Self-service functionaliteit

J2EE, open source en Oracle Appserver

Tijdens het schrijven van dit artikel is de website werk.nl net 'live' gegaan, in het weekend van 11 en 12 november jl. Dit artikel beschrijft de self-service componenten die zijn gebouwd met J2EE-technologie, open-sourcetechnologie en de Oracle Application Server. Ook wordt er ingegaan op de toegankelijkheids-eisen van de website, performance-logging en de scheiding in het ontwikkelproces van PL/SQL- en Java.

Figuur 1. De website werk.nl is het 'uithangbord' van het Centrum voor Werk en Inkomen.

Eerst even wat achtergrondinformatie van de website werk.nl. De website is het uithangbord van het CWI (Centrum voor Werk en Inkomen) wat voorheen de arbeidsbureaus waren. Via de website kunnen werkzoekenden op zoek gaan naar passende vacatures. Werkgevers kunnen op de website vacatures

Figuur 2. Een deel van de homepage van werk.nl.

publiceren en op zoek gaan naar geschikt personeel. Vanuit het CWI worden werklozen actief benaderd om zich in te schrijven op de website om zo de kans te vergroten op een nieuwe baan. Een nieuwe functie van de website is dat nu ook informatie en nieuws rondom het CWI zelf wordt weergegeven.

Herbouw

De website is niet nieuw, maar bestaat reeds enkele jaren. Begin 2005 is er besloten om de voorkant te gaan herbouwen en op bepaalde plekken nieuwe functionaliteit toe te voegen. In totaal hield dit in dat er ongeveer tachtig self-service componenten moesten worden gebouwd.

Uitgangspunten

Bij het bouwen van de website werd een aantal uitgangspunten gehanteerd:

1) Oracle-database van de 'oude' website moest worden hergebruikt.
2) Oracle Application Server (10.1.2) wordt gebruikt
3) Ontwerpbureau heeft de look-and-feel aangeleverd van de nieuwe website, ook zijn er wire-frames gemaakt van de te maken functionaliteit.
4) De website moet voldoen aan de eisen van drempelvrij.nl voor toegankelijkheid (hierover zometeen meer).

Ook zijn vanuit de 'oude' website getallen bekend rondom het gebruik van de website, en aantallen data:

Aantal cv's	300.000
Aantal vacatures	60.000
Hits per dag	2,5 miljoen

Proof of Concept

In juli 2005 is er begonnen met een POC-fase ('Proof of Concept') voor de self-service functionaliteit van de website. Deze fase was bedoeld om technologiekeuzes te valideren en het opzetten van een werkwijze voor de bouwfase. Na het afronden van de POC-fase is er begonnen met de daadwerkelij-

ke bouwfase en is het team uitgebreid. Het uiteindelijke team bestond uit de volgende disciplines:

1 teamlead (Oracle / J2EE)
1 Oracle-ontwikkelaar
4 J2EE-ontwikkelaars
1 tester
1 HTML-ontwikkelaar / ontwerper

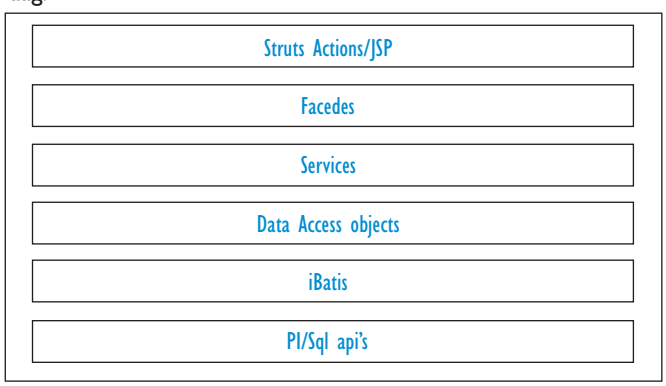
Tijdens de POC-fase is een aantal technologiekeuzes gemaakt voor het bouwen van de nieuwe website (zie tabel).

