Scriptie

Vergelijking Apache Beehive, Spring MVC en Struts 2.

Versie	: 1.1 **FINAL**
Datum	: 7 maart 2007
Gewijzigd	: 30 mei 2007
Naam	: Rik Smith
Studentnr.	: 1173566
Opleiding	: Informatica Voltijd
Periode	: februari – juni 2007
1ste Examinator	: Dhr. E. Gerlofsma

Versiebeheer

Versie	Datum	Opmerkingen
0.1 **FORMAT**	07-03-07	Eerste versie
0.2 **FORMAT**	19-03-07	Aanpassingen Feedback van Leo v/d Meulen verwerkt
0.3	23-03-07	Eerste invulling
0.4	28-03-07	Layout fixes
0.5	17-04-07	Activiteiten Vooronderzoek Prototype
0.6	10-05-07	Layout fixes Onderzoek 0.15 er ingeplakt
0.7	21-05-07	Feedback van Armin Bauer verwerkt Management Samenvatting Over Capgemini
0.8	25-05-07	Code uit Hoofdstuk 6 gehaald, alle code, configuratie en jsp's van de prototypes staan nu in de bijlage Feedback van Armin Bauer verwerkt: Management Samenvatting, Evaluatie
1.0 **FINAL**	29-05-07	Feedback Leo van der Meulen verwerkt. Document naar Final.
1.1 **FINAL**	30-05-07	Layout fixes

Voorwoord

Voor u ligt mijn scriptie geschreven in het kader van mijn afstudeerstage voor mijn studie Informatica Voltijd aan de Hogeschool Utrecht. Deze afstudeerstage heb ik uitgevoerd voor Capgemini Nederland.

Allereerst wil ik Capgemini Nederland bedanken voor het aanbieden van deze afstudeerplaats. Ik werd al vroeg door Capgemini benaderd met de vraag of ik interesse had in een afstudeeropdracht. Na een korte e-mail wisseling volgde een gesprek met Armin Bauer en Jaques van Heijst en een capaciteiten test inclusief de “Occupational Personality Questionaire”. Hierna kreeg ik te horen of ik al dan niet was aangenomen. Ondanks dat ik nipt voor zowel de capaciteiten test als de OPQ niet slaagde, had Armin Bauer vertrouwen in mij en is de afstudeerplek mij aangeboden. Dank hiervoor!

De volgende stap was bij de opleiding goedkeuring aanvragen. De opleiding heeft de opdracht niet in een keer goedgekeurd, maar in overleg met de afstudeerbegeleider Eric Gerlofsma is dit alsnog gebeurd. Ik wil Eric Gerlofsma bedanken voor de constructieve bijdrage aan dit afstudeerproject. Door zijn inhoudelijke kennis van zake heb ik constructieve gesprekken met hem gevoerd over de resultaten en ook ideeën op gedaan voor de te volgen route.

Ook wil ik Akil Kömür bedanken voor het regelmatig doorlezen van mijn documenten. Dankzij jou staan er een stuk minder taalfouten in de documenten.

Als laatste wil ik Armin Bauer en Leo van der Meulen bedanken voor de begeleiding tijdens mijn afstuderen. Als ik vragen had kon ik altijd bij jullie terecht en op de opgeleverde producten kreeg ik altijd constructieve feedback. Ook in de begeleiding kon Leo van der Meulen altijd wel een gaatje in de agenda vinden om in persoon de resultaten door te spreken en waar nodig bij te sturen.

Daarnaast wil ik alle overige directe collega’s bedanken voor de goede sfeer tijdens mijn afstudeerstage.

Rik Smith
Utrecht, 29 mei 2007

Managementsamenvatting

De doelstelling van dit onderzoek is: Inzicht krijgen in de werking van de drie frameworks (Beehive, Spring MVC en Struts 2), om zo aan klanten een onderbouwde aanbeveling te kunnen geven over het te gebruiken framework.

De eerste stap in deze afstudeerstage was het opstellen van een lijst met beschikbare Java EE webframeworks. Uit deze lijst zijn in overleg met de opdrachtgever negen frameworks gekozen. Dit zijn: Beehive, Cocoon, JSF, Maverick, MyFaces, Spring MVC, Struts 2, Tapestry en Wicket. Deze keuze is hoofdzakelijk gebaseerd op bekendheid en of er recente ontwikkel activiteiten aan het framework te ontdekken waren. Uit deze negen frameworks zijn op basis van opgestelde selectiecriteria drie frameworks gekozen om verder te onderzoeken.

Om alle informatie die benodigd is voor de selectie te verzamelen, is er voor deze negen frameworks een vooronderzoek uitgevoerd. In dit vooronderzoek is per framework gekeken naar de ontwikkeling, beschikbare informatie, vraag naar de frameworks op de arbeidsmarkt. Ook is er gekeken naar het aantal medewerkers binnen Capgemini Nederland dat de frameworks op als competentie op hun CV hebben staan. Dit vooronderzoek leverde alle informatie die benodigd was om de selectiecriteria toe te passen. Als aanbeveling kwam uit het vooronderzoek om de frameworks JSF, Spring MVC en Struts 2 te onderzoeken. Omdat de opdrachtgever veel opdrachten doet met producten van het bedrijf BEA, waar Beehive in wordt gebruikt, is besloten om toch ook Beehive te onderzoeken. Hierdoor is er één framework afgevallen, dit is JSF. De uiteindelijk geselecteerde frameworks zijn: Beehive, Spring MVC en Struts 2.

Het onderzoek bestaat uit drie delen: Architectuur van de frameworks, prototypes en de vergelijking. Per framework is allereerst de architectuur in kaart gebracht. Vervolgens is er een prototype ontworpen dat met alledrie de frameworks gebouwd is. Het prototype is een eenvoudige bibliotheek applicatie. In deze applicatie kunnen boeken en klanten worden beheerd en boeken uitgeleend en ingenomen. Elk prototype communiceerd met dezelfde back-end, namelijk EJB 3 Session en entity beans. De schermen van de prototypes zijn zo simpel mogelijk gehouden, alleen wanneer een framework handige extra's biedt (zoals een kalender om een datum in te voeren) zijn deze in het scherm opgenomen. Hierdoor zijn de schermen vrijwel niet van elkaar te onderscheiden.

In de vergelijking zijn de frameworks vergeleken op configuratie, code, functionaliteit, flexibiliteit, performance en testbaarheid. Voor deze onderdelen is het prototype gebruikt als referentie.

Op het gebied van configuratie komt Struts 2 als beste uit de bus. Struts 2 heeft de minste configuratie door het gebruik van veel standaard waarden. Na Struts 2 volgt Beehive, deze heeft meer configuratie doordat er vaak dezelfde configuratie gedaan moet worden. Als laatste volgt Spring MVC. Deze heeft doordat er voor formulieren extra configuratie nodig is en omdat de configuratie wat complexer is een lagere beoordeling.

Bij de vergelijking op code niveau komt Struts 2 ook als beste uit de bus. Struts 2 gebruikt eenvoudige, niet complexe code, de code voor een actie is ook erg kort. Als tweede volgt Spring MVC, deze heeft een iets langere code en is ook iets complexer. Beehive heeft veruit de langste code door het gebruik van annotaties voor de configuratie. Door de hoeveelheid annotaties wordt de code erg complex. Als we de annotaties buiten beschouwing laten is Beehive gelijk met Spring MVC.

Struts 2 biedt de meeste functionaliteit. Er zitten vele extra's in en zowel validatie als security is goed geregeld. Beehive heeft minder extra's dan Struts 2 maar biedt ook enkele extra's. Validatie is goed geregeld en security zelfs zeer goed. Spring MVC biedt geen extra's. Validatie is goed geregeld maar security biedt Spring MVC standaard niet. Er is een security framework "Acegi" voor Spring, dit is niet gebruikt of uitgebreid onderzocht, maar hiermee zou de security wel goed geregeld zijn.

Als meest flexibel komt Struts 2 uit de bus. Dit framework is vrij eenvoudig in configuratie files aan te passen zonder de code aan te hoeven passen. Na Struts 2 volgt Spring MVC. Dit framework heeft een vastere koppeling tussen de controller en de jsp pagina. Hierdoor is dit framework iets moeilijker aan te passen. Als laatste volgt Beehive, dit framework is niet aan te passen zonder wijzigingen in de code. Afgezien daarvan is het framework flexibeler dan Spring in de koppeling met de view.

Spring MVC heeft veruit de beste performance. Beehive volgt een factor 1,4 langzamer, en als laatste volgt Struts 2, deze is een factor 1,7 langzamer dan Spring MVC.

Struts 2 is het beste testbaar. Dit komt omdat de code het minst complex is en de back-end goed te vervangen is door een surrogaat backend. Spring MVC volgt als tweede, deze is door de iets complexere code moeilijker te testen, de backend is net als Struts 2 goed te vervangen. Beehive volgt als laatste, de code is goed te testen maar de back-end is zeer lastig te vervangen.

Op het onderdeel toolondersteuning komen Beehive en Spring MVC als beste uit de bus. Voor Struts 2 zijn geen tools beschikbaar.

De conclusie van dit onderzoek is dat op technisch vlak Struts 2 als beste uit de bus komt. Maar door het gebrek aan toolondersteuning en de tegenvallende performance is het voorlopig nog niet aan te bevelen om dit in een project te gebruiken. Spring MVC scoort technisch iets minder maar dit framework heeft zich in de loop van de tijd al bewezen en er is goede toolondersteuning voor beschikbaar, het is aan te raden dit framework in een keuze mee te nemen. Beehive scoort minder dan Spring MVC maar ook dit framework kan goed in een keuze worden meegenomen.

Inhoudsopgave

1. Organisatie

1.1. Capgemini

Wie Capgemini is

Capgemini heeft meer dan 59.000 werknemers en is de mondiale marktleider in consulting, technologie, outsourcing en local professional services.

Het hoofdkantoor van Capgemini is gevestigd in Parijs. Onze regionale kantoren bevinden zich in Noord-Amerika, Noord-Europa & Azië en het Pacifisch gebied, evenals Centraal en Zuid-Europa. Sogeti is een dochteronderneming, gespecialiseerd in local professional services.

Consulting

Op basis van diepgaande sectorkennis en bedrijfsprocessen biedt Capgemini consulting-services. Hiermee levert Capgemini een essentiële bijdrage aan de bedrijfstransformatie en economische prestaties van klantorganisaties. Volgens onze holistische benaderingswijze bezien wij de organisatie van onze klant en de markt waarin deze actief is. Zo bepalen wij samen met de klant groeistrategieën op zowel de korte als de lange termijn. De collaboratieve benadering van Capgemini is geslaagd wanneer wij in staat zijn een gezamenlijke visie te realiseren.

Technology

Capgemini ontwerpt en integreert technologische oplossingen, ontwikkelt innovaties en transformeert de technische omgevingen van klanten. Deze services concentreren zich op systeemarchitectuur, -integratie en -infrastructuur. Technology-services en consulting-services komen samen op het moment dat de upgrade en transformatie van IT-systemen direct zijn gekoppeld aan de organisationele en strategische prioriteiten van de klant. Capgemini werkt samen met strategische alliantiepartners om de issues en uitdagingen van onze klanten beter te kunnen aanpakken.

Outsourcing

Inspelend op de behoefte van de klant is Capgemini in staat de volledige of gedeeltelijke verantwoordelijkheid op zich te nemen voor het management van de IT-middelen. In ons brede aanbod van diensten staat expertise in het management van IT-systemen en bedrijfsprocessen evenals prijsflexibiliteit centraal. Met als resultaat: de verhouding tussen kosten en prestaties bij onze klanten maximaliseren. Outsourcing van bedrijfsprocessen vormt één van de kernactiviteiten van Capgemini. Dit stelt de klant in staat belangrijke ondersteuningsfuncties - zoals boekhouding of inkoop - uit te besteden. Hierin wordt onze collaboratieve benadering het meest tastbaar voor onze klanten.

Wat Capgemini doet

Consulting

Op basis van diepgaande sectorkennis en bedrijfsprocessen biedt Capgemini consulting-services. Hiermee levert Capgemini een essentiële bijdrage aan de bedrijfstransformatie en economische prestaties van klantorganisaties. Volgens onze holistische benaderingswijze bezien wij de organisatie van onze klant en de markt waarin deze actief is. Zo bepalen wij samen met de klant groeistrategieën op zowel de korte als de lange termijn. De collaboratieve benadering van Capgemini is geslaagd wanneer wij in staat zijn een gezamenlijke visie te realiseren.

Technology

Capgemini ontwerpt en integreert technologische oplossingen, ontwikkelt innovaties en transformeert de technische omgevingen van klanten. Deze services concentreren zich op systeemarchitectuur, -integratie en -infrastructuur. Technology-services en consulting-services komen samen op het moment dat de upgrade en transformatie van IT-systemen direct zijn gekoppeld aan de organisationele en strategische prioriteiten van de klant. Capgemini werkt samen met strategische alliantiepartners om de issues en uitdagingen van onze klanten beter te kunnen aanpakken.

Outsourcing

Inspelend op de behoefte van de klant is Capgemini in staat de volledige of gedeeltelijke verantwoordelijkheid op zich te nemen voor het management van de IT-middelen. In ons brede aanbod van diensten staat expertise in het management van IT-systemen en bedrijfsprocessen evenals prijsflexibiliteit centraal. Met als resultaat: de verhouding tussen kosten en prestaties bij onze klanten maximaliseren. Outsourcing van bedrijfsprocessen vormt één van de kernactiviteiten van Capgemini. Dit stelt de klant in staat belangrijke ondersteuningsfuncties - zoals boekhouding of inkoop - uit te besteden. Hierin wordt onze collaboratieve benadering het meest tastbaar voor onze klanten.

Hoe Capgemini samenwerkt

Uit onderzoek dat is uitgevoerd onder (potentiële) klanten, medewerkers en analisten blijkt dat zaken als technische expertise, kennis van de branche en uitvoerend vermogen niet langer bepalende competenties zijn, maar basisvoorwaarden. Klanten gaan ervan uit dat deze competenties aanwezig zijn bij een dienstverlener en zijn daarom vooral geïnteresseerd in de manier waarop de consultant met hen gaat werken.

Voor Capgemini is samenwerkingsvermogen de belangrijkste onderscheidende factor. Klanten van Capgemini ervaren in de praktijk de samenwerking met Capgemini doorgaans als zeer intensief. Capgemini formuleert oplossingen vanuit klantperspectief, stelt zich op als co-maker en bedenkt, initiëert en implementeert hiertoe innovatieve samenwerkingsvormen. Capgemini noemt zijn manier van samenwerken Collaborative Business Experience.

Voor Capgemini is de Collaborative Business Experience een concrete belofte waarbij beide partijen zich verbinden aan een goed resultaat. In een vroeg stadium identificeert Capgemini de vaardigheden, de kennis en de ervaring die beide partijen kunnen inbrengen om een project tot een succesvol einde te brengen. Deze elementen worden gebundeld op vier doelgebieden van samenwerking:

- Focus op het creëren van waarde;
- Verlagen en beheersen van risico's;
- Optimaliseren van competenties;
- Stroomlijnen van de organisatie.

Binnen Capgemini is samenwerkingsvermogen een sleutelbegrip. De onderneming bouwt op teams van ondernemende professionals, die over de grenzen van hun eigen specialisme kunnen kijken en resultaatgerichtheid kunnen combineren met onconventioneel denken.

Samenwerking beperkt zich niet tot de grenzen van de eigen onderneming. De binnen Capgemini wereldwijd beschikbare kennis, ervaring en faciliteiten worden waar mogelijk benut voor opdrachtgevers. Capgemini completeert dit aanbod door ook de kennis en kunde van verschillende businesspartners te integreren in het werk voor klanten.

Nauwe samenwerking manifesteert zich in:

- Hoge klanttevredenheid;
- Op samenwerking gerichte methoden;
- Sterke teamspirit;
- Hoge betrokkenheid bij het succes van klanten.

Bij het realiseren van de oplossingen stuurt Capgemini samen met zijn klant aan op de effectiviteit van de oplossing. Het wederzijds vertrouwen is hierbij een belangrijk uitgangspunt, zowel voor het eindresultaat als voor het realiseren van de nodige veranderingen.

