is het openen van de dotCMS directory. Vanuit deze directory wordt er naar de bin folder gegaan en daar wordt vervolgens het batch bestand startup.bat aangeroepen. Dit batch bestand is een standaard dotCMS bestand waardoor dotCMS wordt opgestart.

|  |
| --- |
| cd %dotcms%  bin\startup.bat |

Figuur 19: Execute Windows batch command voor het opstarten van dotCMS

**Wachten (3 minuten):** De stap die volgt na het opstarten en afsluiten van dotCMS, gaat fout wanneer dotCMS niet volledig is gestart of afgesloten. Dit opstarten en afsluiten wordt gedaan door een extern proces, namelijk Tomcat. Deze wordt opgestart door middel van een batch bestand. Doordat Tomcat een extern proces is, wacht Jenkins niet tot het proces helemaal voltooid is. Daarom is er een sleep Task ingebouwd, zie figuur 20, die precies drie minuten wacht voordat er aan de volgende stap wordt begonnen.

|  |
| --- |
| **<target name="sleep">  <sleep seconds="180"/> </target>** |

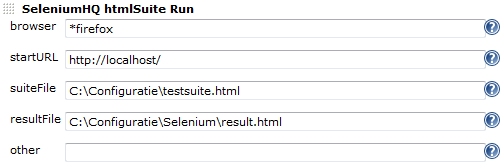
Figuur 20: De Ant sleep Task die drie minuten wacht

**Selenium server starten en naar URL:** Het activeren van de filter in de TestEnabler gebeurt door een Selenium script. Selenium is een tool waarmee er door middel van een HTML script door webpagina’s wordt gelopen. In Jenkins wordt Selenium gebruikt door middel van de Seleniumhq Plugin. Voor het activeren van de TestEnabler wordt het script van figuur 21 gebruikt. In dit script wordt alleen de pagina geopend waarmee de filter wordt geactiveerd.

|  |
| --- |
| **<tr>**  **<td>open</td>**  **<td>/deploy/test</td>**  **</tr>** |

Figuur 21: HTML script voor het openen van de filter in de TestEnabler

In Jenkins wordt het Selenium script geactiveerd door de SeleniumHQ buildstap, zie figuur 22. Hierin wordt aangegeven welke browser er gebruikt moet worden, wat de start URL is, waar het scriptbestand staat en waar de resultaten van het script moeten komen te staan.

  
 Figuur 22: SeleniumHQ buildstap in Jenkins

**dotCMS afsluiten:** dotCMS afsluiten gebeurt, net zoals bij het opstarten van dotCMS, door middel van de buildstap “Execute Windows batch command”. In deze buildstap worden twee stappen uitgevoerd, zie ook figuur 23. De eerste stap is het openen van de dotCMS directory. Vanuit deze directory wordt er naar de bin folder gegaan en daar het batch bestand shutdown.bat aangeroepen. Dit batch bestand is een standaard dotCMS bestand wat dotCMS afsluit.

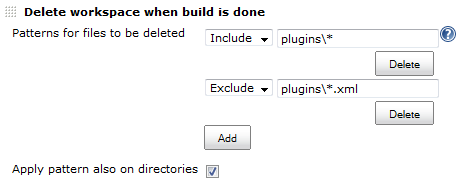
|  |
| --- |
| cd %dotcms%  bin\shutdown.bat |

Figuur 23: Execute Windows batch command voor het opstarten van dotCMS

**Undeployen plug-ins:** Het undeployen van de plug-ins gebeurt door middel van de invoke Ant build methode in Jenkins. De methode roept het al bestaande undeploy batch bestand van dotCMS aan.

**Testresultaten publiceren:** De testresultaten worden in Jenkins automatisch in een nette grafiek weergegeven door middel van de Publish JUnit test result report optie. Het enige wat er hiervoor moet worden geconfigureerd is de locatie waar het testrapport staat.

**Plug-ins verwijderen:** Het verwijderen van de plug-ins is noodzakelijk, aangezien deze in de plugins directory van dotCMS staan. Worden deze niet verwijderd dan worden bij het deployen van een andere plug-in deze plug-ins automatisch mee gedeployed. Voor het verwijderen van de plug-ins wordt er de Jenkins Workspace Cleanup Plugin gebruikt. Bij het configureren van deze plug-in kan er exact worden aangegeven welke folders en bestanden er verwijderd moeten worden, zie figuur 24.

****

Figuur 24: Het verwijderen van de plug-ins door middel van de Jenkins Workspace Cleanup Plugin

## Inrichting testomgeving voor het testen op deploy niveau (2)

De eerste versie voor het testen van plug-ins op deploy niveau werd gepresenteerd aan het dotCMS expertiseteam binnen ISAAC. Uit deze meeting met het dotCMS expertiseteam kwamen enkele voorstellen en verbeterpunten naar voren. Deze voorstellen en verbeterpunten zijn verwerkt in de nieuwe versie. Voor de verbeterde versie van de testomgeving voor het testen op deploy niveau, zie figuur 25. Op de volgende pagina staan de verwerkte voorstellen uitgelegd.   
Figuur 25 De verbeterde inrichting van de testomgeving voor het testen van de plug-in op deploy niveau **Ant script Deploy**

**Ant script Deploy**

Bestanden verwijderen

Testresultaten publiceren

Archiveren dotcms.log

Ophalen TestEnabler

Ophalen plug-in

Clean dotCMS

Undeployen plug-ins

Deploy plug-ins

Deploy dotCMS

Starten dotCMS

Controleren of dotCMS is gestart

Selenium server starten en naar URL

dotCMS afsluiten

Tomcat proces afsluiten

dotCMS niet gestart

dotCMS gestart

Net zoals bij het testen op compile niveau adviseerde Jan-Willem dat het ook handiger zou zijn om één groot Ant script te schrijven voor het testen op deploy niveau. Want in de huidige situatie waren er zes verschillende bouwmethodes binnen één job. Voor het aanmaken van een nieuwe job was dit geen probleem, het kopiëren van een soortgelijke job en het aanpassen van de configuratie zou dan al volstaan. Maar als er een bouwmethode moest worden veranderd, dan diende dit bij alle 60 jobs te gebeuren. De oplossing was dus één groot Ant script maken voor het testen op deploy niveau. Dan zou alleen in dit Ant script iets moeten worden aangepast, aangezien alle jobs hetzelfde Ant script gebruiken. In dit Ant script worden alle stappen verwerkt die in figuur 25 staan weergegeven in het vak “Ant script Deploy”. Het zou ook mogelijk zijn om de scripts van het testen op compile en deploy niveau samen te voegen, echter is er vooraf gekozen om deze stappen apart te houden. Deze stappen worden apart gehouden omdat het mogelijk is dat er in de toekomst bij het testen op deploy niveau een virtuele omgeving wordt opgezet voor het testen. Het testen op deploy niveau met een virtuele omgeving kost dan vermoedelijk veel tijd. Terwijl de bedoeling van het testen op compile niveau is dat het resultaat binnen drie minuten op het scherm te zien is.

Eén van de problemen waar tegenaan werd gelopen is dat het niet mogelijk was om systeemvariabelen te gebruiken. Dit is normaal gesproken wel mogelijk in Ant, maar omdat Jenkins het Ant script start, is Jenkins de “environment”, en worden er alleen de environment variabelen gebruikt die in Jenkins actief zijn. Hierdoor staan er nu bovenaan het script extra properties waarin de zaken zijn gedefinieerd die benodigd zijn, zoals de locatie van dotCMS. Een ander probleem was dat er twee scripts werden gebruikt, één daarvan werd vanuit de hoofdfolder gestart, de andere vanuit de plug-in folder. Hierdoor kunnen de twee scripts niet tegelijkertijd uitgevoerd worden. Dit is opgelost door een macrodef te gebruiken. Deze macrodef houdt in dat er een Ant Task wordt aangeroepen in een ander script, zie figuur 26.

|  |
| --- |
| **<macrodef name="iterate">**  **<attribute name="target"/>**  **<sequential><subant target="@{target}">**  **<fileset dir="plugins" includes="\*/build.xml"/>**  **</subant>sequential>**  **</macrodef>** |

Figuur 26: Ant script voor het aanroepen van een Ant Task in een andere bestand

Voor het starten en afsluiten van dotCMS wordt de functie executable gebruikt, zie figuur 27. Deze functie activeert een ander programma, in dit geval de Windows command line. Aan deze functie kunnen argumenten worden meegegeven. In dit geval wordt er aangegeven dat er in de C:\ directory gestart moet worden. Als tweede argument wordt meegegeven welk batch bestand er uitgevoerd moet worden.