Technologie	Functie binnen de applicatie
Struts/JSP	View-implementatie
Ant	Build tool
Quartz	scheduling functionaliteit
Spring	IoC (Inversion of Control) framework
iBatis	O/R mapping framework
JUnit	Testframework
Jmeter	framework voor performance testen
Eclipse	IDE
SubVersion / Subclipse	Versiebeheer / plugin voor Eclipse

Tegenwoordig zie je bij veel projecten het Spring-framework opduiken, als je de bovenstaande technologiekeuzes bekijkt is Spring een echt voordeel. Dit omdat Spring standaard-integratiepunten biedt voor Struts, Quartz, iBatis, JUnit. De zogenaamde 'plumbing'-code hoef je nu niet meer zelf te schrijven. Functionaliteit voor foutafhandeling is al standaard aanwezig.

Opdeling in lagen

De applicatie is onderverdeeld in verschillende lagen, zoals in Figuur 3 wordt weergegeven. Hieronder een toelichting per laag:



Figuur 3. De applicatie is onderverdeeld in verschillende lagen.

Struts Actions/JSP: in deze laag zijn de Struts-actions gedefinieerd welke de 'V' en 'C' implementeren uit het MVC-model.

Facades: actions kunnen niet direct 'services' aanroepen, dit gaat altijd via een façade. Via een façade kan ook functionaliteit gedeeld worden over actions heen.

Services: een service coördineert het ophalen/opslaan van data uit de verschillende databronnen (LDAP, Oracle-database, webservices)

Data Access Objects (DAO): de verschillende dao's doen de daadwerkelijke implementatie van het ophalen en opslaan van data. Per functionaliteit is er een aparte implementatie, bijvoorbeeld iBatis, een webservice-implementatie.

iBatis: de O/R mapping is geïmplementeerd via het iBatis-framework. Deze roept de PL/SQL API's aan en geeft het eventuele resultaat terug in objecten voor de DAO-laag.

PL/SQL API's: wanneer er data worden opgehaald en opgeslagen uit de database (Oracle), gebeurt dit altijd via PL/SQL API's.

De rol van Spring is dat dit framework de verschillende lagen aan elkaar 'wired', via de verschillende configuratiefiles kunnen componenten 'geïnjecteerd' worden. Zonder dat de ontvangende class weet welke implementatie erachter hangt, of objecten hoeft te instantiëren. Hieronder is een voorbeeld te zien dat de 'WerknemerService' via Spring verschillende componenten krijgt 'geïnjecteerd', bijvoorbeeld de 'werknemerDao' en de 'securityService'.

```
<bean id="werknemerService" class="nl.cwinet.werknl.service.
WerknemerService">
          <constructor-arg><ref bean="werknemerDao"/></constructor-arg>
          <constructor-arg><ref local="property"/></constructor-arg>
          <constructor-arg><ref bean="emailService"/></constructor-arg>
          <constructor-arg><ref bean="securityService"/></constructor-
arg>
          <constructor-arg><ref bean="matchProfielService"/></construc-
tor-arg>
          <constructor-arg><ref bean="eliseService"/></constructor-arg>
          <constructor-arg><ref bean="cmsService"/></constructor-arg>
          <constructor-arg><ref bean="serviceLogger"/></constructor-
arg>
          <constructor-arg><ref bean="commonDao"/></constructor-arg>
          <constructor-arg><ref bean="werknemerCache"/></constructor-
arg>
</bean>
```

Voorts biedt Spring standaard integratiepunten, bijvoorbeeld voor Quartz. Hiermee kunnen jobs vrij eenvoudig worden geconfigureerd, en hoeft dus geen code worden gemaakt om het scheduling-mechanisme te activeren binnen de applicatie. Hieronder een voorbeeld een Quartz-job kan worden geconfigureerd met Spring.

```
<bean id="emsWerkgeverJob" class="org.springframework.scheduling.quartz.
MethodInvokingJobDetailFactoryBean" lazy-init="false">
          <property name="targetObject" ref="werkgeverService"/>
          <property name="targetMethod" value="createMatchEmails"/>
          <property name="concurrent" value="false"/>
</bean>
```