Voor meer informatie: www.nl.capgemini.com

1.2. Projectorganisatie

De projectorganisatie bestaat uit de volgende personen.

Opdrachtgever:	Armin Bauer - armin.bauer@capgemini.com Managing Consultant - Peoplemanager, lid kernteam Competence Center “Integrate!"
Bedrijfsbegeleider:	Leo van der Meulen - leo.vandermeulen@ns.nl Senior Consultant - Software engineer / Software architect
Docentbegeleider:	Eric Gerlofsma – eric.gerlofsma@hu.nl
Opdrachtnemer:	Rik Smith – rik.smith@capgemini.com Student Informatica, Hogeschool Utrecht

2. Projectdefinitie

2.1. Context

De afstudeeropdracht zal worden uitgevoerd bij Capgemini Nederland. De betreffende afdeling, F51 (Integration & Migration Services), is onderdeel van de sector “Telecom, Travel en Utilities” en is een “technology” afdeling. F51 richt zich op het (gepersonaliseerd) toegankelijk maken en onderhouden van informatie, via welk kanaal dan ook en onafhankelijk van tijd en plaats, met als doel de efficiency van de bedrijfsprocessen van de klanten meetbaar te verbeteren. De afdeling F51 kent de volgende Competence Centers:

- Integrate!
- Migration
- Analysis

Uiteindelijk val ik dan onder het Competence Center “Integrate!” dat zich sterk gericht op het implementeren van oplossingen gebaseerd op de Weblogic en Aqualogic producten van BEA Systems en de PORTLETsuite van Componence. Tevens wordt de Java-expertise binnen “Integrate!” verder ontwikkeld zodat (teams van) medewerkers ook allerlei uitdagende opdrachten op J2EE-gebied kunnen uitvoeren.

2.2. Aanleiding

Bij de klanten van Capgemini is een steeds toenemende vraag naar de implementatie van J2EE web frameworks. Op dit moment zijn er veel verschillende J2EE web frameworks beschikbaar. De grote keuze maakt het voor organisaties moeilijk om een onderbouwde keuze te maken tussen de verschillende mogelijkheden.

2.3. Probleemstelling

Door het grote aantal frameworks heeft Capgemini op dit moment niet de informatie die nodig is om een onderbouwde keuze voor een bepaald framework te maken. Hierdoor wordt er niet gekozen op inhoud maar puur op bekendheid en populariteit.

2.4. Doelstelling

De doelstelling van dit project is voor Capgemini: Inzicht krijgen in de werking van de verschillende frameworks, om zo aan klanten een onderbouwde aanbeveling te kunnen geven over het te gebruiken framework.

De doelstelling voor de student is: Kennis maken met een aantal nieuwe frameworks om zo zijn algemene kennis te verdiepen. Een volgende doelstelling is ervaring op doen met het uitvoeren van een onderzoek. Verder heeft de student de doelstelling om zijn kennis en ervaring met plannen te verbeteren.

2.5. Opdracht

De opdracht is een onderzoeksopdracht. Er moet een vergelijkend onderzoek worden gedaan naar drie J2EE web frameworks, bijv. Spring en Struts. Allereerst moet er een lijst met beschikbare frameworks worden opgesteld, deze lijst is te vinden in: “Bijlage 1 – Lijst met kandidaat frameworks”. Uit deze lijst moeten in overleg met de opdrachtgever negen kandidaat frameworks worden geselecteerd voor het onderzoek.

Om uit deze negen kandidaat frameworks drie frameworks te kiezen die onderzocht zullen worden, moeten er selectiecriteria worden opgesteld. Om deze selectiecriteria toe te kunnen passen moet er een vooronderzoek worden uitgevoerd om de benodigde gegevens boven water te krijgen. De frameworks die in overleg met de opdrachtgever voor dit vooronderzoek zijn geselecteerd zijn te vinden in: “Bijlage 2 – Lijst selectie negen frameworks”.

In het vooronderzoek wordt op basis van de selectiecriteria een aanbeveling gedaan voor de te onderzoeken frameworks. Het vooronderzoek is als apart document bij deze scriptie aangeleverd. Het selectie proces staat grafisch weergegeven in figuur 2.1.

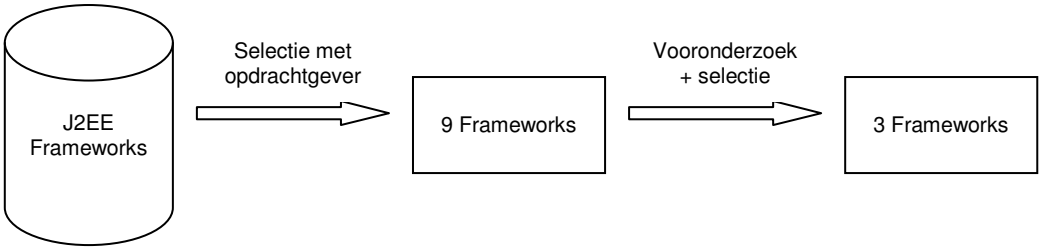


Fig 2.1 - Schematische weergave selectieproces

In overleg met de opdrachtgever zijn de volgende drie frameworks geselecteerd: Apache Beehive, Spring MVC en Struts 2. Deze selectie heeft primair plaats gevonden op basis van de aanbeveling uit het vooronderzoek. De keuze wijkt voor één framework af van de aanbeveling. De opdrachtgever doet veel met producenten van BEA, Beehive is daar ook een product van. De opdrachtgever wil daarom dit framework ook graag in de vergelijking. Daardoor is JSF afgevallen, er is binnen Capgemini namelijk veel vraag naar Spring en Struts en nauwelijks naar JSF.

2.6. Resultaten

De resultaten zullen bestaan uit de volgende op te leveren producten:

- Projectplan
- Lijst met kandidaat frameworks
- Lijst met negen frameworks voor vooronderzoek
- Lijst met drie te onderzoeken frameworks
- Onderzoeksrapport vooronderzoek selectie frameworks
- Ontwerp voor prototypes
- Prototype in elk framework
- Het onderzoeksrapport
- Scriptie

3. Projectaanpak

3.1. Activiteiten

De volgende activiteiten met de bijbehorende resultaten zijn onderkend.

Activiteit	**Resultaat**
Opstellen Projectplan	Projectplan
Oriëntatie bedrijf / Onboarding	-
Opbouwen kennis over uitvoeren onderzoek	-
Opstellen lijst met frameworks	Lijst met frameworks
Evalueren lijst met frameworks	Lijst met negen frameworks
Opstellen selectiecriteria frameworks	Selectiecriteria
Vooronderzoek uitvoeren	Vooronderzoek
Selectie maken uit lijst met negen frameworks	Drie te onderzoeken frameworks
Doelstelling prototype bepalen	Doelstelling prototype
Ontwerpen van prototype	Ontwerp prototype
Opstellen vergelijkingspunten van de drie frameworks	Vergelijkingspunten van de drie frameworks
Bouwen prototypes	Prototypes
Uitvoeren onderzoek	-
Schrijven onderzoeksrapport	Onderzoeksrapport
Schrijven Scriptie	Scriptie
Afstudeerzitting voorbereiden	Presentatie
Afstudeerzitting	-

3.2. Planning per activiteit

De planning per activiteit is te vinden in “Bijlage 3 – Planning per activiteit”.

4. Vooronderzoek

4.1. Doelstelling

Om uit de lijst met negen frameworks (zie: “Bijlage 2 – Lijst selectie negen frameworks”) een keuze te kunnen maken zijn er selectiecriteria opgesteld, deze staan vermeldt in paragraaf 4.2. De doelstelling van het vooronderzoek is: Alle voor deze selectiecriteria benodigde informatie beschikbaar maken. Ook moet er na dit vooronderzoek een globaal beeld zijn van de negen onderzochte frameworks. Dit omvat de visie, globale mogelijkheden en huidige staat. Dit vooronderzoek is als los document bij deze scriptie aangeleverd.

4.2. Selectiecriteria

De selectiecriteria zijn op te delen in drie hoofdgroepen:

- Framework ontwikkeling
- Informatie over de frameworks
- Vraag naar de frameworks

In het vooronderzoek is allereerst gekeken naar de framework ontwikkeling. Er is per framework gekeken wat de visie van de makers is. Dit is lastig omdat de meeste frameworks niet ergens hebben staan: Dit is onze visie. Dit moet dus worden afgeleidt uit de manier waarop het framework toegepast moet worden.

Vervolgens is de huidige staat van de frameworks bekeken. Bijvoorbeeld: Wat was de laatste release datum? Staat er een datum voor de volgende release gepland? Of wat is de gemiddelde release frequentie?
Als laatste is er gekeken naar de toekomst van de frameworks. Wordt er een toekomst visie genoemd? Is er een publieke plaats waar fouten in de software kenbaar kunnen worden gemaakt, zo ja: is het voor het publiek zichtbaar wat er kenbaar gemaakt is?

Het volgende onderdeel is de informatie die over de frameworks beschikbaar is. Dit is op vier manieren bekeken. Allereerst is er gekeken naar het aantal zoekresultaten op Google en naar het aantal resultaten op twee populaire websites waar over Java onderwerpen wordt gepubliceerd als er naar de frameworks wordt gezocht.

Als tweede is er op Google gekeken naar het aantal zoekresultaten als er naar tutorials wordt gezocht. Dit levert helaas niet het fysieke aantal tutorials op maar alleen het aantal zoekresultaten, dit geeft wel een indicatie voor de beschikbaarheid van tutorials.Daarnaast is er gekeken naar de mogelijkheid voor ondersteuning. Dit omvat onder andere: Zit er een ontwikkelaar achter het framework? Is er commerciële ondersteuning?

Als laatste bron van informatie in dit onderdeel is er op Amazon.com gezocht naar boeken over de frameworks.

Het laatste onderdeel van dit vooronderzoek is de vraag naar de frameworks. Hierbij zijn twee gebieden onderzocht: De vraag in de arbeidsmarkt, dit is bekeken door op verschillende vacaturesites te zoeken naar de frameworks.
Als tweede is het aanbod in de arbeidsmarkt onderzocht, dit is bekeken door te kijken hoe vaak de frameworks als competentie op de CV van Capgemini (Nederland) medewerkers voorkomen.

4.3. Conclusie

De conclusie van het vooronderzoek is dat de frameworks JSF, Spring en Struts 2 het meest geschikt zijn voor de selectie.

Dit is gebaseerd op de resultaten uit de verschillende onderdelen.
Aan deze drie frameworks wordt (volop) ontwikkeld. Door de hoeveelheid beschikbare informatie en internetactiviteit lijken deze frameworks toekomst te hebben. Over deze frameworks is veel informatie te vinden, al is dit voor Struts 2 wat minder door de nieuwigheid hiervan. Naar deze frameworks is veel vraag, alleen naar Struts 2 is door de zojuist vermelde reden geen vraag. Naar Struts 1 is echter veel vraag, het lijkt daarom logisch dat de nieuwe versie hiervan ook veel gevraagd zal worden.

5. Frameworks

In dit hoofdstuk wordt de visie van de makers van de frameworks en de technische werking van de frameworks behandeld. Hierbij wordt er vanuit gegaan dat de lezer (enige) voorkennis heeft van Java programmeren en/of J2EE[1].

5.1. Apache Beehive

5.1.1. Introductie

Het doel van Beehive is om J2EE ontwikkeling eenvoudiger te maken door een simpel object model op J2EE en Struts te bouwen. Door het gebruik van de JSR-175 annotaties[2] wordt de hoeveelheid code die geschreven moet worden sterk verminderd. Beehive bestaat uit drie delen:

- NetUI
- Control
- Web Service Metadata (WSM)

NetUI (Waar dit onderzoek zich op zal richten) is een op annotaties gebaseerd framework dat is gebaseerd op Struts (Versie 1) [3]. Alle navigatie logica, status, metadata en foutafhandelings informatie is gecentraliseerd in één herbruikbare “Page Flow Controller” class. Verder bestaat NetUI uit een JSP taglib met onder andere datagrids en boomstructuren. NetUI is ook met JavaServer Faces en Struts te integreren.

5.1.2. Architectuur

Beehive is ontworpen bovenop Struts 1. Beehive draait om een “PageFlowController”. Hierin worden met annotaties de acties gedeclareerd, het resultaat hierbij en foutafhandeling. Omdat alles met annotaties wordt gedaan is er, afgezien van de web.xml, geen configuratie file voor de gebruiker. Dat wil niet zeggen dat er geen configuratie file is, deze (Struts 1) configuratie wordt door Beehive uit de annotaties gegenereerd, dit gebeurt met een Annotation Processing Tool (APT) van Beehive. Deze tool moet bij het compileren worden gebruikt om de runtime configuratie te maken. Dit genereren kan gemakkelijk met Apache Ant[4], Beehive levert hier al een ant-script voor. De hele Beehive applicatie wordt daarmee naar een Struts 1 module gecompileerd.

Controllers

Een voorbeeld van de meest simpele controller:

```
@Jpf.Controller(
        simpleActions={
                @Jpf.SimpleAction(
                        name="begin",
                        path="index.jsp")
        }
)
public class Controller extends PageFlowController
{

}
```

Deze controller specificeerd alleen dat de pagina “index.jsp” wordt weergegeven als de actie “begin.do” wordt aangeroepen. Zoals te zien is wordt een controller gedeclareerd door de annotatie “@Jpf.Controller” te gebruiken. Bij deze annotatie kunnen door de property “simpleActions” te gebruiken acties worden gespecificeerd waarvoor het niet nodig is om een door de programmeur gemaakte methode uit te voeren.

[1] Zie voor informatie: http://nl.wikipedia.org/wiki/Java_2_Enterprise_Edition.
[2] Zie voor informatie: http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html.
[3] Zie voor informatie: http://struts.apache.org/1.3.8/index.html.
[4] Zie voor informatie: http://ant.apache.org/.

De volgende controller geeft niet alleen een pagina weer als een actie wordt aangeroepen, maar deze voert voor de actie “customers” ook code uit.

```
@Jpf.Controller(
        simpleActions={
                @Jpf.SimpleAction(
                        name="begin",
                        path="index.jsp")

        }
)
public class Controller extends PageFlowController
{
  @Jpf.Action(
        forwards = {
                @Jpf.Forward(
                        name = "success",
                        path="customers.jsp",
                        actionOutputs={
                                @Jpf.ActionOutput(
                                        name="customersInput",
                                        type=CustomerEntity[].class,
                                        required=true
                                )
                        }
                )
        }
  )
  public Forward customers() {
        Forward forward = new Forward("success");
        …
        forward.addActionOutput("customersInput", … );
        return forward;
  }
}
```

Als de actie “customers” wordt aangeroepen wordt door Beehive de methode “customers” van de “Controller” aangeroepen. De methode moet als het resultaat “success” is ook een “CustomerEntity[ ]” als output meegeven doormiddel van de functie “addActionOutput”. Deze output is in de JSPs beschikbaar onder de vermelde naam, in bovenstaand voorbeeld is dat “customersInput”. Zoals te zien, wordt een actie gedeclareerd door de annotatie “@Jpf.Action” te gebruiken. Deze annotatie bestaat weer uit één of meer forwards (“@Jpf.Forward”), dat kan je de resultaten noemen. Hierin wordt gespecificeerd wat de actie als resultaat teruggeeft en welke pagina er bij dit resultaat hoort.

Input

Als een pagina een formulier bevat wordt het object waar de gegevens in moeten worden opgeslagen niet via de “ActionOutput” meegegeven. In plaats daarvan moet dit object als tweede parameter aan de constructor van de “Forward” worden meegegeven, of door middel van de methode “addFormOutput”. Dit object kan elk willekeurig Java object zijn, zolang de properties maar getters en setters hebben, dit wordt een JavaBean[5] genoemd. Op deze manier kunnen ook gemakkelijk “Entity Beans” [6]worden meegegeven. Achter de schermen worden deze objecten door een Beehive object als Struts ActionForm gebruikt.

[5] Zie voor informatie: http://nl.wikipedia.org/wiki/JavaBeans.
[6] Zie voor informatie: http://en.wikipedia.org/wiki/EJB.