|  |
| --- |
| **<target name="startup">**  **<exec executable="cmd">**  **<arg value="/c"/>**  **<arg value="${dotcms.home}\bin\startup.bat"/>**  **</exec>**  **</target>** |

Figuur 27: Ant script voor het activeren van een ander programma door middel van de functie exec

Om te controleren of dotCMS al gestart is, was een vrij eenvoudig Ant script te vinden, zie figuur 28. In dit script wordt aangegeven hoelang er maximaal moet worden gecontroleerd of de dotCMS server gestart is, in dit geval is dat 3 minuten. Bij checkevery wordt aangegeven hoe vaak er gecontroleerd moet worden of dotCMS is gestart, in dit geval is dat elke 50 milliseconden. Als laatste wordt er aangegeven welke URL er gecontroleerd moet worden, wat in dit geval de inlogpagina van dotCMS is. Mocht de link niet beschikbaar zijn, dan wordt de timeoutproperty gezet. In de target check-if-startup-failed, zie figuur 29, wordt gecontroleerd of deze timeoutproperty is gezet. Is dit het geval dan wordt Tomcat direct afgesloten en de plug-ins geundeployed.

|  |
| --- |
| **<target name="check-if-dotcms-is-started">**  **<waitfor maxwait="3" maxwaitunit="minute" checkevery="50" . timeoutproperty="server.timeout">**  **<http url="http://localhost/admin"/>**  **</waitfor>**  **</target>** |

Figuur 28: Ant script voor het controleren of dotCMS gestart is

|  |
| --- |
| **<target name="check-if-startup-failed" if="server.timeout">**  **<echo message="dotCMS niet opgestart! tomcat.stop uitvoeren"/>**  **<antcall target="tomcat.stop" />**  **<echo message="Undeploy-plugins uitvoeren"/>**  **<antcall target="undeploy-plugins"/>**  **<fail message="dotCMS niet opgestart, zie de log voor informatie!"/>**  **</target>** |

Figuur 29: Ant script voor het controleren of de server.timeout property is gezet

Het starten van het Selenium script gebeurde voorheen door middel van de buildstap die binnen Jenkins beschikbaar is. Dit is nu echter niet meer mogelijk, doordat de daaropvolgende stappen verwerkt zijn in het Ant script. Hierdoor moest er een andere manier worden gevonden om het Selenium script te activeren. Hiervoor was geen speciale Ant Task, waardoor er net zoals bij het opstarten en afsluiten van dotCMS, gebruik moest worden gemaakt van de functie executable, zie figuur 30. Aan deze functie wordt meegegeven waar de Selenium server staat, welke browser er moet worden gebruikt, wat de basis URL is, welk script er afgespeeld moet worden en waar de resultaten in moeten worden verwerkt.

|  |
| --- |
| **<target name="selenium">**  **<exec executable="cmd">**  **<arg value="/c"/>**  **<arg value="java**  **-jar C:\Configuratie\Selenium\selenium-server-standalone-2.21.0.jar . -htmlSuite \*firefox http://localhost/ . C:\Configuratie\testsuite.html . C:\Configuratie\Selenium\result.html"/>**  **</exec>**  **</target>** |

Figuur 30: Selenium server starten door middel van een Ant script

Het controleren of dotCMS al correct was afgesloten viel echter niet mee. Dit was nodig omdat anders de plug-in niet kan undeployen maar ook voor het archiveren van de dotcms.log in de job. Het probleem was dat dotCMS niet aangeeft wanneer Tomcat is afgesloten en Tomcat houdt het log bestand nog 10 minuten vast nadat dotCMS al is afgesloten. Hierdoor is het niet mogelijk om het dotcms.log bestand te verwijderen aan het einde van de job. Dit moet gebeuren zodat er bij het begin van de volgende job een leeg dotcms.log bestand kan worden gebruikt. Uiteindelijk werd het duidelijk dat er twee manieren waren om dit probleem op te lossen, namelijk:

* Een bepaalde tijd wachten totdat het zeker is dat Tomcat is afgesloten
* Het Tomcat proces killen

De eerste variant was geen goede optie, omdat deze afhankelijk is van de snelheid van de server en omdat het daarna nog steeds niet zeker is of het proces is afgesloten. Dus werd er voor de tweede optie gekozen, namelijk het killen van het Tomcat proces. Na flink wat zoekwerk werd er een soortgelijk Ant script gevonden, zie figuur 31. Na enkele aanpassingen voldeed het script aan de eisen. In dit script worden met behulp van het jps Java proces, de draaiende Java processen gevonden. In dit proces wordt gezocht naar het woord Bootstrap, zoals Tomcat als Java proces heet, en alleen deze regel wordt bewaard. Op deze regel staat het PID, wat staat voor Process ID, en de naam van het proces. Hiervan wordt de naam Bootstrap verwijderd en blijft alleen het PID over. Door middel van de taskkill functie in command line wordt het proces gekilled.

|  |
| --- |
| **<target name="tomcat.stop">**  **<exec executable="C:/Program Files/Java/jdk1.6.0\_31\bin\jps" output="pid.out.file"/>**  **<loadfile srcfile="pid.out.file" property="pid.out">**  **<filterchain>**  **<linecontains>**  **<contains value="Bootstrap"/>**  **</linecontains>**  **<tokenfilter>**  **<deletecharacters chars="Bootstrap"/>**  **<trim/>**  **<ignoreblank/>**  **</tokenfilter>**  **<striplinebreaks/>**  **</filterchain>**  **</loadfile>**  **<echo>Killing Tomcat instance with PID - "${pid.out}"</echo>**  **<exec executable="cmd">**  **<arg value="/c"/>**  **<arg value="taskkill /F /PID"/>**  **<arg value="${pid.out}"/>**  **</exec>**  **<delete file="pid.out.file"/>**  **</target>** |

Figuur 31: Ant script voor het killen van het Tomcat proces

**Archiveren dotcms.log**

Mocht het zo zijn dat het deployen of opstarten van dotCMS foutief verloopt dan is daar meer informatie over te vinden in de dotcms.log. Daarvoor is het handig om het dotcms.log bestand op te slaan in de job als archiefbestand. Dit was na het toevoegen van de tomcat.stop Task in het Ant script, simpel in te stellen in Jenkins na de buildstappen. Het enige wat ingevuld moest worden, is welk bestand er gearchiveerd moest worden.

**Bestanden verwijderen**

Aan deze stap is in de tweede versie toegevoegd dat nu ook de dotcms.log verwijderd moet worden. Dit was eerst echter niet mogelijk omdat Tomcat het bestand vast hield nadat dotCMS al was afgesloten. Maar door de toevoeging van de tomcat.stop Task in het Ant script, waar het proces wordt gekilled, is het nu wel mogelijk om het bestand te verwijderen.

|  |  |
| --- | --- |
| Deze stap is toegevoegd zodat er bij de volgende job met een lege dotcms.log begonnen wordt, zodat er in het gearchiveerde dotcms.log bestand alleen de logregels staan van de zojuist gedraaide job.  In Jenkins werd dit ingesteld door een extra pattern te include met daarin de locatie van de dotcms.log, zoals hier rechts beschreven in figuur 32. | Figuur 32: Inrichting voor het deleten van bestanden |

## Unittesten

Voor het testen van stukken code diende er ook unittesten te worden geschreven. De unittesten werden geschreven met behulp van de JUnit library. Allereerst werden er alleen simpele unittesten geschreven zodat er met het inrichten van de testomgeving kon worden gecontroleerd of de jobs juist waren geconfigureerd.

Nadat de job op zowel compile als deploy niveau volledig was ingericht werd er begonnen met het schrijven van testen voor de CustomLogin plug-in. De CustomLogin plug-in werd uitgekozen omdat deze in een eerder stadium al is aangepast, waardoor er voldoende kennis over de plug-in is voor het schrijven van testen. Ook is er gedocumenteerd hoe unittesten voor plug-ins kunnen worden geschreven. Deze documentatie is voor de medewerkers van ISAAC die de testen gaan schrijven.

Deze fase verliep voorspoedig, omdat er tijdens de opleiding al ervaring was opgedaan met het schrijven van JUnit testen.