Scheiden van verantwoordelijkheden

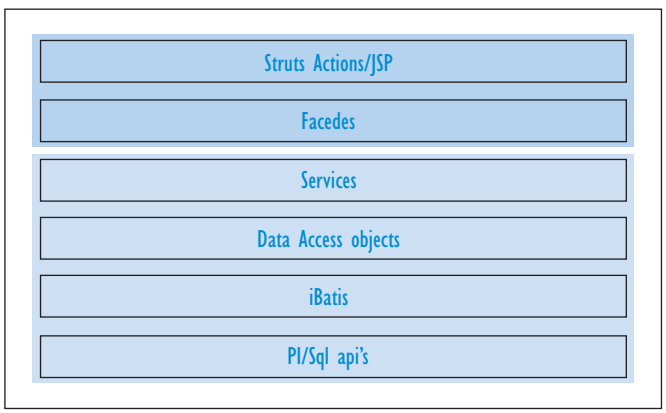
De applicatie bestaat uit verschillende lagen met verschillende technologieën. Er zijn voornamelijk twee technologieën die met elkaar versmolten moeten worden, de JSP/Struts actions met die van iBatis/PL/SQL API's. Het is moeilijk om mensen te vinden die van beide technologie-stacks alles weten en dus 'verticaal' kunnen ontwikkelen. Daarom is er in het project besloten om verantwoordelijkheden te scheiden en 'horizontaal' te ontwikkelen. Er zijn twee blokken gedefinieerd, deze zijn als volgt verdeeld:

Service-georiënteerd: alles beginnend bij de service-laag tot en met de implementatie van PL/SQL-packages.
Action-georiënteerd: alles tot de service-laag, dus JSP, Actions.

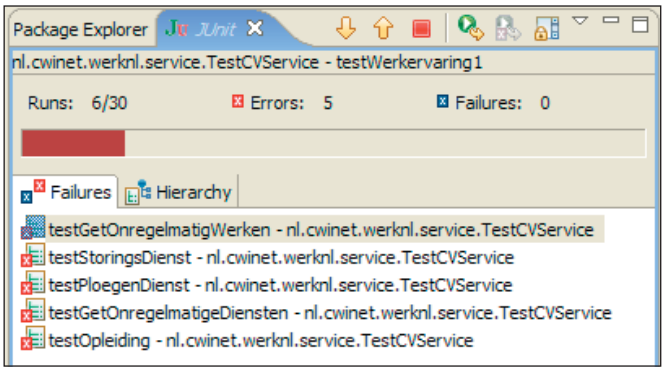
Binnen het ontwikkelteam kan iemand een 'service-ontwikkelaar' zijn, of een 'action-ontwikkelaar'. Vanzelfsprekend weet de 'service-ontwikkelaar' veel meer rondom PL/SQL en Oracle-databases dan een 'action-ontwikkelaar'. Een 'action-ontwikkelaar' ziet vanuit zijn perspectief enkel de service-laag, en niet de PL/SQL-implementatie hierachter.

Testen van services

Belangrijk bij het scheiden van verantwoordelijkheden is dat het koppelpunt 'service' wel moet werken wanneer de 'action-ontwikkelaar' data wil ophalen uit de service-laag. Hiervoor biedt JUnit uitkomst. We hebben gekozen om met JUnit alle service-methodes te gaan testen, met minimaal twee test-methodes per service-methode. Met deze testen valideren we dan achtereenvolgens de service-laag, dao-laag, iBatis en PL/SQL



Figuur 4. In het project werd besloten de verantwoordelijkheden te scheiden en 'horizontaal' te ontwikkelen.



Figuur 5. Met JUnit worden alle service-methodes getest, met minimaal twee test-methodes per service-methode.

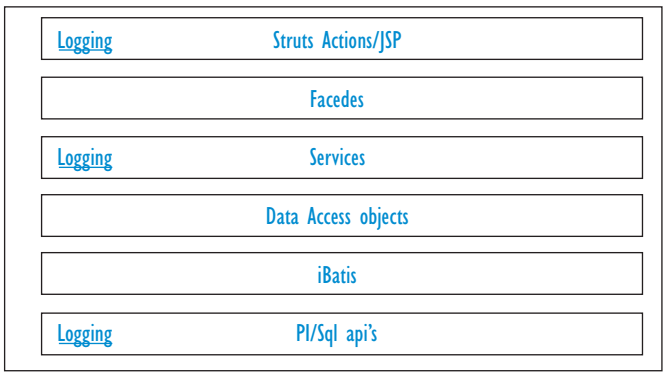
API's. Het grote voordeel is dat naarmate het project vordert je nog steeds kunt valideren of de eerder gemaakte service-functionaliteit nog steeds werkt. Gewoon de verschillende testen opnieuw draaien voor de verschillende services. Wanneer de test een positief resultaat heeft, kan de 'service-ontwikkelaar' zijn werk overdragen, en heeft zo 100 procent zekerheid dat de 'action-ontwikkelaar' meteen aan de slag kan, en niet voor hem vreemde SQLExceptions naar voren krijgt in de applicatie. Het meeste voordeel van deze werkwijze is wanneer je vanaf het begin de testen maakt, en niet achteraf.