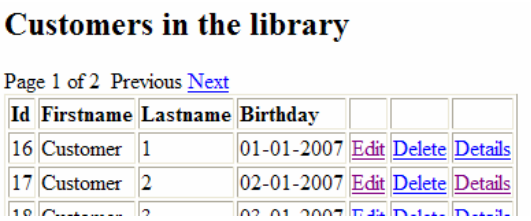
De acties die een formulier moeten opslaan moeten een parameter in de methode hebben. Die parameter zal het verstuurde formulier object bevatten.

```
@Jpf.Action(
        forwards = {
            @Jpf.Forward(
                name = "success",
                action = "customers",
                redirect=true)
        },
        validatableProperties={
                @Jpf.ValidatableProperty(
                    propertyName="firstName",
                    displayName="Field",
                    validateRequired=@Jpf.ValidateRequired()
                ),
                @Jpf.ValidatableProperty(
                     propertyName="birthDayDate",
                     displayName="Field",
                     validateRequired=@Jpf.ValidateRequired()
                )
            },
            validationErrorForward=@Jpf.Forward(name="fail",
path="customerForm.jsp")
    )

    public Forward saveCustomer(CustomerEntity form)
    {
         …
         return new Forward("success");
    }
```

Om validatie uit te voeren op het zojuist ingevulde formulier kan bij de actie die het formulier ontvangt de property “validatableProperties” worden gebruikt. Hier kan met annotaties de validatie worden geregeld. Bijvoorbeeld door zoals in bovenstaand voorbeeld gebruik te maken van de “@Jpf.ValidateRequired” om te controleren dat het veld is ingevuld. Zo staat er in bovenstaand voorbeeld gedeclareerd dat de property “firstName” verplicht is.

Taglib

Naast de controller bestaat Beehive ook uit een JSP taglibrary[7], NetUI genaamd. De library bestaat uit standaard HTML componenten, zoals een tekstveld, tabel, formulier en uit twee extra’s: een datagrid en een boomstructuur (tree). Hieronder volgt een voorbeeld van een datagrid.



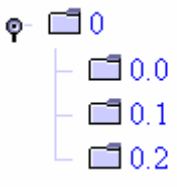
Een datagrid bestaat standaard uit een header, footer, rijen en een “pager”. De pager regelt het verdelen van de informatie over verschillende pagina’s. Het enige wat hiervoor ingesteld moet worden is welke actie er aangeroepen moet worden. Dit is dezelfde actie als waarmee de pagina opgeroepen wordt. Hierin hoeft niets gedaan te worden om ook daadwerkelijk de opgevraagde set rijen te krijgen, dit regelt Beehive. De pager is niet uit te schakelen, het is wel mogelijk om deze op een heel groot getal per pagina te zetten. Dit is iets om rekening mee te houden, als de pager niet wordt weergegeven kunnen er rijen verborgen zijn.

[7] Zie voor informatie: http://en.wikipedia.org/wiki/Tag_library#JSP_tag_libraries.

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib    uri="http://beehive.apache.org/netui/tags-databinding-1.0"    prefix="netui-
data"%>
<%@taglib     uri="http://beehive.apache.org/netui/tags-template-1.0"     prefix="netui-
template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
      <title>Customers</title>
  </head>
  <netui:body>
      <h2>Customers in the library</h2>
      <netui-data:dataGrid name="CustomersGrid"
           dataSource="pageInput.customersInput" border="1">
        <netui-data:configurePager pageAction="customers"/>
        <netui-data:header>
          <netui-data:headerCell headerText="Id" cellAlign="left"/>
          <netui-data:headerCell headerText="Firstname" cellAlign="left"/>
          <netui-data:headerCell headerText="Lastname" cellAlign="left"/>
          <netui-data:headerCell headerText="Birthday" cellAlign="left"/>
          <netui-data:headerCell headerText=" " cellAlign="left"/>
<netui-data:headerCell headerText=" " cellAlign="left"/>                <netui-
data:headerCell headerText=" " cellAlign="left"/>                <netui-
data:headerCell headerText=" " cellAlign="left"/>              </netui-data:header>
        <netui-data:rows>
          <netui-data:spanCell value="${container.item.id}" />
          <netui-data:spanCell value="${container.item.firstName}" />
          <netui-data:spanCell value="${container.item.lastName}" />
          <netui-data:spanCell value="${container.item.birthDayDate}">
            <netui:formatDate pattern="dd-MM-yyyy" />
          </netui-data:spanCell>
          <netui-data:anchorCell value="Edit" action="editCustomer">
            <netui:parameter name="id" value="${container.item.id}"/>
          </netui-data:anchorCell>
          <netui-data:anchorCell value="Delete" action="deleteCustomer">
            <netui:parameter name="id" value="${container.item.id}"/>
          </netui-data:anchorCell>
          <netui-data:anchorCell value="Details" action="customer">
            <netui:parameter name="id" value="${container.item.id}"/>
          </netui-data:anchorCell>
        </netui-data:rows>
      </netui-data:dataGrid>
      <netui:anchor action="addCustomer">Add a customer</netui:anchor>
      <br />
      <netui:anchor action="begin">Back to main</netui:anchor>
  </netui:body>
</netui:html>
```

Een datagrid wordt gedeclareerd met een “dataGrid” tag. Met de property “dataSource” wordt een object aan het datagrid gekoppeld. In de “rows” tag komt een verzameling van de velden die weergegeven moeten worden. Deze velden worden in de eerder gedefinieerde datasource opgezocht.

Naast het datagrid is er zoals gezegd een tree, zoals te zien in onderstaande figuur. Omdat de tree niet in het prototype wordt gebruikt zal er voor de tree niet op de exacte werking worden ingegaan, raadpleeg hiervoor de documentatie van Beehive. Het datagrid is wel besproken omdat deze in de prototypes is verwerkt.



5.2. Spring MVC

5.2.1. Introductie

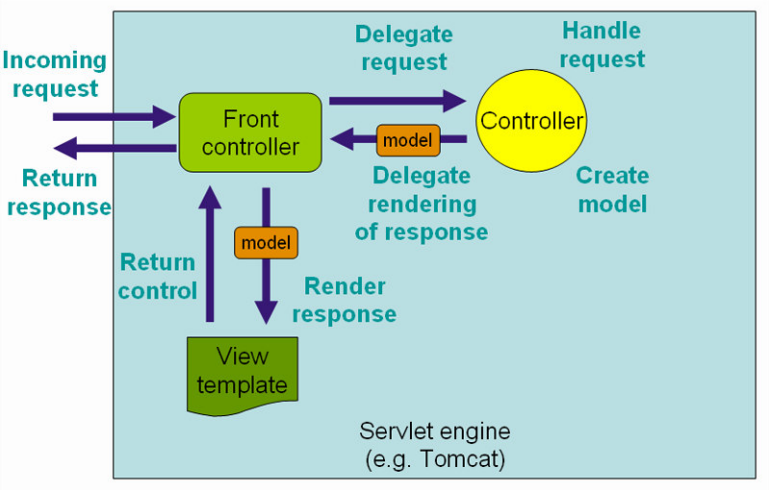
Het idee achter Spring komt oorspronkelijk uit een book, geschreven door Rod Johnson, “Expert One-on-One”.
Het kern idee van Spring is: “Inversion of Control” (IOC). Dit komt neer op het uit handen geven van verantwoordelijkheden aan Spring. Een voorbeeld hiervan is: “Dependency Injection”. Dit houdt in dat de relaties tussen beans niet in de code opgezocht hoeven te worden maar dat Spring dit doet.

Spring MVC werkt zoals de naam al doet vermoeden volgens het MVC model[8]. Spring levert een “DispatcherServlet” met configureerbare mappings. Deze “DispatcherServlet” stuurt de controller aan. De meest simpele controller heeft een “ModelAndView handleRequest(request,response)” methode die wordt aangeroepen als de Controller aangeroepen wordt. Deze geeft als resultaat een ModelAndView, dit is een verzameling van gegevens uit het model (Bijvoorbeeld een object voor een formulier, of gegevens voor in een tabel) en de View waar die informatie naar toe moet.

Er kan verder ook gekozen worden voor geavanceerdere controllers zoals een “SimpleFormController”. Deze controller is handig als er met een formulier gewerkt wordt, met deze controller kan elke JavaBean gebruikt worden om de formuliergegevens in op te slaan.

5.2.2. Architectuur

Spring MVC is ontworpen rond de “DispatcherServlet”. De “DispatcherServlet” geeft de binnenkomende verzoeken door aan de juiste controller volgens configureerbare mappings. Deze configuratie wordt gedaan in de “WebApplicationContext”. Hierin worden de controllers, beans en view-resolvers geregisteerd. De afhandeling van een request ziet er als volgt uit:



Een request komt binnen bij de Servlet engine en die roept de “DispatcherServlet” van Spring aan (dit heet in het plaatje de “Front controller”). Dit wordt ingesteld in de web.xml.

[8] Zie voor informatie: http://en.wikipedia.org/wiki/Model-view-controller.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
 version="2.4"
>
  <display-name>PrototypeSpring</display-name>
  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>

  <!-- The Usual Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

De “DispatcherServlet” roept afhankelijk van de mappings in de “WebApplicationContext” een controller aan. Standaard zoekt de DispatcherServlet naar een bestand met de naam “*dispatcherservletnaam*-servlet.xml” in de “WEB-INF” map. Volgens bovenstaande “web.xml” zou dit het bestand “springapp-servlet.xml” zijn. De controller stelt het model samen en geeft aan welke view er gebruikt moet worden. Vervolgens wordt dit in een ModelAndView aan de “DispatcherServlet” gegeven. Deze roept dan afhankelijk van het resultaat de betreffende View aan en geeft het model mee. De view maakt de response aan en geeft deze weer aan de “DispatcherServlet”. Die stuurt vervolgens de response door naar de cliënt.

Configuratie
De configuratie staat zoals gezegd in de “WebApplicationContext”.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<!--  - Application context definition for "springapp" DispatcherServlet.  -->
<beans>
  <bean id="CustomerController" class="controllers.CustomerController">
  </bean>
  <bean id="urlMapping" class="org.springframework.web.servlet-
  .handler.SimpleUrlHandlerMapping">
    <property name="mappings">
      <props>
        <prop key="/customer.action">CustomerController</prop>
      </props>
    </property>
  </bean>
</beans>
```

Alles in de configuratie wordt gedeclareerd als “bean”. Hierbij moet er een klasse en id voor de bean worden opgegeven. Daarnaast kunnen er nog properties worden opgegeven voor de bean. In dit geval wordt er bijvoorbeeld aan de bean “urlMapping” een lijstje met mappings meegegeven. Aan de hand hiervan wordt bepaald dat als “customer.action” wordt aangeroepen de controller “CustomerController” moet worden gebruikt.

Controllers
De meest simpele controller ziet er als volgt uit:

```
public class CustomerController implements Controller {

  public ModelAndView handleRequest(HttpServletRequest request,
                HttpServletResponse response) throws Exception {
        return new ModelAndView("viewname", "modelname", modelvalue);
  }
}
```

Zoals te zien bestaat de controller uit niets meer dan een klasse die de interface "Controller" implementeerd en daarmee ook de functie "handleRequest" implementeerd die een "ModelAndView" teruggeeft. De "ModelAndView" bestaat uit een view (of naam van een view) en een model. Het model is een verzameling van objecten met voor elk object een naam als key.

De viewnaam die door de controller wordt teruggegeven wordt door een "ViewResolver" omgezet naar de juiste view technologie. Standaard ondersteunt Spring de JSP, Velocity[9] en XSLT[10] technologieën. Een voorbeeld van een "ViewResolver" is de "InternalResourceViewResolver".

```
<bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass">
        <value>org.springframework.web.servlet.view.JstlView</value>
  </property>
  <property name="prefix"><value>/WEB-INF/jsp/</value></property>
  <property name="suffix"><value>.jsp</value></property>
</bean>
```

Als deze "ViewResolver" de view "customer" binnen krijgt zal deze de view "/WEB-INF/jsp/customer.jsp" als "JstlView" aan de "DispatcherServlet" geven. Die stuurt deze dan op zijn beurt als response.

Er zijn behalve de interface "Controller" nog meer controller interfaces, als er bijvoorbeeld een formulier nodig is kan er een "SimpleFormController" worden gebruikt. Hiermee is het mogelijk om een object in te stellen om de formuliergegevens in op te slaan en het is hiermee mogelijk om de submit af te handelen. Onderstaand een voorbeeld van een "CancellableFormController". Een "CancellableFormController" heeft bovenop de functionaliteiten van een "SimpleFormController" de mogelijkheid om het formulier te annuleren.

```
public class CustomerFormController extends CancellableFormController {

  protected Object formBackingObject(HttpServletRequest arg0) throws Exception {
    //return a the object to be used a formbackend
    …
  }

  protected ModelAndView onSubmit(Object command) throws ServletException{
    CustomerEntity ce = (CustomerEntity) command;
    // delegate the update to the business layer
    …
    return new ModelAndView(getSuccessView());
  }
}
```

[9] Zie voor informatie: http://velocity.apache.org/.
[10] Zie voor informatie: http://www.w3.org/TR/xslt.

En de bijbehorende configuratie:

```
<bean id="CustomerFormController" class="controllers.CustomerFormController">
  <property name="formView" value="customerForm" />
  <property name="commandName" value="customer" />
  <property name="successView" value="redirect:customers.action"/>
  <property name="cancelView" value="redirect:customers.action"/>
  <property name="validator">
    <bean class="validators.CustomerValidator" />
  </property>
</bean>
```

De property “formView” regelt de pagina waarin het formulier wordt weergegeven. Dat is in dit geval “customerForm.jsp”. Als dat formulier verstuurd wordt en de validatie is succesvol uitgevoerd wordt de methode “onSubmit” aangeroepen.

In de configuratie wordt er aangegeven wat de naam van het formulier object is, de success-pagina en de annuleer-pagina. Daarnaast is er zoals te zien een validator ingesteld. Een validator is een Java klasse die de “Validator” interface van Spring implementeerd.

```
public class CustomerValidator implements Validator {


  /**
  * This Validator validates just CustomerEntity instances
  */
  public boolean supports(Class clazz) {
    return CustomerEntity.class.equals(clazz);
  }

  public void validate(Object obj, Errors e) {
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "firstName", "field.required");
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "lastName",  "field.required");
  }
}
```

De validator bestaat uit twee methodes. De eerste is een methode om te verifieren dat het object door deze validator gevalideerd kan worden. De tweede is de methode die de daadwerkelijke validatie van het object uitvoert. Er zijn twee manieren waarop de validatie uitgevoerd kan worden. Er kan zoals in het voorbeeld gebruik gemaakt worden van de “ValidationUtils” om een bepaalde property te controleren, maar er kan ook handmatig gecontroleerd worden door het binnenkomende “Object” te gebruiken. Dit object is het met de waardes uit het formulier gevulde object dat als formulier backend gebruikt wordt.

Een controller kan ook voor meer dan één actie gebruikt worden. Om dit te bewerkstelligen moet er gebruik worden gemaakt van een “MultiActionController”. Een “MultiActionController” is op verschillende manieren te configureren. Er kan voor gekozen worden om in de Spring configuratie in te stellen welke URL bij welke methode hoort of er kan zoals in onderstaand voorbeeld een “ParameterMethodNameResolver” worden gebruikt. Hiermee kan worden ingesteld dat via een request parameter wordt aangegeven welke methode er uitgevoerd moet worden.

```
<bean id="methodNameResolver"
  class="org.springframework.web.servlet.mvc.multiaction.Parameter-
  MethodNameResolver">
  <property name="paramName">
    <value>action</value>
  </property>
</bean>

<bean id="CustomerController" class="controllers.CustomerController">
  <property name="methodNameResolver" ref="methodNameResolver"/>
</bean>
```

In bovenstaande Resolver wordt de request parameter “action” gebruikt om in de “CustomerController” de uit te voeren methode te bepalen. De waarde van de “action” parameter is de naam van de uit te voeren methode.

Een derde mogelijkheid is om een “InternalPathMethodNameResolver” te gebruiken. Deze neemt de bestandnaam die wordt aangeroepen als methodenaam. Als er bijvoorbeeld “test.action” wordt aangeroepen resulteert dit in de methode “test”. Er moet wel worden ingesteld welke controller er bij bijvoorbeeld “test.action” hoort.