## Jenkins op een server draaien

Nadat de stappen lokaal gerealiseerd waren, was de volgende stap om Jenkins op een server te laten draaien. Hierdoor zou het mogelijk worden om de testen ook in de nacht te laten draaien, wat zou laten zien of de configuratie ook op een langere termijn werkt. Deze stap leverde geen enkel probleem op omdat de server werd ingericht door Deniz, de Systeem Administrator. Hierdoor verliep het installeren en configureren van Jenkins op detzelfde manier als op een lokale PC. Tijdens deze procedure is ook de documentatie geschreven voor ISAAC over hoe de Jenkins server en de compile- en deployjob ingericht moeten worden.

# Conclusie en aanbevelingen

Uit het onderzoek blijkt dat het wel degelijk interessant kan zijn om een testomgeving op te zetten voor plug-ins. Sterker nog: het hoeft helemaal niet zoveel geld te kosten als in eerste instantie in deze scriptie werd aangegeven. Zo is de testomgeving Jenkins helemaal gratis te verkrijgen en kost het alleen de werkuren die daar in moeten worden gestoken.

Er dient echter wel rekening te worden gehouden met het aantal plug-ins wat er getest moet worden. Zoals bij de deelvraag “Wanneer een automatische testomgeving gebruiken?” wordt aangegeven moet de omvang van het software project groot genoeg zijn, beschikbaar voor meerdere versies zijn of het moet vaak handmatig worden getest om te controleren of het nog wel naar behoren werkt. Mocht dat het geval zijn dan is het interessant om een testomgeving op te gaan zetten. Voor ISAAC is dit zeker het geval, aangezien ze meer dan 60 plug-ins in de omloop hebben, het content management systeem vaak verandert van versie en daardoor ook van API en veel plug-ins voor meer dan één project worden gebruikt.

Zo kwam er uit het gemaakte concept van de testomgeving naar voren, dat in de nieuwe versie van dotCMS 2.0, 26 van de 59 plug-ins niet correct werken. Waarbij de test voor elke plug-in minder dan drie minuten duurde en voor de plug-ins die niet correct werkte zelfs maar enkele seconden. Dit bespaard veel tijd ten opzichte van het handmatig testen door een werknemer.

Al met al is de conclusie dat het voor ISAAC zeker rendabel is om het concept van de testomgeving verder uit te werken zodat de testomgeving volledig benut kan worden voor het testen van plug-ins.

# Evaluatie

## Zelfreflectie

Na kritisch gekeken te hebben naar mijn functioneren tijdens de afgelopen afstudeerperiode zijn er positieve, maar ook negatieve punten te noemen.

Het bijhouden en volgen van de planning was een moeilijk punt. Doordat deze opdracht midden in het bedrijfsleven werd uitgevoerd, veranderde de precieze opdracht meerdere malen. Ook werd er direct advies uitgebracht van collega’s bij ISAAC over mogelijke kleine aanpassingen. Dit was erg leerzaam, maar daardoor is het me niet altijd gelukt om de planning precies aan te houden.

Tijdens deze afstudeerperiode heeft men vooral mij de keuzes laten maken qua onderzoeken en het oplossen van problemen. Waar nodig werd ik bijgestaan door collega’s, want zoals er hier werd gezegd: “er zijn geen problemen, maar uitdagingen”. Doordat ik eerst zelf naar oplossingen zocht, heb ik mezelf verder kunnen ontwikkelen. Als er een probleem was waar ik zelf niet uitkwam, werd ik stap voor stap intensief begeleid. Hierdoor zou ik nu deze problemen wel zelf kunnen oplossen.

De stage was voor mij een positieve ervaring en wordt vervolgd met een baan bij ISAAC.

## Doelen

Het belangrijkste doel, uitgezonderd van het project, was het opdoen van zoveel mogelijk nieuwe kennis en ervaring. Met alle nieuw geleerde technieken en opgedane ervaringen tijdens deze afstudeerperiode is dit dan ook volledig geslaagd. Het doel voor het project was om te onderzoeken hoe een automatische testomgeving het beste kon worden opgezet en hiervoor een conceptversie te maken.

Dit doel is niet volledig behaald. Dit kwam onder andere doordat er extra werkzaamheden voor ISAAC zijn uitgevoerd, buiten de stage opdracht om. Deze extra werkzaamheden waren erg leerzaam en hebben veel nieuwe ervaringen opgeleverd.

Van de afgesproken functionaliteit, zijn er nog enkele zaken die nog geïmplementeerd moeten worden. Wel is de huidige functionaliteit beter te onderhouden door de medewerkers van ISAAC dan vooraf was afgesproken.

Het invoeren van een virtuele dotCMS in de testomgeving is niet gebeurd. Dit is niet uitgevoerd omdat hier geen tijd meer voor was. Dit kwam doordat de prioriteit lag op het verbeteren van de functionaliteiten van de automatische testomgeving. Deze verbeteringen vloeide voort uit de presentaties die zijn gegeven voor het dotCMS team van ISAAC en voor Jan-Willem.

Ook al is de oorspronkelijke doelstelling niet volledig gehaald, het algehele project is zeker een succes dankzij alle opgeleverde kennis en de reeds ontwikkelde basis voor een verder te ontwikkelen automatische testomgeving voor het testen van plug-ins.

# Literatuurlijst

**Internetbronnen**

CI Feature Matrix (confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix)

Chris Read, Continuous Integraton Server Comparison 2008 (http://blog.chris-read.net/2008/09/22/continuous-integration-server-comparison-2008/)

Cuy Clapperton, What are plug-ins? (www.bbc.co.uk/webwise/guides/about-plugins)

Jiantao Pan, Software Testing (www.ece.cmu.edu/~koopman/des\_s99/sw\_testing)

John Ferguson, Which open source CI tool is best suited for your application's environment? (http://www.javaworld.com/javaworld/jw-11-2006/jw-1101-ci.html?page=1)

Klaus Svarre, Content management system (CMS) (searchsoa.techtarget.com/definition/content-management-system)

Martin Fowler, Continuous Integration (http://www.martinfowler.com/articles/continuousIntegration.html)

Roy Osherove, The difference between continuous Integration tools and automated build tools (http://osherove.com/blog/2010/1/14/the-difference-between-continuous-integration-tools-and-auto.html)

Vlad Kofman, The Best Continuous Integration Tools (www.developer.com/open/article.php/3803646/The-Best-Continuous-Integration-Tools.htm)

What is content management (www.contentmanager.eu.com/cms.htm)

**Continuous Integration Servers**

Bamboo (www.atlassian.com/software/bamboo/overview)

CruiseControl (cruisecontrol.sourceforge.net/)

Jenkins (jenkins-ci.org/)

Teamcity (www.jetbrains.com/teamcity/index.html)

Bijlagen

1. Project Initiation Document

|  |  |  |
| --- | --- | --- |
| **PROJECT: aFSTUDEERSTAGE**  **Testtool voor Plug-ins in dotCMS**  logo-ISAAC( Project Initiation Document ) | | |
|  |  |  |
|  |  |  |

**Documenthistorie**

**Revisies**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versie** | **Status** | **Datum** | **Wijzigingen** |
| 0.1 | concept | 17-02-2012 | PID document volledig ingevuld |
| 0.2 | concept | 27-02-2012 | n.a.v. Xander’s commentaar enkele zaken aangepast |
| 1.0 | final | 05-03-2012 | n.a.v. Coen’s commentaar enkele zaken aangepast |
|  |  |  |  |

**Goedkeuring**

Dit document behoeft de volgende goedkeuringen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versie** | **Datum goedkeuring** | **Naam** | **Functie** | **Paraaf** |
| 1 |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Distributie**

Dit document is verstuurd aan:

|  |  |  |  |
| --- | --- | --- | --- |
| **Versie** | **Datum verzending** | **Naam** | **Functie** |
| 0.1 | 17-02-2012 | Xander Steinmann | Stagebegeleider |
| 0.2 | 02-03-2012 | Coen Crombach | Docentbegeleider |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Managementsamenvatting**

**Doel van dit document**

Dit document heeft tot doel het project te definiëren, als basis te dienen voor het management ervan en de beoordeling van het succes van het project mogelijk te maken.