Performance

Voor een website als werk.nl met vele bezoekers per dag is het zeer belangrijk dat er constant wordt gekeken naar de performance van de applicatie. Dit was dan ook een belangrijke eis bij het opzetten van de applicatie. Toch blijft het tijdens het ontwikkelen moeilijk in te schatten op welke vlakken mogelijke performance-issues naar voren gaan komen.
Daarom hebben we gekozen voor een aanpak van 'performance-logging' op verschillende lagen in de applicatie. Op de volgende lagen is performance logging toegevoegd:

PL/SQL laag, bij het uitvoeren van een PL/SQL API wordt gelogd hoe lang een functie over een bepaald stuk functionaliteit heeft gedaan.
Service laag, vanuit de service-laag wordt gelogd hoe lang de totale operatie (inclusief PL/SQL API) heeft geduurd.
Actions laag, per action wordt gelogd hoelang deze over de Action heeft gedaan (inclusief service-call).

Het voordeel van het op voorhand toevoegen van performance-logging is dat je niet op zoek hoeft te gaan naar componenten die mogelijk langzaam zijn. Deze komen direct naar voren bij het bekijken van logging. Ook komen daar bijvoorbeeld componenten naar voren waar je in het begin helemaal niet aan zou denken. Bijvoorbeeld dat het gebruik van java.net.URLEncoder relatief veel cpu-tijd kost, hiervoor gebruiken we nu org.apache.commons.codec.net.URLCodec.



Figuur 6. Aan een aantal lagen is performance-logging toegevoegd.

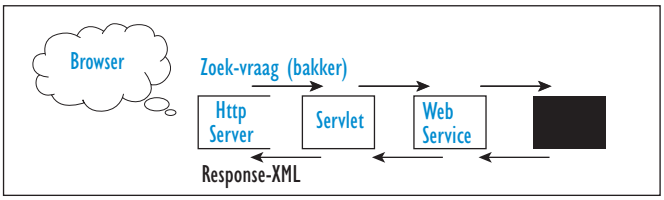
Wanneer je geen performance logging hebt geïntroduceerd, dan moet je verschillende deployments doen met 'ad-hoc debugging' voor het achterhalen van pijnpunten binnen de applicatie. De logging is zo opgezet, dat deze run-time kan worden uit/aangezet, en dat een bepaald niveau (aantal ms) kan worden ingesteld. Vanaf dit niveau worden de methodes gelogd.

Ajax en de WebCache -

Binnen de website is ook een aantal Ajax componenten opgenomen, voor het opzoeken van adresgegevens en beroepsinformatie. Dit is zeer gebruiksvriendelijk vanwege de directe feedback naar de gebruiker. Die ziet geen 'Zoek'-buttons meer. Tijdens de implementatie van deze Ajax-componenten hebben we nadrukkelijk gekeken naar de mogelijke performance-impact van de vele Ajax-calls welke worden uitgevoerd op de server. In Figuur 8 zien we hoe zo'n call er uitziet.

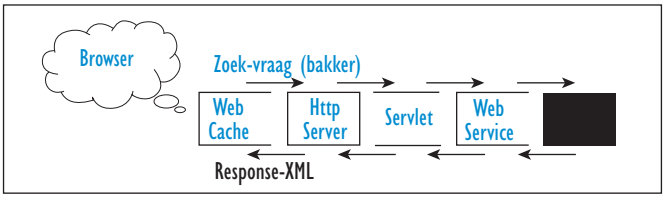
In dit plaatje wordt er een Ajax-call vanuit de browser gedaan. Deze gaat via de Http-Server naar de servlet. Deze roept vervolgens weer een webservice aan voor het daadwerkelijk opvragen van de gegevensbron (in dit geval zoeken naar beroepsinformatie). Uiteindelijk ontvangt de browser de gegenereerde xml, die via het XMLHttpRequest wordt weergegeven aan de eindgebruiker.
Zoals te zien is in het plaatje, moet een enkele Ajax-call toch

Figuur 7. Binnen de site is een aantal Ajax componenten opgenomen voor het opzoeken van adresgegevens en beroepsinformatie.