Taglibs
Bij Spring worden twee taglibraries meegeleverd. Eén library is een algemene, vooral bedoelt voor databinding met Spring en de tweede is een library voor formulieren. In onderstaande JSP staat een voorbeeld van de formulier library:

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
  <head>
    <title>Add new customer</title>
  </head>
  <body>
    <h2>Add new customer</h2>
    <form:form method="post" action="customerForm.action"
      commandName="customer">
      <table>
        <tr>
          <th align="left">Firstname:</th>
          <td>
            <form:input path="firstName" size="30" />
          </td>
          <td>
            <font color="red"><form:errors path="firstName" /></font>
          </td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" value="Submit"></td>
        </tr>
        <form:hidden path="id" />
      </form:form>
    </body>
</html>
```

De eerste tag is de “form” tag. Hiermee kan een formulier aan een bijbehorende “command” in Spring gekoppeld worden. Alle velden die vervolgens in het formulier met Spring tags declareerd worden, gebruiken deze “command”. Door de “input” tag te gebruiken verschijnt er een HTML “input type=text” veld. Deze is aan het opgegeven veld in de Spring “command” gekoppeld. De naam van de Spring tag komt overeen met de HTML tag, zo levert de Spring “password” tag bijvoorbeeld een HTML “password” veld op. Met de “errors” tag kunnen de formulierfouten uitgelezen worden, dit kan per veld of alle fouten. In totaal zijn er 12 tags in de library.

Dependency Injection
Het kernprincipe van Spring is: “Dependency Injection”. Dit houdt in dat de relaties die een object met andere (Spring) objecten heeft door Spring voor hem worden opgezocht in plaats van door het object zelf.

Een voorbeeld van een controller zonder “Dependency Injection” die een relatie met een “Session Bean” heeft:

```
public class CustomerController implements Controller {

  private CustomerSession cs;

  public CustomerController() {

          //In de Constructor de benodigde relatie op zoeken door een
          //JNDI-lookup te doen.
          String factory = "org.jnp.interfaces.NamingContextFactory";
          String url = "jnp://localhost:1099";
          Properties props = new Properties();
          props.setProperty(Context.INITIAL_CONTEXT_FACTORY, factory);
          props.setProperty(Context.PROVIDER_URL, url);

          InitialContext ctx = new InitialContext(props);
          Object ref = ctx.lookup("CustomerSessionBean/remote");
          this.cs = (CustomerSession)
          javax.rmi.PortableRemoteObject.narrow(ref,CustomerSession.class);
  }

  public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response)throws Exception {
          ……
  }
}
```

Vervolgens een voorbeeld van dezelfde controller maar dan met “Dependency Injection”:

```
public class CustomerController implements Controller {

  private CustomerSession cs;

  public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response)throws Exception {
          ……
  }

  public CustomerSession getCs() {
          return cs;
  }
  public void setCs(CustomerSession cs) {
          this.cs = cs;
  }
}
```

Door de volgende configuratie doet Spring de “Dependency Injection”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE          beans          PUBLIC          "-//SPRING//DTD          BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <bean id="CustomerController" class="CustomerController">
    <property
            name="cs"
            ref="CustomerSession" />
    </bean>

    <bean
        id="CustomerSession"
        class="org.springframework.jndi.JndiObjectFactoryBean">
        <property
            name="jndiName"
            value="CustomerSessionBean/remote" />
    </bean>
</beans>
```

Er zijn twee mogelijkheden waarop Spring “Dependency Injection” kan uitvoeren. De eerste mogelijkheid is: De dependencies via parameters aan een constructor mee geven. De tweede mogelijkheid is: De dependencies via setter methoden in stellen. Door gebruik te maken van de tag “property” met een naam en een referentie (ref) naar de dependency zal Spring een “set*Name*(*ref*)” uitvoeren. In het voorgaande voorbeeld is gebruik gemaakt van de setter werkwijze. Er is niet aan te geven welke van de twee methodes er beter is, dit is een keuze die een ieder zelf kan maken.

Zoals te zien in de voorgaande Spring configuratie kan een EJB als Spring bean gedeclareerd worden. Dit kan door een “JndiObjectFactoryBean” te gebruiken. Deze bean doet in feite hetzelfde als een handmatige Jndi-lookup[11] alleen doet Spring het. De gedeclareerde Spring bean kan ook worden geinjecteerd in een andere Spring bean. Daarmee is het dus mogelijk om één keer een EJB te declareren en deze waar nodig in een bean te injecteren, dit scheelt dubbele code en configuratie.

Een groot voordeel van de “Dependency Injection” werkwijze is ook dat de configuratie losgekoppeld wordt van de applicatie en op een centrale plaats geregeld wordt. Een ander voordeel is er bij het unit testen van de controller. Bij het unit testen is het niet wenselijk om met de echte backend te praten. Ten eerste kan dit onvoorspelbare resultaten opleveren, omdat bijvoorbeeld de informatie in de database kan veranderen. Ten tweede moet het zeker zijn dat het mislukken van een test door een fout in de controller wordt veroorzaakt, en niet door een fout in de backend. De backend zou namelijk met een aparte test moeten worden getest. Omdat de configuratie los staat van de controller kan bij het testen gemakkelijk met een surrogaat object in plaats van het echte object gewerkt worden. Dit surrogaat object ziet er hetzelfde uit als het echte object alleen geeft deze een vaste set door de tester bepaalde gegevens terug. Daardoor is het zeker dat er met de juiste waardes gewerkt wordt.

[11] Zie voor informatie: http://en.wikipedia.org/wiki/JNDI.

5.3. Struts 2

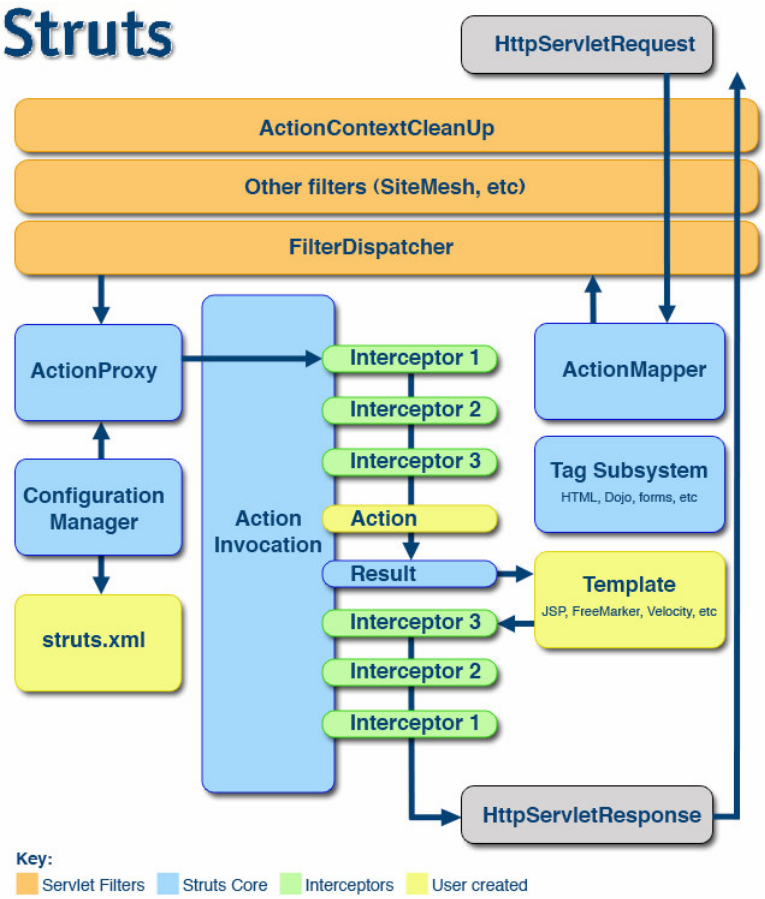
5.3.1. Introductie

Het Struts 2 framework werkt volgens het MVC model. De controller wordt door Struts geleverd en integreert met andere technologie om het Model en de View te leveren.

De controller fungeert als een brug tussen het model en de webview. Waneer de Struts controller een request ontvangt roept deze een Action aan. Welke Action er aangeroepen moet worden kan in de struts.xml worden ingesteld. De Action praat met het model, en fungeert als een soort verkeersregelaar tussen de View en het Model. Formulier gegevens kunnen rechtstreeks in een Bean worden opgeslagen of in een Action.

5.3.2. Architectuur

Het volgende plaatje geeft de afhandeling van een inkomend verzoek weer in Struts 2.



In het diagram komt het verzoek binnen bij de Servlet container[12] (Bijvoorbeeld JBoss[13]) binnen en wordt aan Struts gegeven als “HttpServletRequest”. Dit “HttpServletRequest” wordt doorgegeven aan een reeks standaard filters. Dit wordt ingesteld in de web.xml.

[12] Zie voor informatie: http://en.wikipedia.org/wiki/Servlet_container.
[13] Zie voor informatie: http://labs.jboss.com/jbossas/.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
 version="2.4"
>
<display-name>PrototypeStruts</display-name>
  <!-- use this filter if you use the struts.xml configuration -->
  <filter>
          <filter-name>struts2</filter-name>
          <filter-class>
                  org.apache.struts2.dispatcher.FilterDispatcher
          </filter-class>
  </filter>

  <!-- use this filter if you use the zero configuration feature -->
  <filter>
          <filter-name>struts2</filter-name>
          <filter-class>
                  org.apache.struts2.dispatcher.FilterDispatcher
          </filter-class>
          <init-param>
                  <param-name>actionPackages</param-name>
                  <param-value>actions</param-value>
          </init-param>
  </filter>

  <filter-mapping>
          <filter-name>struts2</filter-name>
          <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!-- The Usual Welcome File List -->
  <welcome-file-list>
          <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <!-- use this listener to load the Spring configuration -->
  <listener>
          <listener-class>
                  org.springframework.web.context.ContextLoaderListener
          </listener-class>
  </listener>
</web-app>
```

De reeks bestaat altijd uit het “FilterDispatcher” filter, als deze wordt aangeroepen consulteert deze de “ActionMapper” om te bepalen of er voor dit verzoek een actie moet worden aangeroepen. Als er een actie moet worden aangeroepen, wordt het verzoek doorgegeven aan de “ActionProxy”. De “ActionProxy” raadpleegt de “Configuration Manager” (die vanuit de door de gebruiker gemaakte struts.xml wordt geïnitialiseerd). Met de opgevraagde informatie creëert de “ActionProxy” een “ActionInvocation” die verantwoordelijk is voor het “command pattern”[14]. Dit omvat het uitvoeren van de ingestelde “interceptors” voordat de “Action” zelf wordt uitgevoerd. Zodra de “Action” een resultaat teruggeeft is de “ActionInvocation” verantwoordelijk voor het opzoeken van het resultaat in de “struts.xml” dat hoort bij het resultaat verkregen van de “Action”. Het resultaat wordt dan uitgevoerd wat vaak het vertalen van een template in JSP of FreeMarker[15] inhoudt naar het weer te geven resultaat. Tijdens dat vertalen kunnen de templates gebruik maken van de Struts tags om bijvoorbeeld met de “Action” te communiceren.

Nadat het resultaat vertaald is zorgt de “ActionInvocation” ervoor dat de interceptors weer worden uitgevoerd, maar nu in omgekeerde volgorde. Uiteindelijk wordt dan de response door de filters die geconfigureerd zijn in de “web.xml” teruggegeven.

[14] Zie voor informatie: http://en.wikipedia.org/wiki/Command_pattern.
[15] Zie voor informatie: http://freemarker.sourceforge.net/.

Configuratie
De configuratie gebeurt in de"struts.xml":

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <!-- Include Struts 2 default (from Struts 2 JAR). -->
  <include file="struts-default.xml"/>

  <!-- Configuration for the default package. -->
    <package name="StrutsSample" extends="struts-default">

  <!--   Interceptor   to   automatically   try   to   set   the   request   parameters   in
  the Action class -->
       <default-interceptor-ref name="defaultStack"/>

       <!-- This Action calls the execute method of the class -->
       <action name="customer" class="actions.CustomerAction">
            <result name="input">/customerForm.jsp</result>
            <result name="error">error.jsp</result>
            <result name="success" type="redirect-action">customers</result>
        </action>

       <!-- Action which calls a specific method in the class -->
       <action name="customers" class="actions.CustomerAction" method="list">
            <result>/customers.jsp</result>
        </action>
    </package>
</struts>
```

Door de door Struts geleverde "struts-default.xml" te "includen" in de "struts.xml" is er al een basis configuratie die aangepast (door het te overschrijven in de "struts.xml") en hergebruikt kan worden. Hierin kunnen de acties worden gedefinieerd met de resulaten die ze opleveren. Ook kan hierin de interceptor volgorde gedefinieerd worden.

Een nieuwe (experimentele) mogelijkheid in Struts is: "Zero Configuration". Hiermee is het mogelijk om zonder configuratie in XML- en property-bestanden te werken. De naam is enigszins verwarrend, hieruit zou afgeleid kunnen worden dat er helemaal geen configuratie meer is, maar er moeten nog wel annotaties worden gebruikt om het resultaat aan te geven. Om dit mogelijk te maken moet aan het Struts filter een extra parameter "actionPackages" mee te geven. Struts scant dan vervolgens de opgegeven packages voor klassen die een uitbreiding zijn op "Action" of waarvan de naam eindigt op "Action". Door de annotaties "Result" of "Results" te gebruiken kan worden aangegeven wat de resultaten van de actie zijn.

Als er wordt opgegeven dat de acties in het package "a.actions" zitten en een "Action" zit in een package "a.actions.b" geldt de packagenaam "b" als namespace tenzij anders aangegeven in de klasse. De URL zou dus bijvoorbeeld worden: http://host/applicatie/b/actienaam.action.

Actions
De meest simpele "Action" ziet er als volgt uit:

```
public class CustomerAction {
  public String execute() {
          return "success";
  }
}
```

Zoals te zien is bestaat deze "Action" uit niet meer dan een standaard Java klasse met een methode "execute()" die als resultaat een "String" terug geeft. De methode "execute()" is de methode die standaard wordt aangeroepen als er geen methode is ingesteld voor de actie. Vanuit een JSP kan elke property in een "Action" die een getter methode heeft worden gebruikt.

Een “Action” kan ook meerdere methodes hebben die uitgevoerd kunnen worden. De bijbehorende Java klasse staat hieronder. Dit werkt niet met de “Zero Configuration” mogelijkheid, er wordt dan alleen de “execute()” methode uitgevoerd.

```
public class CustomerAction {

  private CustomerEntity customer;
  private CustomerSession customerSession;

  public String execute() {
          return “success”;
  }

  public String edit() {
          ……
          return “input”;
  }

  public String add() {
          ……
          return “input”;
  }

  public String delete() {
          ……
          return “success”;
  }

  public String detail() {
          ……
          return “detail”;
  }

  public String save() {
          //save         the         customer         that         is         submitted
          ……
          return “success”;
  }

  public CustomerEntity getCustomer() {
          return customer;
  }

  public void setCustomer(CustomerEntity customer) {
          this.customer = customer;
  }
}
```

Het gebruik van meerdere methodes in een “Action” kan op verschillende manieren. De eerste manier is door in de “struts.xml” voor elke methode een actie aan te maken. Een voorbeeld staat hieronder.

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <!-- Include Struts 2 default (from Struts 2 JAR). -->
  <include file="struts-default.xml"/>

  <!-- Configuration for the default package. -->
    <package name="StrutsSample" extends="struts-default">

       <action name="customer" class="CustomerAction" method="list">
         <result name="detail">/customer.jsp</result>
       </action>

       <action name="deleteCustomer" class="CustomerAction" method="delete">
         <result name="success" type="redirect-action">customers</result>
       </action>

       <action name="editCustomer" class="CustomerAction" method="edit">
         <result name="input">/customerForm.jsp</result>
       </action>

       <action name="addCustomer" class="CustomerAction" method="delete">
         <result name="input">/customerForm.jsp</result>
       </action>
    </package>
</struts>
```

De tweede manier is om gebruik te maken van “wildcard mappings”. Hiermee kan door een actienaam conventie te gebruiken de binnenkomende actienaam worden gemapt op een methode. Een voorbeeld staat hieronder.

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <!-- Include Struts 2 default (from Struts 2 JAR). -->
  <include file="struts-default.xml"/>

  <!-- Configuration for the default package. -->
  <package name="StrutsSample" extends="struts-default">
    <action name="*Customer" class="CustomerAction" method="{1}">
      <result name="detail">/customer.jsp</result>
      <result name="success" type="redirect-action">customers</result>
      <result name="input">/customerForm.jsp</result>
    </action>
  </package>
</struts>
```

Door de actie “addCustomer” aan te roepen wordt het verzoek met de actie “*Customer” gemapt. Het deel van de naam wat met de “*” overeenkomt wordt gebruikt om de methodenaam te bepalen. Dit komt in dit geval dus overeen met “add”.