De twee belangrijkste redenen voor gebruik van dit document zijn:

* om er zeker van te zijn dat het project goed gedefinieerd is en er duidelijkheid bestaat over de invulling van het project, voordat ISAAC en Fontys Hogescholen goedkeuring geven voor dit project.
* om te dienen als basisdocument op grond waarvan het management en de stagebegeleider de voortgang en wijzigingen kunnen toetsen en bewaken.

**Aanleiding**

ISAAC gebruikt een CMS (Content Management Systeem) genaamd dotCMS, waarvoor ze talloze plug-ins hebben ontwikkeld. Nu blijkt soms dat na een update van dotCMS er enkele plug-ins niet meer goed functioneren. Ze willen daarom een geautomatiseerde testomgeving die individueel plug-ins kan testen, zowel op compilatie niveau (checken of de API’s van dotCMS zijn veranderd) als op functieniveau (bepaalde functies aanroepen en controleren of het resultaat overeenkomt met de verwachtingen). De omgeving moet zo goed mogelijk geautomatiseerd worden, zodat de medewerkers na zo min mogelijk handelingen te zien krijgen of alles nog correct werkt.

**Globale aanpak**

De eerste stap zal zijn het bekend worden met het content management systeem. Deze stap bestaat uit het doorlopen van het content management systeem en het ontwikkelen van een plug-in voor het desbetreffende systeem. Nadat deze stappen met succes zijn doorstaan zal de volgende stap bestaan uit het onderzoek. In het onderzoek wordt er uitgezocht hoe je een plug-in het beste kunt testen en waarmee je dit zou moeten doen. De uitkomst van het onderzoek zal bepalen hoe de testomgeving eruit komt te zien en met welk soort testen de plug-ins zullen worden getest.

**Globale kosten en doorlooptijd**

Het project duurt 85 dagen/17 weken, van 13-02-2012 tot ± 15-06-2012, waarin de stagiair full time aan het project gaat werken. In totaal is er dus 680 uur beschikbaar.

**Inhoudsopgave**

[**1** **Inleiding 5**](#_Toc317254930)

[Opbouw van dit document 5](#_Toc317254931)

[**2** **Achtergrond 6**](#_Toc317254932)

[**3** **Projectdefinitie 7**](#_Toc317254933)

[3.1 Projectdoelstellingen 7](#_Toc317254934)

[3.2 Gekozen oplossing of aanpak 7](#_Toc317254935)

[3.3 Scope van het project 7](#_Toc317254936)

[3.4 Producten c.q. eindresultaat 8](#_Toc317254937)

[3.5 Beperkingen 8](#_Toc317254938)

[3.6 Afhankelijkheden 8](#_Toc317254939)

[3.7 Aannames 8](#_Toc317254940)

[**4** **Projectorganisatiestructuur 9**](#_Toc317254941)

[4.1 Stagiair 9](#_Toc317254942)

[*Rolbeschrijving 9*](#_Toc317254943)

[*Specifieke verantwoordelijkheden 9*](#_Toc317254944)

[4.2 Stagebegeleider 9](#_Toc317254945)

[*Rolbeschrijving 9*](#_Toc317254946)

[4.3 Docentbegeleider 10](#_Toc317254947)

[*Rolbeschrijving 10*](#_Toc317254948)

[4.4 Projectsupport 10](#_Toc317254949)

[*Rolbeschrijving 10*](#_Toc317254950)

[**5** **Projectbeheersing 11**](#_Toc317254951)

[5.1 Rapportage 11](#_Toc317254952)

[5.2 Voortgangsbewaking 11](#_Toc317254953)

[5.3 Kwaliteit 11](#_Toc317254954)

[**Bijlage A: ·Communicatieplan 12**](#_Toc317254955)

[Revisies 12](#_Toc317254956)

[**Bijlage B: Planning 14**](#_Toc317254957)

**Inleiding**

**Doel van dit document**

Dit document is opgesteld om alle relevante basisinformatie en uitgangspunten van het project vast te leggen om het op de juiste wijze te kunnen besturen. Het heeft tot doel het project te definiëren, als basis te dienen voor het management ervan en de beoordeling van het succes van het project mogelijk te maken.

Dit Project Initiatie Document (of PID) behandelt de volgende fundamentele aspecten van het project:

Wat beoogt men met het project te bereiken?

Waarom is het belangrijk om deze doelstellingen te bereiken?

Wie zijn er betrokken bij het managen van het project en wat zijn hun rollen en verantwoordelijkheden?

Hoe en wanneer zullen de maatregelen die in dit PID besproken worden gerealiseerd worden?

Het document wordt gebruikt:

om er zeker van te zijn dat het project een gezonde basis heeft voordat de Stuurgroep gevraagd wordt zich aan het project te committeren;

om te dienen als basisdocument op grond waarvan de Stuurgroep en de Projectmanager de voortgang en wijzigingen kunnen toetsen en bewaken en vragen omtrent geldigheid van het project tijdens de uitvoering ervan kunnen beoordelen.

**Opbouw van dit document**

Om aan te geven welke onderdelen worden bijgewerkt en dus nieuwe versies zullen krijgen tijdens de voortgang van het project is dit Project initiatie document verdeeld in twee secties: een statisch gedeelte en een dynamisch gedeelte:

Het "statische" deel bestaat uit de hoofdstukken en bijlagen:

Achtergrond (Hoofdstuk 2)

Projectdefinitie (Hoofdstuk 3)

Projectorganisatiestructuur (Hoofdstuk 4)

Projectbeheersing (Hoofdstuk 5)

Communicatieplan (Bijlage A)

Projectplanning (Bijlage B)

**Achtergrond**

ISAAC gebruikt een CMS (Contant Management Systeem) genaamd dotCMS, waarvoor ze talloze plug-ins hebben ontwikkeld. Nu blijkt soms dat na een update van dotCMS er enkele plug-ins niet meer goed functioneren. Ze willen daarom een geautomatiseerde testomgeving die individueel plug-ins kan testen, zowel op compilatie niveau (checken of de API’s van dotCMS zijn veranderd) als op functieniveau (bepaalde functies aanroepen en controleren of het resultaat overeenkomt met de verwachtingen). De omgeving moet zo goed mogelijk geautomatiseerd worden, zodat de medewerkers na zo min mogelijk handelingen te zien krijgen of alles nog correct werkt. Dit pakken we projectmatig aan omdat de stagiair nog geen enkele ervaring heeft met dotCMS. Het is daarom belangrijk dat het projectmatig wordt aangepakt, zodat alles in stappen kan worden gedaan.

**Projectdefinitie**

**Project****doelstellingen**

De testtool voor plug-ins in dotCMS moet worden gerealiseerd omdat nu nog alles handmatig moet worden getest wanneer er een nieuwe versie van dotCMS uitkomt. Normaal gesproken wordt er bij een release tijdig aangegeven mocht er iets veranderen in de API’s. Dit is niet het geval bij dotCMS, waardoor er sommige plug-ins niet meer werken omdat de API’s die ze gebruiken ineens zijn gewijzigd. De ideale situatie zou zijn wanneer er een testomgeving zou komen waarin de plug-ins automatisch worden getest. Afhankelijk van de kwaliteiten van het stagiair zal er worden bepaald in hoeverre de opdracht moet voldoen aan de voorwaarde, hierover in *3.2 Gekozen oplossing of aanpak* meer. De voordelen die uit dit project zullen voortvloeien zijn dat de medewerkers van ISAAC bij een nieuwe release van dotCMS niet meer alle plug-ins handmatig moeten testen.

**Gekozen oplossing of** **aanpak**

Aangezien de stagiair nog geen enkele ervaring heeft met dotCMS is het eerst belangrijk dat hij eerst leert hoe dotCMS werkt, dit gebeurt naar aanleiding van het boek: “dotCMS From the Ground Up” van Michael Fienen. Wanneer er voldoende kennis over dotCMS in het algemeen is vergaard, zal er aan de hand van het uitbreiden van een bestaande plug-in extra kennis worden vergaard. Dit houdt in dat er kennis is over hoe de plug-in in elkaar zit, hoe deze uit te breiden is en hoe je deze deployed in dotCMS. Dit zullen we in stap drie uitbreiden met het automatisch deployen en compileren van een plug-in door middel van build scripts. Hierna zal er verder worden gegaan met het testen in JUnit, dit zal vooral op compilatie niveau zijn. Waarna de functies worden getest door middel van een scripttaal zoals Groovy. Mocht dit allemaal voltooid zijn dan wordt dit aan de testomgeving Hudson gekoppeld. Uiteindelijk wordt de plug-in automatisch in een schone virtuele omgeving van VMWare gedeployed.