Figuur 8. Een Ajax-call vanuit de browser.

een aantal stappen afleggen voordat er antwoord kan worden gegeven. Om het aantal stappen te verminderen, hebben we voor deze functionaliteit de WebCache geïntroduceerd. In het plaatje ziet dat er als volgt uit:



Figuur 9. WebCache het verkeer afvangt voor de Http-Server.

Wat in het plaatje te zien is, is dat de WebCache het verkeer afvangt voor de Http-Server, wanneer de WebCache een bepaalde zoekvraag reeds in de cache heeft, wordt er direct de response-xml teruggegeven. Dit scheelt enorm in de processing op de andere componenten aan de achterkant van de flow (applicatie-server en database). Uiteraard is zo'n mechanisme pas echt goed toe te passen wanneer het gaat om echt statische data. Wat verder nog geïmplementeerd is om de WebCache optimaal te benutten is een zogenaamde 'rampup' job. Deze vult de WebCache met alle beroepsinformatie waarop gebruikers kunnen zoeken. Hiermee voorkomen we dat het voor de gebruiker 'de eerste keer' langzaam is. Met deze rampup job versturen wij zo'n 100.000 Http-requests met daarin zoekvragen rondom beroepsinformatie. Met deze zoekvragen wordt dan automatisch de WebCache gevuld.

Toegankelijkheidseisen

Zoals aangeven in het begin was een belangrijk uitgangspunt dat de website moest voldoen aan de eisen rondom 'drempelvrij.nl'. Dit is een set eisen rondom het toegankelijk maken van websites voor iedereen, inclusief mensen met een functiebeperking en senioren. Vanuit de organisatie achter 'drempelvrij.nl' is er voor de livegang een audit gedaan om te controleren of de website voldoet aan de gestelde eisen rondom toegankelijkheid. Hieronder een aantal concrete eisen waaraan de website moet voldoen:

- Javascript
  De website dient te functioneren wanneer de browser geen Javascript ondersteunt, speciale browsers voor gehandicapten ondersteunen dit namelijk niet

- Images
  Iedere <img >-tag dient voorzien te zijn van een 'alt'-attribuut, browsers voor blinden kunnen dan voorlezen aan de slechtziende wat een bepaald image voor doel heeft.
- Taalovergangen
  Wanneer er taalovergangen zijn in teksten (bijvoorbeeld 'Working in the Netherlands'), dient dit te worden aangegeven. Wanneer teksten worden voorgelezen, weet de spraakcomputer wanneer er bijvoorbeeld een Engelse tekst moet worden voorgelezen. Via een -tag kan dit worden aangegeven, <span lang=''gb''>.
- New window
  Wanneer er een nieuw browser-window wordt geopend, dient dit van tevoren te worden aangekondigd. Slechtzienden weten zo van te voren dat er een nieuw window wordt geopend.

Belangrijk is wel dat eisen bekend zijn voordat er met het project wordt begonnen. Dat heeft ons veel tijd bespaard, zeker eisen rondom 'Non-javascript' ondersteuning zijn zeer lastig om deze achteraf te implementeren.
Ook een HTML-ontwerper heeft ons enorm geholpen bij Javascript en HTML/CSS-issues. Een gemiddelde ontwikkelaar weet toch te weinig van echte vormgeving en gebruikerservaring. Zeker voor een gebruiksvriendelijke website voor een grote doelgroep is het noodzakelijk om een dergelijke functie te hebben binnen je ontwikkelteam.

Conclusie

Zoals uit het artikel blijkt was het werk.nl project een uitermate interessant project, zowel technisch als functioneel. Vooral de scheiding van verantwoordelijkheden tussen Java en PL/SQL, in combinatie met JUnit heeft het ontwikkelproces enorm versneld, en de kwaliteit verhoogd. Voor mij persoonlijk was het enorm uitdagend om leiding te geven aan een groot ontwikkelteam met zeer goede mensen. De eerste fase is nu opgeleverd, en volgende releases zijn in ontwikkeling en worden begin volgend jaar in productie gebracht.

Feike Visser is Senior Software Engineer bij Cumquat Information Technology (www.cumquat.nl) in Zeist, en heeft tien jaar ervaring met Oracle-technologie, sinds 2000 houdt Feike zich bezig met J2EE in combinatie met Oracle. Vragen en opmerkingen over dit artikel kunnen gestuurd worden aan feike@cumquat.nl.