De derde methode is om “DynamicMethodInvocation” te gebruiken. Dit werkt op ongeveer dezelfde manier als “wildcard mappings” maar hiervoor hoeft er in de “Action” declaratie geen “*” gebruikt te worden. Wel moet in de configuratie “DynamicMethodInvocation” aan worden gezet. Als dit is gedaan kan een methode worden aanroepen door de syntax “actienaam!methodenaam.action” te gebruiken. In het voorbeeld kan dit dus “customer!add.action” zijn.

Input
Als een property in een “Action” een setter methode heeft kan de property vanuit de JSP worden gevult. Een voorbeeld hiervan is een formulier dat een “CustomerEntity” object moet vullen. Voor dit voorbeeld wordt dezelfde klasse als in het vorige voorbeeld en “DynamicMethodInvocation” configuratie gebruikt. Als er een nieuw “CustomerEntity” object moet worden aangemaakt kan dit in de methode “add()” gedaan worden en vervolgens kan dit nieuwe object worden toegewezen aan de property “customer”.

In de JSP kan de property “customer” weer worden uitgelezen en de properties uit de “CustomerEntity” kunnen worden gekoppeld aan invoervelden. Hoe dit in zijn werk gaat zal later worden beschreven in deze paragraaf. Als het formulier vervolgens verstuurd wordt naar “customer!save.action” wordt het gevulde “CustomerEntity” object in de “CustomerAction” geplaatst door middel van de “setCustomer(CustomerEntity customer)” methode. Vervolgens is in de methode “save()” het verstuurde “CustomerEntity” object beschikbaar en kan de verdere verwerking van dit object worden gedaan.

Voor een formulier kan dus elke willekeurige JavaBean als backend worden gebruikt. Het is ook mogelijk om de waardes uit het formulier rechtstreeks in een “Action” op te slaan.

Validatie van formulieren in Struts gebeurt met xml-bestanden. De naam van deze bestanden moet zijn: “Klassenaam-actienaam-validation.xml” dus bijvoorbeeld “CustomerAction-customer-validation.xml”. Struts gaat dan elke keer als de actie “customer” wordt aangeroepen validatie uitvoeren. Dit is met “DynamicMethodInvocation” niet altijd gewenst. Om te voorkomen dat er wordt gevalideerd kan de annotatie “@SkipValidation” bij een methode worden geplaatst. Een voorbeeld van een validatiebestand voor de actie “customer”:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="customer.firstName">
     <field-validator type="requiredstring">
        <message key="requiredstring" />
      </field-validator>
  </field>
  <field name="customer.lastName">
     <field-validator type="requiredstring">
        <message key="requiredstring" />
      </field-validator>
  </field>
  <field name="customer.birthDayDate">
    <field-validator type="required">
      <message key="requiredstring" />
    </field-validator>
  </field>
</validators>
```

Er wordt al een aantal standaard validators meegeleverd. Zo is er zoals in bovenstaand voorbeeld de “requiredstring” validator. Deze validator controleert niet alleen of er iets is ingevuld, maar ook of dit niet alleen spaties zijn. Er is verder ook een emailadres validator, die controleert of het een juist emailadres formaat is. Er zijn ook validators die een bereik controleren, deze zijn er voor onder andere integers en datums.

Niet alleen waardes uit een formulier maar ook request parameters kunnen in een “Action” opgeslagen worden. Als gebruik wordt gemaakt van de “Parameters Interceptor” probeert Struts via de beschikbare setter methodes de request parameters in de “Action” op te slaan. Als de parameter “id” wordt meegegeven zal Struts zoeken naar een methode “setId(...)” en probeert deze dan aan te roepen met de waarde van de parameter “id”.

Interceptors
Een handige interceptor tijdens het bouwen kan de “DebuggingInterceptor” zijn. Deze “DebuggingInterceptor” kan xml files genereren waarin alle beschikbare properties met waarde in vermeld staan tijdens een request.

Er is voor Struts 2 ook een “ProfilingInterceptor”. Met deze interceptor is te zien hoe lang het uitvoeren van de acties duurt. Omdat in te schakelen moet tevens de property “DevMode” van Struts 2 op true worden.

Taglib
Bij Struts 2 wordt een standaard taglib geleverd. Deze bestaat uit een grote verscheidenheid aan tags. Deze zijn te verdelen in twee groepen, algemene tags en UI Tags. De algemene tags bestaan uit onder andere de “if”, “else” en “elseif” tags. Ook bestaan deze bijvoorbeeld uit de “url” tag. Hiermee kan Struts een URL genereren voor bijvoorbeeld een actie die met een “a” tag moet worden aangeroepen.

Een voorbeeld:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
        <s:head theme="ajax"/>
                <title>Add new customer</title>
        </head>
        <body>
                <h2>Add new customer</h2>
                <s:url id="url" action="customers"  />
                <a href="<s:property value="#url"/>">Back to customers</a>
        </body>
</html>
```

Daarnaast zijn er zoals genoemd de “UI” tags. Deze zijn weer te verdelen in formulier tags en niet formulier tags. De formulier-tags bestaan onder andere uit de “textfield”, “password” en “hidden” tags.

Een voorbeeld:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
  <s:head theme="ajax"/>
                <title>Add new customer</title>
        </head>
        <body>
                <h2>Add new customer</h2>
                <s:form action="customer!save" method="post">
                        <s:textfield key="customer.firstName"
                                label="Firstname" size="40" required="true"/>
                        <s:textfield key="customer.firstName"
                                label="Lastname" size="40"
                                required="true"/>
                        <s:date id="bdate"name="customer.birthDayDate"
                        format="dd-MM-yyyy" />
                        <s:datetimepicker name="customer.birthDayDate"
                                value="%{#bdate}" displayFormat="dd-MM-yyyy"
                                label="Birthday" required="true"/>
                        <s:hidden key=" customer.Id"/>
                        <s:submit value="Submit"/>
                        <s:submit value="Cancel"
                                name="redirect-action:customers"/>
                </s:form>
        </body>
</html>
```

Door aan de “s:textfield” property “key” de waarde “customer.firstName” mee te geven vult Struts het textfield met de waarde van de property “firstName” van de property “customer” uit het actie object (als deze bestaat). Ook zal Struts als de property “label” niet is ingevuld op de waarde van “key” in de “ResourceBundle” een waarde voor de tekst van het label zoeken.

In de formulier tags zitten ook Dojo[16] widgets, dit zijn javascript hulpmiddelen, verwerkt die gebruikt kunnen worden. Een voorbeeld hiervan is de “datetimepicker”. Dit is een javascript kalender. Om deze te gebruiken moet ook de theme “ajax” worden ingesteld in de “head”. Een theme zorgt er onder andere voor dat de benodigde javascript bestanden worden geladen. Een theme kan ook zorgen voor css bestanden.

Spring Integratie
Struts biedt ook de mogelijkheid om met Spring te integreren. De Struts objecten worden dan niet meer door de standaard Struts ObjectFactory gemaakt maar door Spring. Dit kan onder andere een voordeel bieden bij het beheren van de backend. Als gebruik wordt gemaakt van EJBs als backend is het mogelijk om deze in Spring te declareren. Vervolgens kan Spring deze dan automatisch injecteren in de properties van een Struts “Action”.

Om deze integratie in te schakelen moeten er drie stappen worden doorlopen. Stap één is het kopiëren van de Spring plugin jar naar de applicatie directory. Stap twee is in de “web.xml” een Spring “ContextLoaderListener” te registreren.

Dit kan zoals hieronder staat weergegeven:

```
<listener>
          <listener-class>
                org.springframework.web.context.ContextLoaderListener
          </listener-class>
</listener>
```

De laatste stap is de Spring Configuratie, dit kan met de “applicationContext.xml”. In deze file moeten alle Spring objecten gedefinieerd worden. Zie hieronder een voorbeeld:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE          beans          PUBLIC          "-//SPRING//DTD          BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean
        id="bookSession"
        class="org.springframework.jndi.JndiObjectFactoryBean">
        <property
            name="jndiName"
            value="BookSessionBean/remote" />
    </bean>
    <bean
        id="customerSession"
        class="org.springframework.jndi.JndiObjectFactoryBean">
        <property
            name="jndiName"
            value="CustomerSessionBean/remote" />
    </bean>
</beans>
```

Als geen verdere instellingen worden gedaan zal Spring automatisch de properties en de beans op naam met elkaar matchen. Met deze configuratie zal Spring dus de property “customerSession” uit de actie met een referentie naar een instantie naar de bean “customerSession” vullen. Het is ook mogelijk om dit in te stellen op type, dan zal Spring het matchen op basis van het property-type en de bean-klasse doen.

[16] Zie voor informatie: http://dojotoolkit.org/.

Struts 1 vs. Struts 2
In versie 1 moet elke actie een abstracte basisklasse uitbreiden. In versie 2 is dit optioneel. Er zijn basis klassen die extra functionaliteit bieden maar elke klasse met een “execute()” methode kan als “Action” worden gebruikt. Versie 1 was ook afhankelijk van de Servlet API. Dit is in versie 2 niet meer het geval. Als het nodig is, kan nog wel bijvoorbeeld het HttpServletRequest of -Response object worden geraadpleegd, maar dit is niet standaard.

Om input op te slaan was er in versie 1 een “ActionForm” nodig. Dit is in versie 2 niet meer het geval. De input kan op verschillende manieren worden opgeslagen: Direct in het “Action” object of bijvoorbeeld direct in een business object.

Om in de View objecten te gebruiken uit de actie hoeven die objecten niet zoals in Struts 1 aan de view te worden meegegeven. Deze kunnen door de view met getter methodes uit de actie gelezen worden. Dit koppelt de view losser aan de “Action”.

6. Prototype

In dit hoofdstuk zal het gemaakte prototype worden behandeld. Allereerst wordt de doelstelling gegeven en vervolgens het algemene ontwerp. Als laatste volgt per framework een beschrijving hoe het prototype is geïmplementeerd. Hierin zal het proces voor het afhandelen van een pagina verzoek worden beschreven aan de hand van de use cases “List books” en “Add book”. Alle code, configuratie en jsp pagina’s zijn per framework in de bijlagen te vinden. Voor de overzichtelijkheid zijn de “import statements” uit de code gelaten.

6.1. Doelstelling

Het prototype moet de punten waarop de frameworks vergeleken worden meetbaar maken. Hiervoor moet het alle aspecten van een webapplicatie raken. Het moet werken volgens het Model View Controller (MVC)[17] model en als backend EJB’s gebruiken.

Aan de hand van dit prototype moet de hoeveelheid code en configuratie meetbaar zijn. Ook moet hiermee de flexibiliteit / aanpasbaarheid getoetst worden. Voor de GUI mogelijkheden en perfomance zou dit prototype ook gebruikt kunnen worden.

6.2. Ontwerp

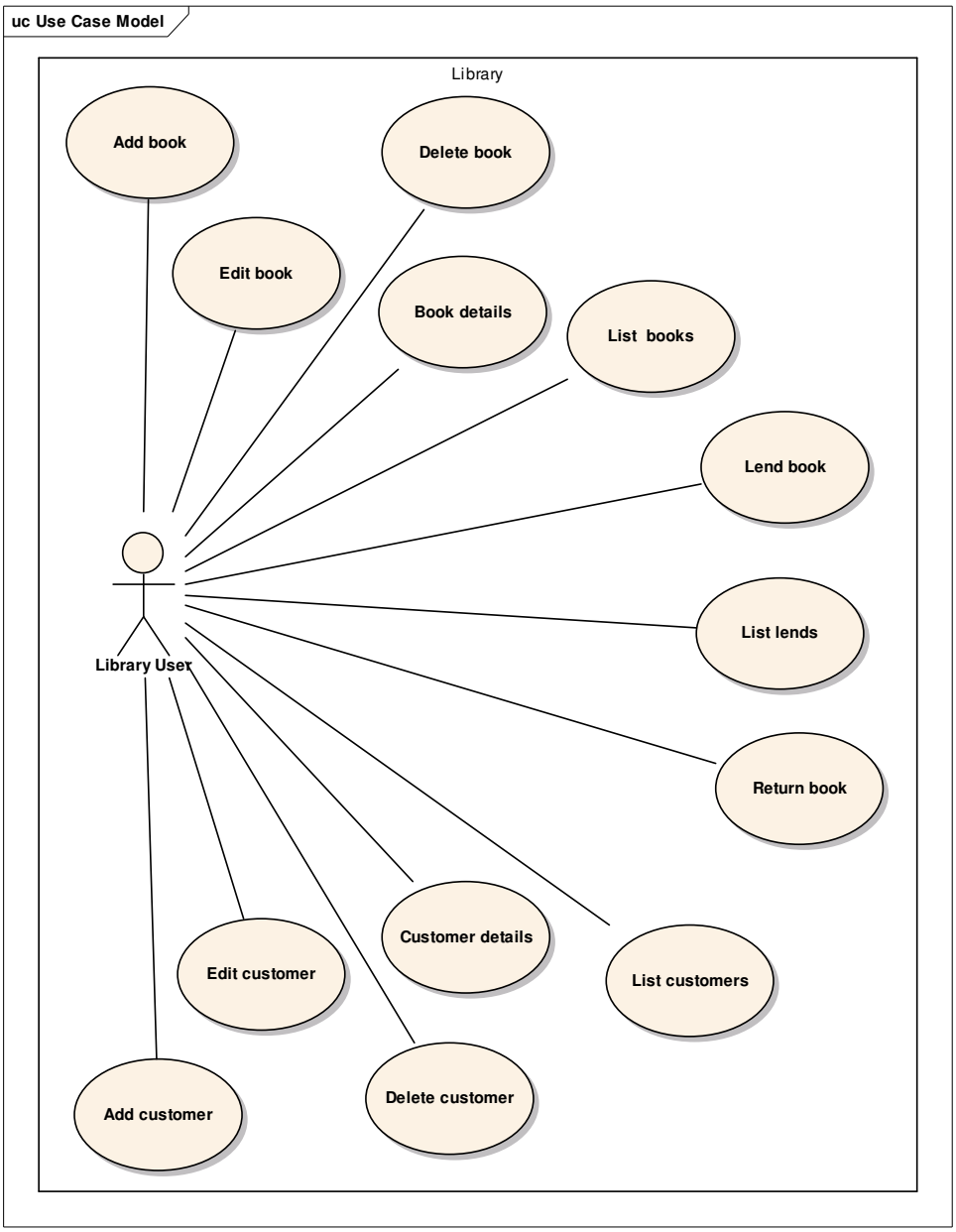
6.2.1. Inleiding

Het prototype is een eenvoudige bibliotheek applicatie. Hierin kunnen onder andere klanten en boeken worden beheerd. Verder is het mogelijk het lenen en terugbrengen van boeken te registreren. Dit zal in de volgende paragraaf verder worden uitgewerkt. Het prototype wordt (in ieder geval in eerste instantie) zo simpel mogelijk gehouden. De prototypes gebruiken waar beschikbaar de extra functionaliteiten van de frameworks. Zo gebruikt het prototype in Beehive het datagrid en in Struts 2 een kalender als er een datum ingevuld moet worden.

[17] Zie voor informatie: http://nl.wikipedia.org/wiki/Model-view-controller.

6.2.2. Use cases

Voor het prototype zijn use cases gedefinieerd. Dit lijkt in eerste instantie voor een klein, simpel prototype nogal veel, maar de use cases lijken veel op elkaar. De use cases staan weergegeven in figuur 3.2.1.