Hieronder staan de stappen van de aanpak nogmaals één voor één opgesomd:

1. Kennis vergaren over dotCMS
2. Het leren van dotCMS: Aan de hand van het uitbreiden van een bestaande plug-in wordt extra kennis vergaard over hoe dotCMS werkt
3. Automatisch deployen van een plug-in: m.b.v. build scripts zal een plug-in automatisch worden gedeployed en gecompileerd
4. JUnit testing: testen m.b.v. JUnit, dit zal in eerste instantie vooral op compilatie niveau zijn
5. Functies testen m.b.v. een scripttaal zoals Groovy en integreren in dotCMS/JUnit
6. Koppelen aan een testomgeving (Hudson)
7. Automatisch deployen van een versie van een plug-in in een VMWare omgeving: Het deployen gebeurt steeds in een schone virtuele omgeving

Afhankelijk van de kwaliteiten van de stagiair zal er worden bepaald welke onderdelen er wel/niet worden uitgevoerd tijdens deze stage.

**Scope van het project**

Het project wordt uitgevoerd op de developers afdeling van ISAAC door de stagiair. De personen die binnen ISAAC met dotCMS werken zullen worden betrokken bij het project. Gezien de kennis die ze al bezitten over dotCMS en de plug-ins die ze zelf hebben ontwikkeld. Advies en feedback binnen het project zal vooral worden verzorgd door de stagebegeleider, Xander Steinnman.

**Producten c.q. eindresultaat**

*Welke producten moeten worden opgeleverd?*

Het ideale eindresultaat betreft een automatische testtool voor plug-ins in dotCMS. Dit moet ervoor zorgen dat de medewerkers van ISAAC bij een update van dotCMS niet alle plug-ins handmatige moeten gaan controleren en testen.

Om de medewerkers van ISAAC gebruik te laten maken van het systeem zal er een handleiding worden gemaakt waarin staat hoe je het systeem precies moet gebruiken voor het testen van de plug-ins.

Het volgende behoort tot het eindresultaat:

* Het systeem
* Goede documentatie in het systeem
* Handleiding van het systeem
* Onderzoek document over het testen van plug-ins in een content management systeem
* Overdracht / Presentatie

**Beperkingen**

Het beheren van het systeem behoort niet tot het eindresultaat, de stagiair is hiervoor niet verantwoordelijk. Het systeem wordt overgedragen aan een nog aan te wijzen persoon binnen ISAAC, de overdracht behoort wel tot het eindresultaat.

Het volgende behoort niet tot het eindresultaat:

* Beheer van systeem na de realisatie

**Afhankelijkheden**

Het project is afhankelijk van de koppeling tussen de verschillende systemen. Zo moet het mogelijk zijn om dotCMS in Eclipse of een andere Java IDE omgeving te draaien om de plug-ins door middel van Unittesten te testen. Verder moet er in het ideale scenario ook mogelijk zijn om automatisch een virtuele omgeving op te zetten met een clean-install van dotCMS met daarbij de behorende plug-ins.

Het project is afhankelijk van de software die door ISAAC wordt aangeboden. Als er benodigde software niet of niet op tijd wordt geleverd aan de stagiair, kan het project vertraging oplopen of kunnen bepaalde functionaliteiten zelfs niet gerealiseerd worden.

Benodigde software:

* dotCMS (week 1)
* Eclipse (week 3)
* VMWare (week 10)

**Aannames**

Na de afronding van de stage wordt het product beheerd door ISAAC. Het beheer zal vooral bestaan uit het verbeteren en toevoegen van functionaliteiten indien dit gevraagd wordt voor de nieuwe plug-ins.

**Projecto****rganisatiestructuur**

Xander Steinmann

Stagebegeleider

Coen Crombach

Docentbegeleider

Danny Gloudemans

Stagiair

ISAAC Medewerker

Projectsupport

**Stagiair**

***Rolbeschrijving***

De stagiair is verantwoordelijk voor de voortgang van het project en het eindresultaat. Hij heeft de volgende taken en verantwoordelijkheden:

* Het opstellen van het Project Initiation Document, en daarmee doelstellingen, producten en eindresultaat definiëren;
* Het realiseren van het project, waarmee de opgestelde doelstellingen, producten en eindresultaat worden behaald;
* Houdt de stagebegeleider en docentbegeleider voldoende op de hoogte van de voortgang van het project;

***Specifieke verantwoordelijkheden***

Bepalen de te leveren functionaliteit, architectuur van de oplossing en fasering van de ontwikkelingen.

**Stagebegeleider**

***Rolbeschrijving***

De stagebegeleider is verantwoordelijk voor het begeleiden van de stagiair tijdens het project. Hij heeft de volgende taken en verantwoordelijkheden:

* ontvangt de afstudeerder vóór aanvang van de afstudeerperiode en maakt afspraken over de inhoud, uitvoering, en eventuele beloning van de afstudeerwerkzaamheden in de vorm van een ondertekende afstudeerovereenkomst;
* ontvangt de afstudeerder bij aanvang van het afstuderen en introduceert hem/haar in het bedrijf;
* zorgt dat de werkzaamheden uitgevoerd worden conform de doelstellingen van het afstuderen;
* is inhoudelijk deskundig en begeleidt de afstudeerder om een zo goed mogelijk resultaat te behalen; hij/zij begeleidt de afstudeerder dus ook bij het opstellen van het PID;
* is aanwezig bij de bedrijfsbezoeken en bespreekt met de docentbegeleider en de afstudeerder het sociaal en vakinhoudelijk functioneren;
* overlegt met de afstudeerder en/of de docentbegeleider indien dat nodig is;
* tekent de afstudeerscriptie voor gezien;
* vult de bedrijfsbeoordeling in en ondertekent het ingevulde dagenverantwoord formulier, voert aan het einde van het afstuderen een beoordelingsgesprek met de afstudeerder en koppelt de bedrijfsbeoordeling terug met de docentbegeleider.

**Docentbegeleider**

***Rolbeschrijving***

De docentbegeleider is verantwoordelijk voor de procesmatige begeleiding van de stage. Hij heeft de

volgende taken en verantwoordelijkheden:

* Geeft advies als je afstudeeropdracht in de eerste weken nog wijzigt over de afstemming tussen je POP en je afstudeeropdracht;
* Adviseert bij het volgen van het Tienstappenplan;
* Maakt na ontvangst van het PID een afspraak voor een eerste gesprek bij het bedrijf met jou als afstudeerder en je bedrijfsbegeleider;
* Bespreekt en geeft feedback op het PID;
* Bezoekt tijdens de afstudeerperiode het afstudeerbedrijf twee keer en bespreekt de voortgang tot dan toe aan de hand van de opgeleverde producten;
* Houdt het verloop van het afstuderen in de gaten aan de hand van logboeken/blogs en bekijkt of de student toekomt aan het realiseren van zijn competenties;
* Begeleidt de afstudeerder bij het schrijven van zijn afstudeerscriptie en beroepsproducten en geeft daar feedback op;
* Geeft feedback aan de afstudeercoördinator over het bedrijf, de opdracht, de begeleiding in de vorm van een evaluatie.

**Projectsupport**

***Rolbeschrijving***

Projectsupport voorziet de stagiair van de reeds verworven kennis en informatie, en geven raad en oplossingen bij voorkomende problemen tijdens het project.

**Projectbeheersing**

**Rapportage**

|  |  |  |  |
| --- | --- | --- | --- |
| **Partij:**  **Rapport:** | **Stagiair** | **Stagebegeleider** | **Docentbegeleider** |
| **PID** | O | A+G | A+G |
| **Logboek** | O |  | I |
| **Afstudeerscriptie** | O | A+G | A+T |
| **Handleiding/Documentatie van het product** | O | G |  |
| **Dagenverantwoording** | O | G | I |

**Legenda:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **O** | Opstellen | **A** | Adviseren | **I** | Ontvangen ter informatie |
| **T** | Toetsen | **D** | Distribueren/archiveren | **G** | Goedkeuren |

**Voortgangsbewaking**

De voortgang van het project ligt volledig in handen van het de stagiair. Toch zal er af en toe overleg zijn tussen het projectsupport en/of de stagebegeleider om advies en kennis te werven.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Overleg** | **Aanwezig** | **Frequen­tie** | **Tijdstip** | **Doel** | **Onder­werpen** | **Notulen** |
| Informeel advies | Projectsupport, Stagiair | Onbepaald |  | Advies en kennis verwerven | Inhoudelijke aspecten | Notities |
| Informeel advies | Stagiair, Stagebegeleider | Onbepaald |  | Advies en kennis verwerven | Inhoudelijke aspecten | Notities |

**Kwaliteit**

De kwaliteit van het project wordt bewaakt door het testen van de producten. Tevens wordt er overleg gepleegd en feedback gevraagd aan de betrokken personen.