Figuur 3.2.1 - Use case diagram

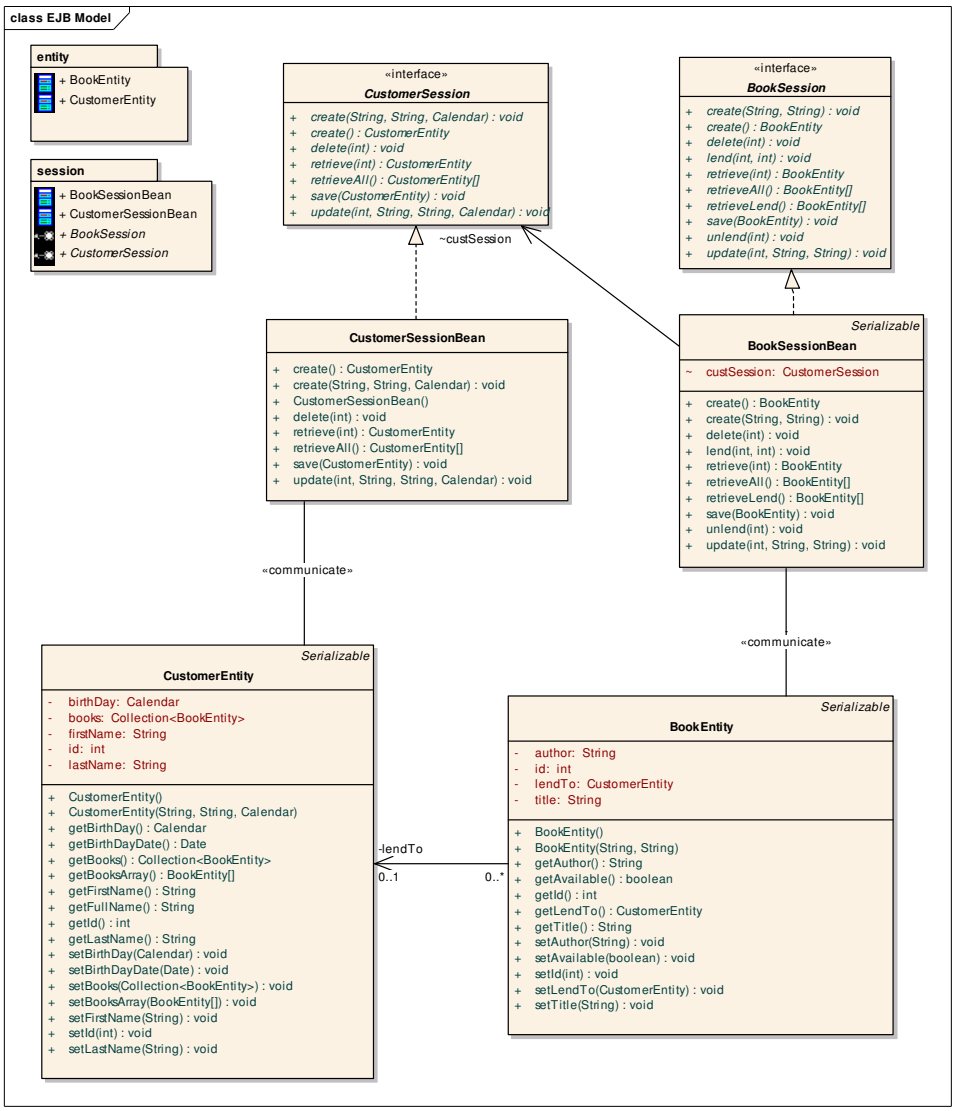
De detailinformatie van een boek omvat onder andere of het is uitgeleend en zo ja, aan wie. De detailinformatie van een klant omvat onder andere welke boeken hij geleend heeft.

Het uitlenen en innemen werkt op de meest simpele manier mogelijk. Bij uitlenen is er een invoerveld voor het klantnummer en voor het boeknummer. Voor het innemen moet alleen het boeknummer ingevoerd worden.

6.2.3. Backend

De backend bestaat uit “Enterprise Java Beans”[18] (EJBs), versie 3. Deze backend wordt voor elk prototype gebruikt, en dus maar één keer geschreven. Voor de businesslogica zijn er “Session Beans” en voor de dataopslag “Entity Beans”. Het klassendiagram staat in figuur 3.2.2.

Een boek kan uitgeleend zijn aan maximaal 1 klant. Een klant kan meerdere boeken geleend hebben.



Figuur 3.2.2 – Klassendiagram Backend.

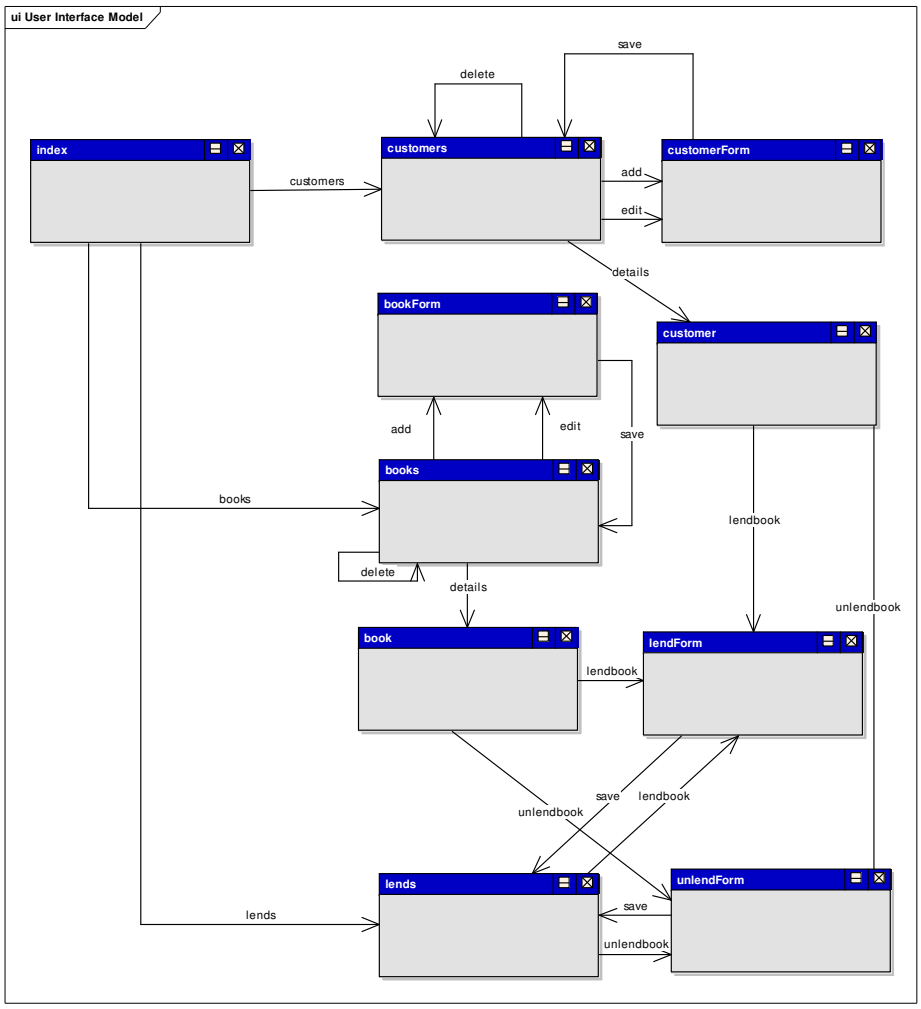
[18] Zie voor informatie: http://en.wikipedia.org/wiki/EJB.

6.2.4. Frontend

De frontend van de applicatie zal bestaan uit JSP pagina's. In de volgende paragrafen wordt aandacht besteedt aan de pageflow en de schermontwerpen.

6.2.4.1. Pageflow

De pageflow tussen de pagina's is te zien in figuur 3.2.3. Voor de duidelijkheid van het diagram staan niet alle flows erin. De mogelijkheid om in de lijst met boeken direct te drukken op "return book" of "lend book" staat er bijvoorbeeld niet in.



Figuur 3.2.3 – Pageflow van de prototypes

6.2.4.2. Schermontwerpen

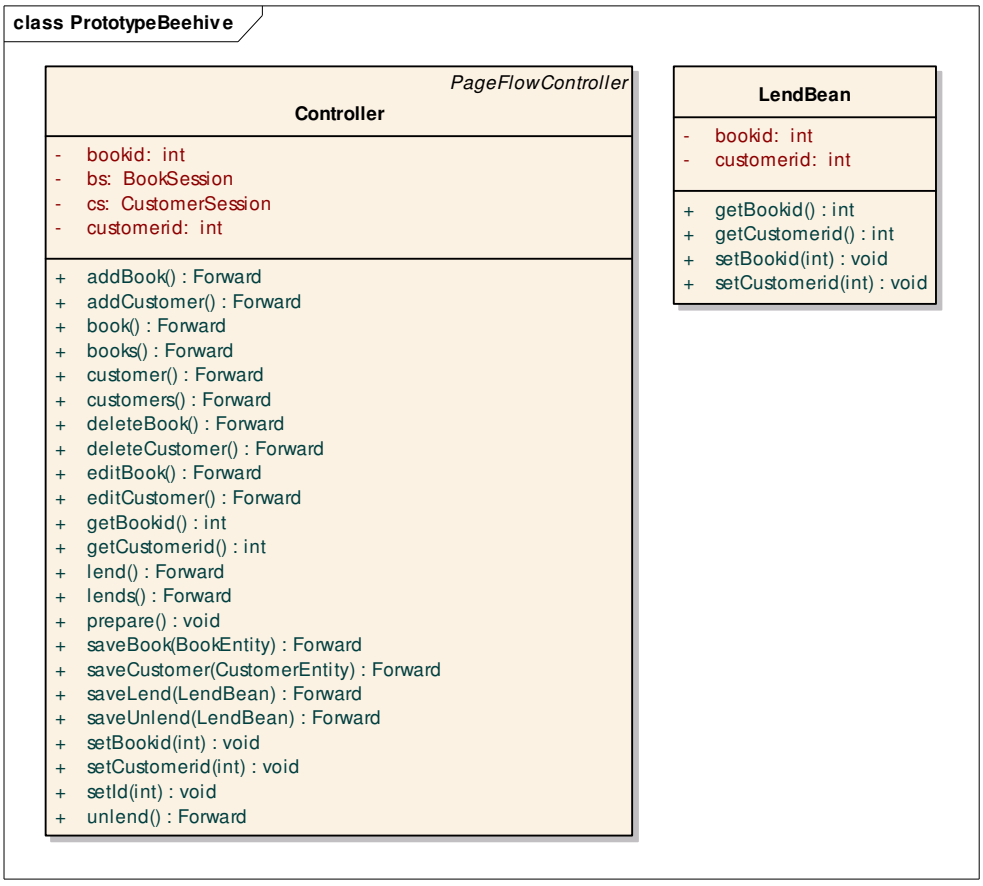
De schermen zijn allemaal zo kaal mogelijk ontworpen. De schermen bestaan uit simpele invoervelden met een label ervoor. Als er een overzicht moet worden weergegeven bestaat dit uit een tabel met rijen en een kopregel. Aan dit basis ontwerp kan per prototype het één en ander aangepast worden. Zo kan gedacht worden aan een kalender als er een datum moet worden ingevuld. Dit zit niet in het schermontwerp omdat niet alle frameworks zulke mogelijkheden hebben.

6.3. Implementatie

Per framework volgt hier een beschrijving over de technische implementatie. Dit hoofdstuk gaat er vanuit dat de kennis over de frameworks al aanwezig is. Dit is behandeld in hoofdstuk 5. Alle code, configuratie en jsp pagina's zijn per framework in de bijlage te vinden. Voor de overzichtelijkheid zijn de "import statements" uit de code gelaten.

6.3.1. Apache Beehive

Het Beehive prototype bestaat uit één controller met daarin alle acties. Daarnaast is er één bean, dit is niets meer dan een Java object met getters en setters, om de input van het uitleen- / inneemformulier in op te slaan.

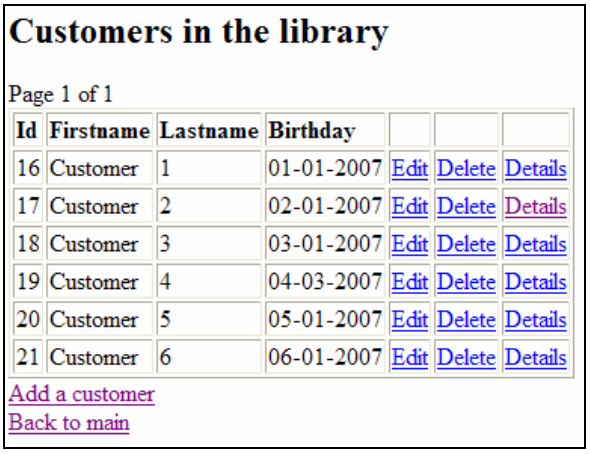


Om het proces wat te verduidelijken volgt hier een voorbeeld.

Als er in het menu voor "Customers" wordt gekozen word de actie "customers.do" aangeroepen. Beehive ziet dat de methode "customers()" moet worden uitgevoerd.

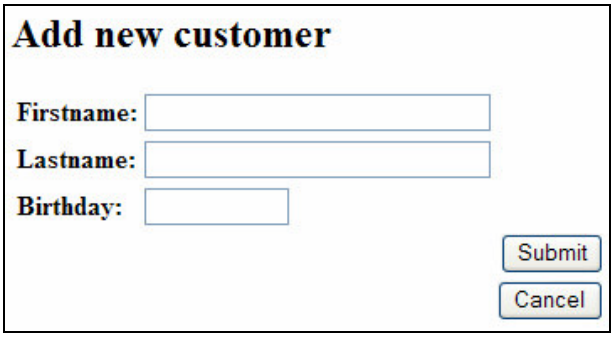
Deze methode haalt de lijst met klanten op bij de "CustomerSessionBean" en stopt deze in de "ActionOutput". Daarnaast geeft de methode aan dat het resultaat "success" is. Door de annotatie bij de methode weet Beehive dat bij "success" de pagina "customers.jsp" hoort. Deze pagina geeft op basis van de klanten in de "pageInput" een lijst van klanten weer met behulp van een "datagrid".

Dit resulteert in de volgende pagina:

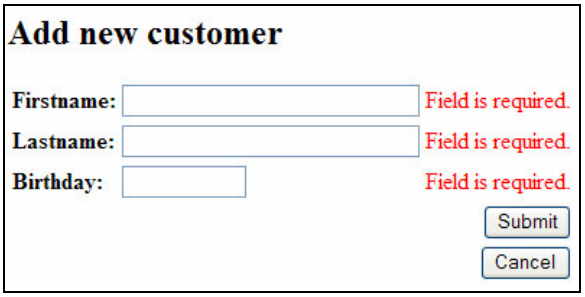


Als er vervolgens in het klantenoverzicht op “Add a customer” wordt gedrukt wordt de actie “addCustomer.do” uitgevoerd. Beehive ziet dat hiervoor de methode “addCustomer()” uitgevoerd moet worden. Deze methode vraagt aan de “CustomerSessionBean” een nieuw “CustomerEntity” object en geeft dit aan het resultaat mee. Het resultaat “success” wordt teruggegeven en Beehive weet dat hier de pagina “customerForm.jsp” bij hoort.

Dit levert het volgende resultaat op:



Wanneer op de knop “Submit” wordt gedrukt word de actie “saveCustomer” uitgevoerd. In de annotatie staat validatie ingesteld voor deze actie . Er zal nu dus worden gevalideerd en dit mislukt omdat er niets is ingevuld. Het invoerscherm wordt weer getoond met de bijbehorende foutmelding(en).