**Bijlage A: ·Communicatieplan**

**Revisies**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versie** | **Status** | **Datum** | **Wijzigingen** |
| 0.1 | concept | 17-02-2012 | Communicatieplan ingevuld |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Inleiding**

Dit communicatieplan benoemt alle partijen die een (positief of negatief) belang hebben bij het project en de wijze waarop zij bij het project zullen worden betrokken en welke communicatievormen daarbij gebruikt worden. Het gaat hierbij om partijen en communicatie buiten de formele projectmanagementstructuur zoals beschreven in het PID.

**Belanghebbenden bij het project**

|  |  |  |  |
| --- | --- | --- | --- |
| **Wie** | **Namens** | **Belang** | **Communicatievorm(\*)** |
| *persoon* | *groep, afdeling* | *Welk belang bij project?* | *Hoe wordt deze persoon betrokken bij het project?* |
| Stagebegeleider | ISAAC | Tijdig ontvangen van een kwalitatief goed product | adviseren, accepteren, overleggen. |
| Projectsupport | ISAAC | Het leveren van informatie en kennis aan de stagiair | Aansturen, adviseren, overleggen. |
| Stagiair | Fontys | Ervaring opdoen, zelfontwikkeling binnen een ICT bedrijf. | Uitvoeren, besluiten, overleggen. |
| Docentbegeleider | Fontys | Stageadvies | Adviseren |

*(\*) bijvoorbeeld door: overleggen, adviseren, informeren, besluiten, accepteren, uitvoeren, aansturing*

**Communicatiekanalen**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Van** | **Naar** | **Informatie** | **Medium** | **Frequentie** |
| *Persoon of groep* | *Persoon of groep* | *Soort informatie* | *Email, telefoon, memo, rapport* |  |
| Stagiair | Stagebegeleider | PID | E-mail | Eenmalig |
| Stagiair | Docentbegeleider | PID | Email | Eenmalig |
| Stagebegeleider | Stagiair | Feedback | Email, mondeling | onbepaald |
| Stagiair | Docentbegeleider | voortgangsgesprek | Email | Wekelijks |
| Docentbegeleider | Stagiair | Feedback | Email, mondeling | Onbepaald |
| Stagiair | Projectsupport | Raad/advies vragen | Email, mondeling | Onbepaald |
| Projectsupport | Stagiair | Raad/advies | Email, mondeling | Onbepaald |
| Stagiair | Stagebegeleider | Inplannen Terugkomdag | Email | Eenmalig |
| Stagiair | Docentbegeleider, Stagebegeleider | Afspraak inleveren Scriptie | Email | Eenmalig |
| Stagiair | Docentbegeleider, Stagebegeleider | Scriptie | Email, schriftelijk | Eenmalig |
| Stagiair | Docentbegeleider, Stagebegeleider | Inplannen Bedrijfsbezoeken | Email | Twee-malig |

**Gegevens betrokken personen**

**Stagebegeleider:**

Naam: Xander Steinmann

E-mailadres: [xander.steinmann@isaac.nl](mailto:xander.steinmann@isaac.nl)

Telefoonnummer: 040 215 53 66

**Docentbegeleider:**

Naam: Coen Crombach

E-mailadres: [c.crombach@fontys.nl](mailto:c.crombach@fontys.nl)

Telefoonnummer: 8850 89241

**Stagiair:**

Naam: Danny Gloudemans

E-mailadres: [danny.gloudemans@isaac.nl](mailto:danny.gloudemans@isaac.nl)

**Bijlage B: Planning**

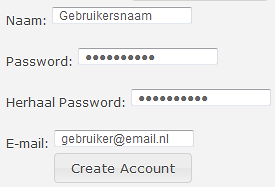
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Week | | Stage ISAAC 2012 | | |
| Stage | **School** | **Begin Datum** | **Activiteit** | **Te leveren producten** |
| 1 | 2 | 13-02-2012 | Installeren PC |  |
| 1-2 | 2-3 | 13-02-2012 | *Lezen dotCMS boek / kennismaking dotCMS* |  |
| 1-5 | - | 13-02-2012 | Maken van Project Initiation Document |  |
| 2-5 | 3-5 | 23-02-2012 | *Schrijven van een Plug-in voor dotCMS* |  |
| 4 | 4 |  |  |  |
| 5 | 5 |  | 1e bedrijfsbezoek docentbegeleider Coen Crombach |  |
| 5 | 5 | 12-03-2012 | Inleveren definitieve versie van PID | PID |
| 5 | 5 | 12-03-2012 | *Automatisch deployen van een plugin* |  |
| 6-7 | 6-7 | 19-03-2012 | *JUnit testing: testen m.b.v. Junit* |  |
| 6-7 | 6-7 | 19-03-2012 | Requirements/Use cases/Wat precies te testen? |  |
| 7-10 | 7-10 | 26-03-2012 | Vergaren kennis van deze componenten |  |
| 7-10 | 7-10 | 26-03-2012 | Onderzoek verwerken naar test componenten |  |
| 7-9 | 7-9 | 26-03-2012 | *Functies testen m.b.v. een scripttaal* |  |
| 8 | 8 |  |  |  |
| 9 | 9 |  | Terugkomdag naar Fontys |  |
| 10 | 10 |  |  |  |
| 10-11 | 10-11 | 16-04-2012 | *Koppelen aan een testomgeving (Hudson)* |  |
| 12-16 | 12-16 | 07-05-2012 | *Automatisch deployen van een versie van een plugin in een VMWare omgeving* |  |
| 13 | 13 |  |  |  |
| 14 | 14 |  |  |  |
| 15 | 15 |  |  |  |
| 16 | 16 |  | Inleveren scriptie | Scriptie |
| 16 | 16 | 04-06-2012 | *Het testen van het testsysteem* |  |
| 17 | 17 |  | 2e bedrijfsbezoek docentbegeleider Coen Crombach | Product / Overdracht |
| 18 | 18 |  | Afleveren afstudeerscriptie en definitieve portfolio |  |

# Ontwikkelingen CustomLogin Plug-in

In deze bijlage wordt er dieper op de ontwikkeling van de CustomLogin Plug-in ingegaan.

**Situatie februari 2012**

De CustomLogin Plugin is de eerste plug-in waarmee kennis is gemaakt tijdens de afstudeerstage. De situatie van de CustomLogin Plugin was toen als volgt.

Aan de front-end kant wordt er een formulier met de naam, wachtwoord en het emailadres ingevuld. Met deze informatie maakt de CustomLogin Plugin een customuser aan. Het voordeel van deze customuser, is dat er meer informatie kan worden opgeslagen dan in een normale dotCMS gebruiker. Het was echter wel zo dat er een dotCMS gebruiker gekoppeld moest worden aan de customuser zodat het mogelijk was om met de customuser in te loggen en gebruik te maken van de dotCMS functionaliteiten. De informatie werd in twee aparte tabellen opgeslagen. In het ene tabel, de customuser tabel, kwam de informatie te staan van de customuser en in de andere tabel, de dotcmsuser tabel, kwam de koppeling tussen de customuser en een dotCMS gebruiker.

De plug-in heeft de volgende functionaliteiten:

* Wijzigen van een account
* In- en uitloggen met een account
* Wachtwoord vergeten
* Wachtwoord wijzigen d.m.v. een sleutel

**CustomLogin Plugin I**

De CustomLogin Plugin was niet goed genoeg voor gebruik in productieomgeving. Dit kwam onder andere omdat de koppelingstabel niet nodig was, omdat deze informatie ook in de customuser tabel zou kunnen staan. Ook werd er niet gecontroleerd of het wachtwoord sterk genoeg was. Dit waren de twee aanpassingen die gedaan moesten worden tijdens de ontwikkeling van CustomLogin Plugin I.

Om de koppelingstabel te verwijderen, moesten de essentiële velden worden samengevoegd met de customuser tabel. Uit een kort onderzoek naar wat er nodig was om een dotCMS gebruiker in te loggen, bleek dat de gebruikersnaam van de dotCMS gebruiker voldoende was om in te loggen. Via de dotCMS API is het mogelijk om met de gebruikersnaam het wachtwoord op te vragen, waardoor er met de dotCMS gebruiker ingelogd kan worden.

|  |  |  |
| --- | --- | --- |
| Customuser tabel | | dotcmsuser tabel |
| website | failedLogins | website |
| name | dateCreated | username |
| email | dateLastLogin | password |
| encryptedPassword | dateReEnable |
| salt | dateForgotPasswordId |
| Enabled | forgotPasswordId |
| emailConfirmedId | passwordReset |

Hierdoor waren de velden website en password overbodig in de dotcmsuser tabel. Waardoor alleen het veld dotcmsuser erbij kwam in de customuser tabel.

Het controleren of het wachtwoord wel sterk genoeg is, gebeurt door middel van een reguliere expressie. Een reguliere expressie is een patroon waaraan een tekst moet voldoen, bijvoorbeeld dat er in een email adres een @, punt en drie gedeeltes tekst moeten zitten. Voor het wachtwoord is deze ((?=.\*\\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%]).{6,20}) reguliere expressie gebruikt die aan de volgende punten voldoet:

- Het wachtwoord moet minimaal één cijfer bevatten  
- Het wachtwoord moet minimaal één kleine letter bevatten  
- Het wachtwoord moet minimaal één hoofdletter bevatten  
- Het wachtwoord moet minimaal één symbool (@#$%) bevatten  
- Het wachtwoord moet minimaal 6 en maximaal 20 karakters lang zijn

De reguliere expressie staat in de plugin.properties. De plugin.properties is een bestand waarin de configuratie van de plug-in staat, deze is door middel van de PropertyMapper class uit te lezen. Het wachtwoord wordt gecontroleerd met behulp van een matcher. De matcher controleert of het wachtwoord voldoet aan de daarvoor ingegeven reguliere expressie. De code voor het controleren van het wachtwoord ziet er als volgt uit:

|  |
| --- |
| **private** **static** Pattern *pattern*;  **private** **static** Matcher *matcher*;  **public** **static** **boolean** validatePassword(String password) {  String expressie = PropertyMapper.*getInstance*().get("passwordPattern");  *pattern* = Pattern.*compile*(expressie);  *matcher* = *pattern*.matcher(password);  **if**(password == **null** || !*matcher*.matches()) {  **return** **false**;  }  **return** **true**;  } |

**CustomLogin Plugin II**

In de tweede fase van de ontwikkeling van de CustomLogin Plugin was het de bedoeling om de gebruikersinformatie op te slaan als content. Door het op te slaan als content zou het mogelijk zijn om alle informatie ook in de front-end te gebruiken. Om dit te realiseren moest het volgende worden aangepast:

* Customuser tabel moest worden verwijderd
* Gegevens van customuser moesten in nieuw content worden opgeslagen
* De identifier van het content moest worden opgeslagen in de sessie, zodat in de front-end gebruik kan worden gemaakt van de informatie uit het content

Voordat de customuser tabel werd verwijderd, werd er eerst uitgezocht hoe er nieuw content aangemaakt kon worden. Als eerste moest er een structure worden aangemaakt. Deze structure is de standaard voor hoe het content eruit komt te zien. In deze structure wordt exact aangegeven welke gegevens erin moeten komen te staan. Bijvoorbeeld de dateCreated moet van het type datum en tijd zijn.

De code voor het aanmaken van content in dotCMS gebeurt als volgt. De methode createUserIndotCMS wordt aangeroepen, waar ook de CustomUser aan mee wordt gegeven. In de CustomUser staat alle informatie die in het content moet komen te staan. De eerste stap is het ophalen van de structure. Is de structure succesvol opgehaald, dan worden de properties Live en StructureInode gezet. Deze twee properties zijn nodig omdat het content dan gebruikt kan worden in de front-end en het content bij de juiste structure wordt gezet. De volgende stap is om het content te vullen met de Customuser gegevens. Als laatste stap wordt het content doorgegeven aan de saveContentlet methode die ervoor zorgt dat het content daadwerkelijk wordt opgeslagen.

|  |
| --- |
| **private** **static** **void** createUserIndotCMS(CustomUser user) {  Structure structure = *getCurrentUserContentStructure*();  Contentlet contentlet = **new** Contentlet();  contentlet.setLive(**true**);  contentlet.setStructureInode(structure.getInode());  contentlet = *setContentletFieldsBasedOnUser*(contentlet, user);  *saveContentlet*(contentlet);  } |

De structure wordt op basis van de naam opgehaald met behulp van de StrutureFactory. De StructureFactory is een class binnen de dotCMS API. De StructureFactory geeft op basis van de naam, het structure object terug. Mocht de structure niet bestaan dan komt er een RuntimeException en stopt het programma. In de dotCMS log valt te lezen wat het probleem is.

|  |
| --- |
| **public** **static** Structure getCurrentUserContentStructure() {  Structure s = StructureFactory.*getStructureByVelocityVarName*(PropertyMapper.*getInstance*()  .get("structureName"));  **if** (s.getName() == **null** || s.getName().equals(**null**)) {  **throw** **new** RuntimeException("Invalid Structure, structure doesn't exists: " + . PropertyMapper.*getInstance*().get("structureName"));  }  **return** s;  } |

In de methode setContentletFieldsBasedOnUser wordt er gekeken of de veldnaam overeenkomt met de verwachte parameternaam. Komen de namen overeen dan wordt het veld gezet met de bijbehorende gebruikerswaarde. Uiteindelijk worden alle structure velden gecontroleerd, mochten er velden zijn waarop niet wordt gecontroleerd, dan wordt dit aangegeven in de log. Dit hoeft overigens geen fout te zijn, aangezien het mogelijk is om extra velden toe te voegen in het content die alleen gebruikt worden in de front-end en niet in de plug-in zelf.

|  |
| --- |
| **private** **static** Contentlet setContentletFieldsBasedOnUser(Contentlet contentlet, CustomUser user) {  **if**(!UtilMethods.*isSet*(contentlet.getStructureInode())) {  **throw** **new** RuntimeException("The structure inode must be set in the contentlet ");  }  List<Field> fields = FieldsCache.*getFieldsByStructureInode*(contentlet.getStructureInode());  **for** (Field f : fields) {  **if**(f.getVelocityVarName().equals(ParameterNames.*structureUserContentEmail*)){  contentlet.setStringProperty(f.getVelocityVarName(), user.getEmail());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentPassword*)) {  contentlet.setStringProperty(f.getVelocityVarName(), user.getEncryptedPassword());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentEmailConfirmedId*)) {  contentlet.setStringProperty(f.getVelocityVarName(), user.getEmailConfirmedId());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentFailedLogins*)) {  contentlet.setLongProperty(f.getVelocityVarName(), user.getFailedLogins());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentDateCreated*)) {  contentlet.setDateProperty(f.getVelocityVarName(), user.getDateCreated());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentDateLastLogin*)) {  contentlet.setDateProperty(f.getVelocityVarName(), user.getDateLastLogin());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentDateReenabled*)) {  contentlet.setDateProperty(f.getVelocityVarName(), user.getDateReEnable());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentDateForgotPasswordId*)) {  contentlet.setDateProperty(f.getVelocityVarName(), user.getDateForgotPasswordId());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentForgotPasswordId*)) {  contentlet.setStringProperty(f.getVelocityVarName(), user.getForgotPasswordId());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentEnabled*)) {  contentlet.setBoolProperty(f.getVelocityVarName(), user.isEnabled());  } **else** **if** (f.getVelocityVarName().equals(ParameterNames.*structureUserContentWachtwoord*)) {  contentlet.setStringProperty(f.getVelocityVarName(), "");  } **else** { Logger.*info*(CustomUserService.**class**, "Structure field "+f.getFieldName()+" was not set . in contentlet");  }  }  **return** contentlet;  } |

Als laatste stap wordt het content opgeslagen door de saveContentlet methode. In deze methode wordt door middel van de ContentletAPI, een class van de dotCMS API, het content opgeslagen.

|  |
| --- |
| **private** **static** **void** saveContentlet(Contentlet contentlet) {  ContentletAPI contentletAPI = APILocator.*getContentletAPI*();  UserAPI userAPI = APILocator.*getUserAPI*();  **try** {  Contentlet c = contentletAPI.checkin(contentlet, userAPI.getSystemUser(), **false**);  } **catch** (DotContentletValidationException e) {  **throw** **new** RuntimeException(e);  } **catch** (DotContentletStateException e) {  **throw** **new** RuntimeException(e);  } **catch** (IllegalArgumentException e) {  **throw** **new** RuntimeException(e);  } **catch** (DotDataException e) {  **throw** **new** RuntimeException(e);  } **catch** (DotSecurityException e) {  **throw** **new** RuntimeException(e);  }  } |

Uiteindelijk kwam er een groot probleem naar voren. In eerste instantie leek het namelijk niet mogelijk om content te updaten. Dit probleem werd opgelost nadat er met Xander naar was gekeken. Ook Xander wist in eerste instantie niet precies wat het probleem was, maar uiteindelijk bleek dat de inode ook gezet moest worden. Dit verhielp het probleem waardoor de code voor het updaten van het content er als volgt uit kwam te zien:

|  |
| --- |
| **public** **static** **void** updateUser(CustomUser user){  ContentletAPI contentletAPI = APILocator.*getContentletAPI*();  UserAPI userAPI = APILocator.*getUserAPI*();  Contentlet contentlet = **new** Contentlet();  **try** {  contentlet = contentletAPI.findContentletByIdentifier(user.getIdentifier(), **true**, 1, . userAPI.getSystemUser(), **false**);  } **catch** (DotContentletStateException e) {  **throw** **new** RuntimeException(e);  } **catch** (DotDataException e) {  **throw** **new** RuntimeException(e);  } **catch** (DotSecurityException e) {  **throw** **new** RuntimeException(e);  }  Structure s = *getCurrentUserContentStructure*();  contentlet.setStructureInode(s.getInode());    contentlet = *setContentletFieldsBasedOnUser*(contentlet, user);  contentlet.setLive(**true**);  contentlet.setInode("");  *saveContentlet*(contentlet);  } |

**CustomLogin Plugin III**

Voor de CustomLogin Plugin III moesten er drie grotere veranderingen worden doorgevoerd. Het content moest namelijk worden toegevoegd door middel van de standaard ‘voeg content toe’ functie in dotCMS, de overbodige velden moesten uit de plug-in worden gehaald en de plug-in moest gebruik gaan maken van de ISAAC mail service.

Het toevoegen van content door middel van de ‘voeg content toe’ functie van dotCMS, moest gebeuren door een pre-hook te gebruiken. Een pre-hook is een class die ervoor zorgt dat de code in de methode wordt uitgevoerd voordat de overige originele code wordt uitgevoerd. Als eerste wordt er gecontroleerd of het gaat om de customuser structure, is dit het geval dan wordt er gecontroleerd of het emailadres en het wachtwoord aan de validatie eisen voldoen. Daarna wordt het wachtwoord gehasht en in het content veld ‘encryptedPassword’ gezet, waarna het wachtwoord veld leeg wordt gemaakt.

|  |
| --- |
| @Override  **public** **boolean** checkin(Contentlet currentContentlet, ContentletRelationships relationshipsData, List<Category> cats, List<Permission> selectedPermissions, User user, **boolean** respectFrontendRoles) {  Logger.*info*(**this**.getClass(), "Checkin: " +currentContentlet.getStructure().getName());  **if**(checkIfCurrentContentletStructureIsTheCustomUserStructure(currentContentlet.getStructure())){  Logger.*info*(**this**.getClass(), "Checkin user via de SaveUserContentHook: " . +currentContentlet.getStringProperty(ParameterNames.*structureUserContentEmail*));  CustomUser customUser = **new** CustomUserImpl();  **try** {  **if**(!currentContentlet.getStringProperty(ParameterNames.*structureUserContentWachtwoord*).  isEmpty() && !"".equals(currentContentlet.getStringProperty("wachtwoord"))) {  customUser = CustomCreateAlterUser.*doCreateUser*(currentContentlet  .getStringProperty(ParameterNames.*structureUserContentWachtwoord*),  currentContentlet.getStringProperty(ParameterNames  .*structureUserContentWachtwoord*),currentContentlet  .getStringProperty(ParameterNames.*structureUserContentEmail*)); . currentContentlet.setStringProperty(ParameterNames.*structureUserContentPassword*, . customUser.getEncryptedPassword());  . . currentContentlet.setStringProperty(ParameterNames.*structureUserContentWachtwoord*, "");  } **else** {  customUser.setEmail(currentContentlet.getStringProperty(ParameterNames  .*structureUserContentEmail*));  }    } **catch** (DotRuntimeException e) {  Logger.*error*(**this**, "Exceptie DotRuntime: " + e.getMessage());  } **catch** (InvalidUserDataException e) {  Logger.*error*(**this**, "Exceptie InvalidUserDataException: " + e.getMessage());  }  }  **return** **true**;  } |

In de checkIfCurrentContentletStructureIsTheCustomUserStructure methode wordt gecontroleerd of het huidige content van het type customUser is.

|  |
| --- |
| **public** **boolean** checkIfCurrentContentletStructureIsTheCustomUserStructure(Structure s) {  Structure structure = StructureFactory.*getStructureByVelocityVarName*(PropertyMapper  .*getInstance*().get("structureName"));  **if** (s.getName().equals(structure.getName())) {  **return** **true**;  }  **return** **false**;  } |

Verder moesten de hieronder genoemde velden worden verwijderd uit het content omdat deze overbodig waren voor het project waarin deze plug-in gebruikt zou worden.

* Website
* Naam
* dotCMSUser
* Salt
* PasswordReset

De website en de naam zijn overbodig omdat het emailadres uniek moest zijn, waardoor het niet meer noodzakelijk was om een naam te gebruiken als inlognaam of dit te valideren op website niveau.

De dotCMSUser en salt zijn niet meer in het content nodig, omdat deze in de plugin.properties zijn gezet. De dotCMSUser is in de plugin.properties gezet omdat dit voor elke gebruiker hetzelfde moest zijn, waardoor het overbodig is om dit in het content te zetten. Daarbij kan de gebruiker het zelf veranderen in het content wat voor problemen kan zorgen, mocht de gebruiker het bijvoorbeeld in de Administrator veranderen. De salt staat ook in de plugin.properties, ook dit is voor de veiligheid. Alhoewel het beter is om per user een aparte salt te gebruiken is het nog slechter om de salt op te slaan in hetzelfde content als het gehashte wachtwoord.

Het passwordReset veld zorgde ervoor dat er niet meer ingelogd kon worden wanneer iemand een nieuw wachtwoord aanvroeg. Maar als de gebruiker het wachtwoord wel weer weet moet het mogelijk zijn dat de gebruiker kan inloggen met zijn oude wachtwoord. Door passwordReset te verwijderen was dit mogelijk.

Het verwijderen van de overbodige velden ging probleemloos, echter nam het veel tijd in beslag om te testen of de functionaliteiten nog naar behoren werkte.

Als laatste moet het mogelijk zijn om gebruik te maken van de ISAAC Mail Service. De bedoeling was dat wanneer er een formID werd meegegeven dat de ISAAC Mail Service wordt gebruikt. Hiervoor had Maarten een API geschreven waardoor er maar enkele regels code nodig waren. De volgende code verzend een email via de ISAAC Mail Service:

|  |
| --- |
| **public** **void** doPost(HttpServletRequest request) {  Pair<EmailForm, EmailBuilder> pair;  pair = RequestHandler.*constructFromRequest*(request, **null**);  EmailForm form = pair.getFirst();  EmailBuilder email = pair.getSecond();  form.send(email);  } |

Voordat deze stap geïmplementeerd kon worden, moest er eerst worden uitgezocht hoe de ISAAC Mail Service exact werkt. Aangezien er nog geen ervaring met Mail Services was, werden er eerste enkele testapplicaties gemaakt, waarin verschillende scenario’s werden getest aan de hand van de handleiding. Nadat de handleiding succesvol was doorlopen werd de ISAAC Mail Service toegevoegd aan de CustomLogin plug-in.

# Batch bestand voor automatisch testen deployen

In deze bijlage bevind zich het batchbestand, wat voor Jenkins werd gebruikt om meer kennis te krijgen over het automatisch deployen van een plug-in. In hoofdstuk 6 sectie 1 van de scriptie is er meer informatie over te vinden.