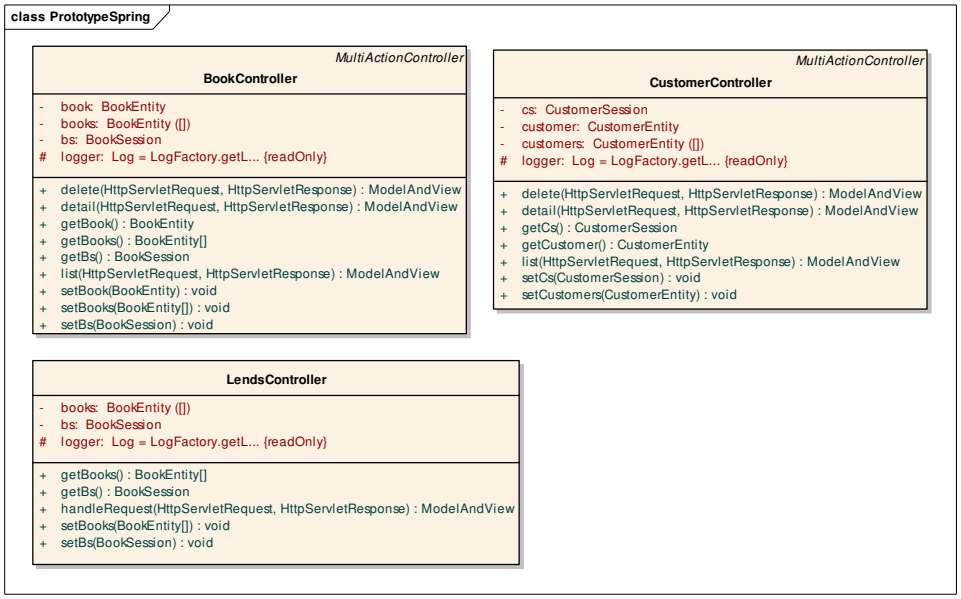


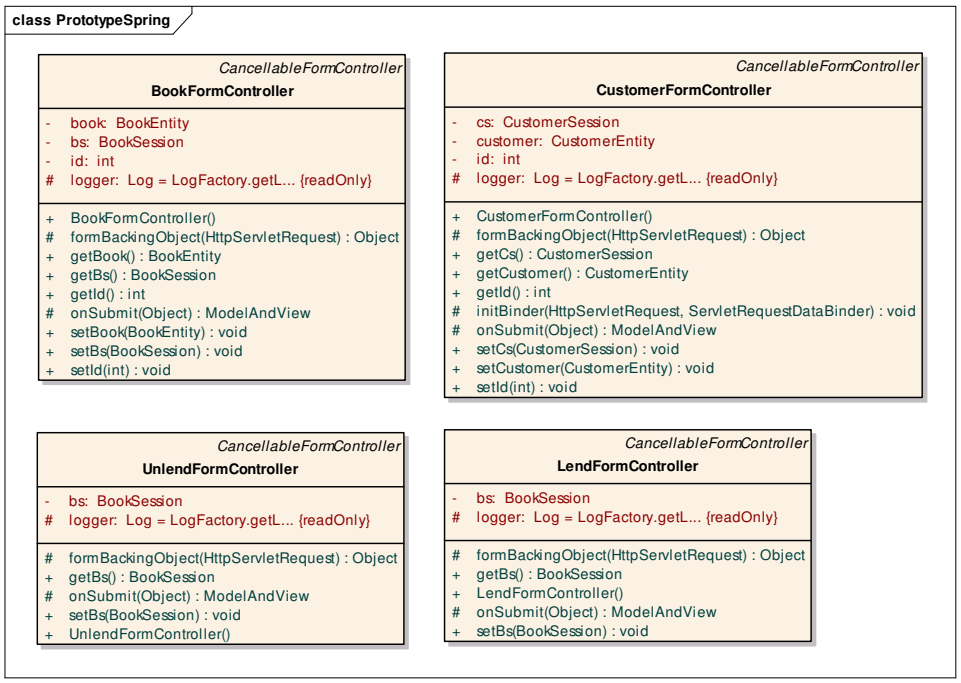
Als de velden vervolgens correct ingevuld worden en er weer op “Submit” wordt gedrukt is de validatie wel succesvol. Dit resulteert in het uitvoeren van de methode “saveCustomer(CustomerEntity)” die op zijn beurt weer de methode “save(CustomerEntity)” van de “CustomerSessionBean” aanroept waarbij het zojuist door Beehive gevulde “CustomerEntity” object wordt meegegeven. Als dit gelukt is wordt als resultaat “success” teruggegeven en toont Beehive weer het klantenoverzicht.

Dit voorbeeld van de werkwijze geldt ook voor de boeken.

6.3.2. Spring MVC

In het Spring prototype zijn er zeven controllers. Voor elk formulier is er een controller en daarnaast is er een controller voor het overzicht van uitgeleende boeken. Alle overige acties die betrekking hebben op boeken of klanten zijn verwerkt in respectievelijk de “BookController” en “CustomerController”.





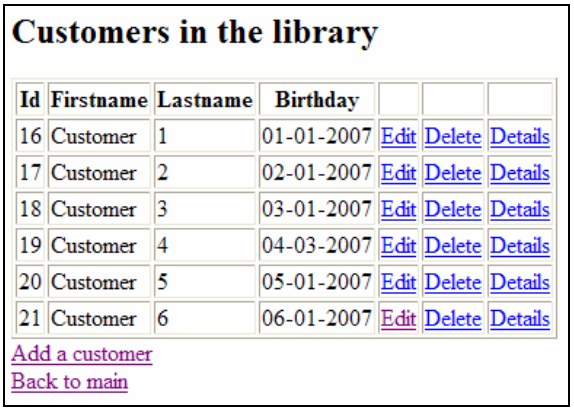
In de Spring configuratie staan deze controllers als bean geregistreerd. Ook de “BookSessionBean” en “CustomerSessionBean” zijn geregistreerd en wanneer nodig worden deze geïnjecteerd in de controllers. Bij de “BookController” en “CustomerController” wordt de methode die uitgevoerd moet worden als parameter in de URL meegegeven. Bij de “BooksController” en “CustomersController” staat dit standaard ingesteld.

Om het proces wat te verduidelijken volgt hier een voorbeeld.

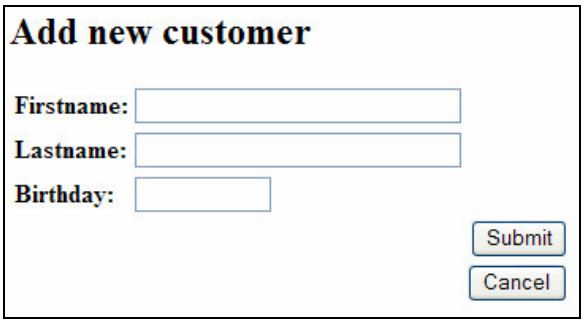
Als er in het menu voor “Customers” wordt gekozen, word de actie “customers.action” aangeroepen. Spring ziet in de configuratie dat voor deze actie de methode “list()” uit

“CustomerController” uitgevoerd moet worden. Deze methode haalt de lijst met klanten op bij de “CustomerSessionBean” en stopt deze in het “Model”. Daarnaast geeft de methode aan dat de view “customers” moet zijn. Spring zoekt vervolgens de bijbehorende JSP pagina op, in dit geval “WEB-INF/jsp/customers.jsp”. Deze pagina geeft op basis van de klanten in het “Model” een lijst van klanten weer.

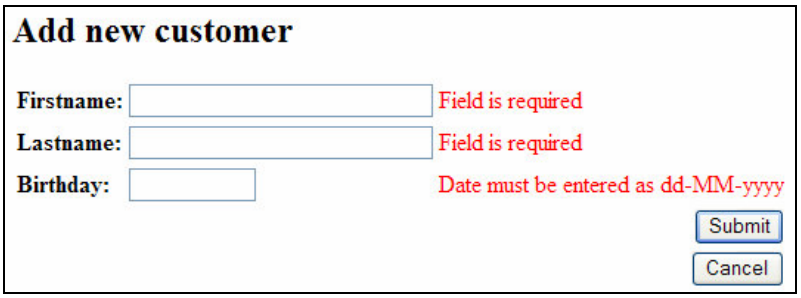
De betreffende JSP staat hieronder:



Als er vervolgens in het klantenoverzicht op “Add a customer” wordt gedrukt wordt de actie “customerForm.action” uitgevoerd. Spring ziet dat hier de controller “CustomerFormController” bijhoort. Deze controller vraagt aan de “CustomerSessionBean” een nieuw “CustomerEntity” object. Dit object geeft de controller terug als “formBackingObject”. Als view wordt de ingestelde “formView” gebruikt, in dit geval “customerForm”. De pagina “customerForm.jsp” wordt getoond, deze staat hieronder:



Wanneer er daarna op de knop “Submit” wordt gedrukt, word de actie “customerForm.action” uitgevoerd. In de configuratie staat dat er met de “CustomerValidator” gevalideerd moet worden. Er zal nu dus gevalideerd worden en dit mislukt omdat er niets is ingevuld. Het invoerscherm wordt nu weer getoond met de bijbehorende foutmelding(en).



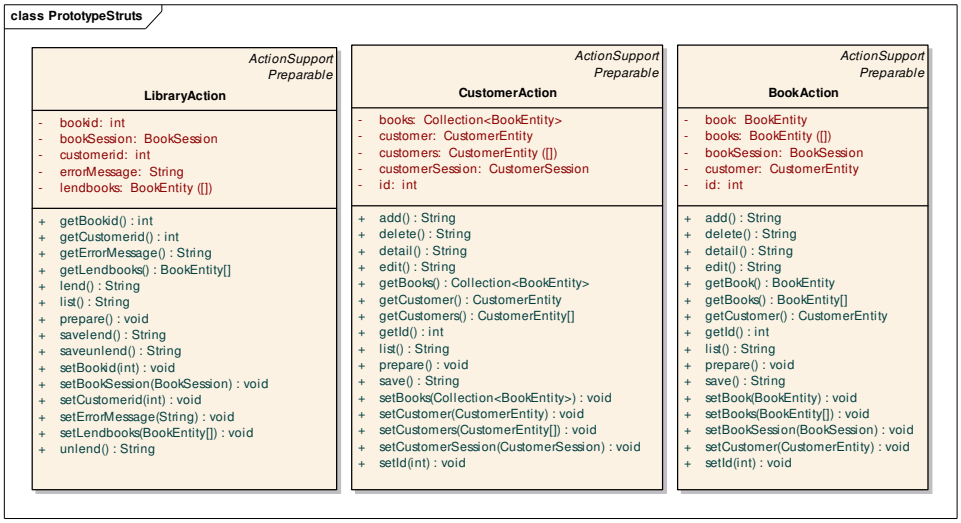
Als de velden nu correct worden ingevuld en er weer op “Submit” wordt gedrukt is de validatie wel succesvol. Dit resulteert in het aanroepen van de methode “onSubmit()” in de “CustomerFormController” die op zijn beurt weer de methode “save(CustomerEntity)” van de

“CustomerSessionBean” aanroept waarbij het zojuist door Spring gevulde “CustomerEntity” object wordt meegegeven. Als dit gelukt is wordt de “successView” (Klantoverzicht) getoond.

Dit voorbeeld van de werkwijze geldt ook voor de boeken.

6.3.3. Struts 2

Het Struts prototype heeft drie Action objecten: “BookAction”, “CustomerAction” en “LibraryAction”. In “BookAction” staan alle acties die betrekking hebben op een boek: lijst van boeken, bewerken, toevoegen, opslaan en weergeven van details voor een boek. “CustomerAction” heeft dezelfde acties als “BookAction” alleen hebben deze betrekking op de klant. In de “LibraryAction” staan alle acties die betrekking hebben op het uitlenen en innemen van de boeken.



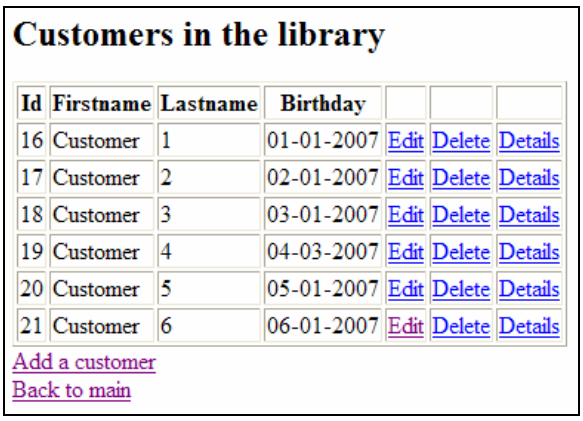
In de “struts.xml” staan de acties benodigd voor de klant use cases gedefinieerd.

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<!-- Include Struts 2 default (from Struts 2 JAR). -->
<include file="struts-default.xml"/>

<!-- Configuration for the default package. -->
    <package name="StrutsSample" extends="struts-default">
       <default-interceptor-ref name="defaultStack"/>
       <action name="customers" class="actions.CustomerAction" method="list">
            <result>/customers.jsp</result>
        </action>

        <action name="customer" class="actions.CustomerAction" method="input">
            <result name="success" type="redirect-action">customers</result>
                 <result name="detail">/customer.jsp</result>
                 <result name="input">/customerForm.jsp</result>
                 <result name="error">error.jsp</result>
        </action>
        …
    </package>
</struts>
```

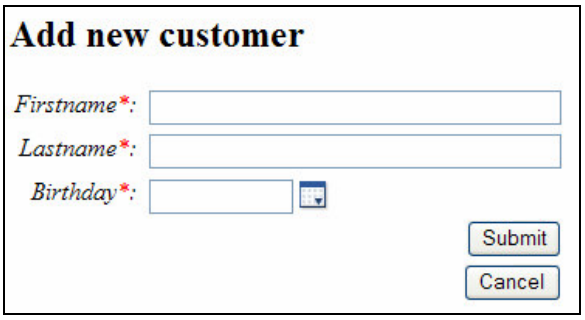
Wat opvalt, is dat niet alle use cases die betrekking hebben op de klant gedefinieerd staan. Een gedeelte van de use cases wordt met de “Actienaam!methodenaam” syntax aangeroepen. Zo worden de details voor een klant aangeroepen met “customer!detail.action”. Er is wel een aparte actie voor customers en customer ondanks dat dit dezelfde klasse is. Dit is gedaan omdat het logischer lijkt om acties die je op één klant doet qua actienaam te scheiden van de acties die je op meerdere klanten doet. Als er een lijst met klanten nodig is, is het vreemd om dit aan een “customer.action” te vragen, dan lijkt het logischer om dit aan “customers.action” te vragen. In dit geval komt het dan neer op de scheiding tussen lijst van klanten en de overige use cases die op een klant betrekking hebben. Deze configuratie methode en scheiding geldt ook voor books en book. De complete “struts.xml” is in de bijlagen te vinden.

Om het proces wat te verduidelijken volgt hier een voorbeeld.

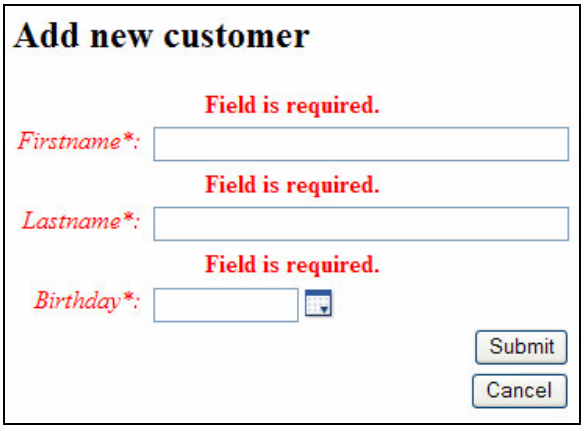
Als er in het menu voor “Customers” wordt gekozen word de actie “customers.action” aangeroepen. Struts ziet in de configuratie dat voor deze actie de methode “list()” uit “CustomerAction” uitgevoerd moet worden. Deze methode haalt de lijst met klanten op bij de CustomerSessionBean en geeft aan dat het resultaat “success” is. Struts ziet dan dat bij “success” de pagina “customers.jsp” hoort, en geeft deze weer. De pagina “customer.jsp” geeft dan vervolgens een lijst met klanten gebaseerd op de property “customers” in de “CustomerAction”.



Als er vervolgens in het klanten overzicht op “Add a customer” wordt gedrukt wordt de actie “customer!add.action” uitgevoerd. Struts ziet dat niet de standaard methode van de actie “customer” moet worden uitgevoerd maar de methode “add()”. Deze methode vraagt aan de “CustomerSessionBean” een nieuw “CustomerEntity” object en geeft het resultaat “input” terug. Struts ziet dat bij dit resultaat de pagina “customerForm.jsp” hoort, en geeft deze weer.



Wanneer er daarna op de knop “Submit” wordt gedrukt wordt de actie “customer!save.action” uitgevoerd met de bijbehorende methode “save()”. Voor de actie “customer” is validatie ingesteld en bij de methode “save()” staat niet aangegeven dat de validatie overgeslagen moet worden. Er gaat dus eerst gevalideerd worden voordat de methode daadwerkelijk wordt aangeroepen. Hieruit komt dat de velden niet zijn ingevuld. Het invoerscherm wordt weer getoond met de bijbehorende foutmelding(en).



Als de velden nu correct ingevuld worden en er weer op “Submit” wordt gedrukt is de validatie wel succesvol. Dit resulteert in het aanroepen van de methode “save()” die op zijn beurt weer de methode “save(CustomerEntity)” in de “CustomerSessionBean” aanroept waarbij het zojuist door Struts gevulde “CustomerEntity” object wordt meegegeven.

Dit voorbeeld van de werkwijze geldt ook voor de boeken.

Een belangrijk onderdeel van Struts 2 is de integratie met Spring. Om dit ook te belichten is hiervan gebruik gemaakt. Spring zoekt de EJBs op en injecteert ze in de Acties. Hiervoor zijn in de Spring configuratie twee beans gedefinieerd die twee van de Session Beans instantiëren. Dit scheelt (dubbele) regels code, en het zorgt voor een centrale plek van waaruit de koppeling met de backend geconfigureerd kan worden.

7. Vergelijking

7.1. Inleiding

In dit hoofdstuk zullen de drie frameworks worden vergeleken op het gebied van configuratie, code, functionaliteit, flexibiliteit, performance, testbaarheid en toolondersteuning. De resultaten worden in de conclusie weergegeven in een tabel met een score. Deze score kan zijn: --, -, + en ++. Deze scores betekenen tenzij anders aangegeven: Zeer slecht, slecht, goed en zeer goed. Als de test uit verschillende onderdelen bestaat heeft elk onderdeel een wegingsfactor. Aan de scores zijn de waardes 1, 2, 3 en 4 gekoppeld. Voor elk onderdeel wordt de score vermenigvuldigd met de wegingsfactor. De resultaten hiervan worden bij elkaar op geteld en door het totaal van de weginsfactoren gedeeld. Het resultaat is een getal tussen de 1 en 4 wat overeenkomt met: --, - , + en ++. De formule is als volgt:

$$Score_{totaal} = \frac{\sum_{n}^{i=0} (Score_i * Weging_i)}{\sum_{n}^{i=0} Weging_i}$$

7.2. Configuratie

7.2.1. Inleiding

In dit onderdeel zal de benodigde configuratie voor de frameworks worden vergeleken. Per framework zal de configuratie voor dezelfde actie en validatie worden gebruikt. Hierbij zal gekeken worden naar de hoeveelheid configuratie en de complexiteit er van.

7.2.2. Beehive

Beehive configuratie voor één actie:

```
@Jpf.Action(
  forwards = {
    @Jpf.Forward(
      name = "success",
      path="customers.jsp",
      actionOutputs={
        @Jpf.ActionOutput(
          name="customersInput",
          type=CustomerEntity[].class,
          required=true
        )
      }
    )
  }
)
```

Beehive configuratie voor validatie:

```
@Jpf.Action(
        forwards = {
            @Jpf.Forward(
                name = "success",
                action = "customers",
                redirect=true),
            @Jpf.Forward(
                  name = "error",
                  action = "error",
                redirect=true)
        },
        validatableProperties={
                @Jpf.ValidatableProperty(
                    propertyName="firstName",
                    displayName="Field",
                    validateRequired=@Jpf.ValidateRequired()
                ),
                @Jpf.ValidatableProperty(
                      propertyName="lastName",
                      displayName="Field",
                      validateRequired=@Jpf.ValidateRequired()
                  ),
                  @Jpf.ValidatableProperty(
                              propertyName="birthDayDate",
                              displayName="Field",
                              validateRequired=@Jpf.ValidateRequired()
                          )
            },
            validationErrorForward=@Jpf.Forward(name="fail",
             path="customerForm.jsp")
    )
```

Configuratie voor gebruik EJBs:

EJBs kunnen niet in de configuratie worden ingesteld, dit kan alleen met code worden gedaan.

7.2.3. Spring

Spring configuratie voor één actie:

```
<bean id="propsMethodNameResolver" class="org.springframework.web.servlet.mvc.multi-
  action.PropertiesMethodNameResolver">
  <property name="mappings">
    <value>/customers.action=list</value>
  </property>
</bean>

<bean id="CustomersController" class="controllers.CustomerController">
  <property name="methodNameResolver" ref="propsMethodNameResolver"/>
  <property
    name="cs"
    ref="CustomerSession" />
</bean>

<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrl-
  HandlerMapping">
  <property name="mappings">
    <props>
      <prop                        key="/customers.action">CustomersController</prop>

    </props>
  </property>
</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.Internal-
  ResourceViewResolver">
  <property name="viewClass">
    <value>org.springframework.web.servlet.view.JstlView</value>
  </property>
  <property name="prefix"><value>/WEB-INF/jsp/</value></property>
  <property name="suffix"><value>.jsp</value></property>
</bean>
```

Spring configuratie voor formulieren met validatie:

```
<bean id="CustomerFormController" class="controllers.CustomerFormController">
  <property name="cs" ref="CustomerSession" />
  <property name="formView" value="customerForm" />
  <property name="sessionForm" value="true" />
  <property name="successView" value="redirect:customers.action"/>
  <property name="cancelView" value="redirect:customers.action"/>
  <property name="validator">
    <bean class="validators.CustomerValidator" />
  </property>
</bean>
```

De daadwerkelijke validatie wordt in de code van de validator gedaan.

Configuratie voor het gebruik van EJBs:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE          beans          PUBLIC          "-//SPRING//DTD          BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean
        id="customerSession"
        class="org.springframework.jndi.JndiObjectFactoryBean">
        <property
            name="jndiName"
            value="CustomerSessionBean/remote" />
    </bean>
</beans>
```

7.2.4. Struts 2

Struts configuratie voor één actie:

```
<action name="customers" class="actions.CustomerAction" method="list">
  <result>/customers.jsp</result>
</action>
```

Voor het gebruik van validatie voor formulieren hoeft er geen extra configuratie gedaan te worden.

Configuratie voor het gebruik van EJBs:

De Spring-plugin moet geactiveerd worden en net als bij spring de EJB als bean gedeclareerd.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE          beans          PUBLIC          "-//SPRING//DTD          BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean
        id="customerSession"
        class="org.springframework.jndi.JndiObjectFactoryBean">
        <property
            name="jndiName"
            value="CustomerSessionBean/remote" />
    </bean>
</beans>
```

7.2.5. Conclusie

Beehive heeft voor een actie niet veel configuratie maar omdat deze configuratie bij elke actie methode moet worden gedaan onstaat er veel (dubbele) configuratie.

Spring heeft door het feit dat er voor formulieren aparte controllers nodig zijn en omdat er een “ViewResolver” geconfigureerd moet worden
veel configuratie. Door deze “ViewResolver” wordt de configuratie ook complexer.

Struts heeft de minste configuratie voor een actie. Voor formulieren hoeft geen extra configuratie gedaan te worden, dus dat scheelt behoorlijk. De weinige configuratie komt door het gebruik van veel standaard waarden. Des te meer aangepaste Struts er nodig is, des te meer configuratie zal er zijn.

Ook door het gebruik van “DynamicMethodInvocation” of “Wildcard mappings” is de configuratie erg klein te houden.

Onderdeel (wegingsfactor)	**Beehive**	**Spring**	**Struts 2**
Hoeveelheid (1)	+	+	++
Complexiteit (1)	+	-	++
Totaal	+	-	++

7.3. Code

7.3.1. Inleiding

In dit onderdeel is de code voor de verschillende frameworks bekeken. Allereerst is er gekeken naar het aantal regels code die gemiddeld nodig is voor een actie. En omdat niet elke regel code een statement is, is er ook gekeken naar het aantal statements. Vervolgens staat er nog even kort per framework weergegeven hoe complex de code is. Per framework zal de code voor dezelfde actie en validatie worden gebruikt als in het vorige onderdeel.

7.3.2. Hoeveelheid code

De hoeveelheid code is op twee onderdelen vergeleken. Allereerst is het gemiddelde aantal regels code per actie (-methode) gemeten. Bij Beehive wordt hiervoor ook de annotatie meegenomen. Daarna is het gemiddelde aantal statements gemeten. De Beehive annotaties wordt hierbij als één statement gerekend.

Dit leverde het volgende resultaat op:

Framework	Regels code	Aantal statements
Beehive	30	11
Spring	14	11
Struts	9	6

Hieruit blijkt dat Beehive de meeste regels code heeft, dit komt grotendeels door de annotaties. Dit blijkt wel uit het aantal statements, dat scheelt een factor drie met het aantal regels code. Uit dit onderdeel blijkt ook dat Struts veruit de minste code heeft, Spring en Beehive komen (zonder de annotaties) vrijwel gelijk uit.

7.3.3. Beehive

De code van Beehive is doordat de configuratie met annotaties wordt gedaan erg complex. Annotaties zouden een hulp middel moeten zijn om de configuratie te vereenvoudigen maar door de grootte van deze annotaties wordt de code erg complex. Een tweede complicerende factor is het feit dat alle waardes die door de jsps gebruikt moeten worden expliciet aan de “Forward” moeten worden meegegeven. De view is los gekoppeld van de code, alleen door de annotatie is deze in zekere zin toch gekoppeld.

```
@Jpf.Action(
  forwards = {
    @Jpf.Forward(
      name = "success",
      path="customers.jsp",
      actionOutputs={
        @Jpf.ActionOutput(
          name="customersInput",
          type=CustomerEntity[].class,
          required=true
        )
      }
    )
  }
)
public Forward customers() {
  Forward forward = new Forward("success");
  …
  forward.addActionOutput("customersInput", … );
  return forward;
}
```

Voor het opzoeken van de EJBs is de volgende code benodigd:

```
String factory = "org.jnp.interfaces.NamingContextFactory";
String url = "jnp://localhost:1099";
Properties props = new Properties();
props.setProperty(Context.INITIAL_CONTEXT_FACTORY, factory);
props.setProperty(Context.PROVIDER_URL, url);

InitialContext ctx = new InitialContext(props);

Object ref2 = ctx.lookup("CustomerSessionBean/remote");
customerSession                          =                          (CustomerSession)
javax.rmi.PortableRemoteObject.narrow(ref2,CustomerSession.class);
```

7.3.4. Spring

De code van Spring wordt erg complex doordat het “Model” met de “View” moet worden teruggegeven door de actie. Dit zorgt ook voor een koppeling tussen de viewpagina en de controller, terwijl dat er volgens Spring niet is.

```
public ModelAndView handleRequest(HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
  return new ModelAndView(“viewname”, “modelname”, modelvalue);
}
```

Een ander nadeel van de Spring code is dat de acties aan de Servlet API gebonden zijn (de HttpServletRequest en HttpServletReponse). Met testen kan dit problemen op leveren, daarom worden bij Spring mock objecten voor deze twee objecten geleverd, maar het is geen ideale situatie.

Doordat de configuratie met de backend niet in Spring wordt gedaan is hier geen code voor nodig.

7.3.5. Struts 2

De code voor een Struts 2 actie is erg simpel. Het enige wat er gedaan hoeft te worden is een String teruggeven met daarin het resultaat. Eventuele waarden die de “View” nodig heeft kunnen in het actie object worden opgeslagen en doormiddel van bijbehorende getters opgevraagd worden vanuit de “View”. De actie klasse kan interfaces implementeren om extra functionaliteit te verkrijgen, maar dit is niet verplicht.

```
public class CustomerAction {
  public String execute() {
          return “success”;
  }
}
```

De eenvoud van deze code maakt onderhoud erg gemakkelijk.
Doordat de configuratie met de backend niet in Spring wordt gedaan is hier geen code voor nodig.

7.3.6. Conclusie

Onderdeel (wegingsfactor)	Beehive	Spring	Struts 2
Hoeveelheid (1)	- / +	+	++
Complexiteit (1)	- / +	+	++
Totaal	- / +	+	++

Bij Beehive staan 2 waarden genoemd. De eerste waarde (-) is de score met de annotaties meegewogen. Hierdoor ontstaat een enorme hoeveelheid code en wordt de code erg complex. De tweede score (+) is de score zonder de annotaties meegewogen. De code die dan overblijft, is redelijk kort en niet erg complex. In de visie van Beehive staat dat de annotaties de code moeten verkleinen, dit maken ze niet waar. Spring heeft een redelijk korte code en deze is ook niet erg complex. Alleen als er met formulieren gewerkt wordt, is de code iets complexer doordat er een aantal functies geïmplementeerd moeten worden.

Struts 2 komt als beste uit dit onderdeel. Het heeft de minste code, en de code is het minst complex. Er hoeven geen interfaces te worden geïmplementeerd, maar als het nodig is kan dat wel. Ook is de actie methode het meest eenvoudig, dit bestaat alleen uit het vullen van het actie object met de juiste waardes en het teruggeven van een String. In tegenstelling tot Spring is ook Struts 2 ook niet afhankelijk van de Servlet API. In Spring moet zowel het model als de view door de actie methode worden teruggegeven. In Beehive moet ook het model worden teruggegeven maar hier wordt niet specifiek de view genoemd, alleen een resultaat code aan de hand waarvan Beehive de resultaatpagina opzoekt.

7.4. Functionaliteit

7.4.1. Inleiding

In dit onderdeel zal gekeken worden naar de functionaliteit die de frameworks bieden. Allereerst wordt de functionaliteit op GUI gebied vergeleken. Daarna zal de functionaliteit op het gebied van formulier validatie worden vergeleken.
Als laatste is er nog een korte vergelijking over de security van de frameworks.

7.4.2. GUI

In Beehive zijn de GUI Functionaliteiten beperkt tot de standaard HTML componenten, een datagrid en een tree.

Spring bevat behalve standaard html geen GUI functionaliteiten.

Struts bevat veel extra gui functionaliteiten, onder andere een datepicker. Daarnaast is er al ingebouwde AJAX ondersteuning.

7.4.3. Validatie

Beehive
Om in Beehive een actie te laten valideren dient er een extra parameter aan de "@Jpf.Action" annotatie te worden meegegeven, dit is: "validatableProperties". Daarin staat een lijst van "@Jpf.ValidatableProperty" annotaties.

```
@Jpf.Action(
        forwards = {
            @Jpf.Forward(
                name = "success",
                action = "customers",
                redirect=true)
        },
        validatableProperties={
                @Jpf.ValidatableProperty(
                    propertyName="firstName",
                    displayName="Field",
                    validateRequired=@Jpf.ValidateRequired()
                ),
                @Jpf.ValidatableProperty(
                      propertyName="lastName",
                      displayName="Field",
                      validateRequired=@Jpf.ValidateRequired()
                 ),
          @Jpf.ValidatableProperty(
                      propertyName="birthDayDate",
                      displayName="Field",
                      validateRequired=@Jpf.ValidateRequired()
                 )
            },
            validationErrorForward=@Jpf.Forward(name="fail",
path="customerForm.jsp")
    )

    public Forward saveCustomer(CustomerEntity form)
    {
          …
  return new Forward("success");
    }
```

Spring
Voor het valideren van de formulier output kan de volgende code worden gebruikt:

```
public class CustomerValidator implements Validator {


  /**
  * This Validator validates just CustomerEntity instances
  */
  public boolean supports(Class clazz) {
    return CustomerEntity.class.equals(clazz);
  }

  public void validate(Object obj, Errors e) {
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "firstName", "field.required");
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "lastName",  "field.required");
  }
}
```

Struts
Struts configuratie voor validatie:

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
       "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="customer.firstName">
     <field-validator type="requiredstring">
        <message key="requiredstring" />
      </field-validator>
  </field>
  <field name="customer.lastName">
     <field-validator type="requiredstring">
        <message key="requiredstring" />
      </field-validator>
  </field>
  <field name="customer.birthDayDate">
         <field-validator type="required">
     <message key="requiredstring" />
    </field-validator>
  </field>
</validators>
```

Het nadeel van deze validatie is dat je alleen per actie en niet per actie methode kan valideren. Bij DynamicMethodInvocation kan dit een probleem zijn. Het is wel mogelijk om voor bepaalde actiemethodes de validatie uit te schakelen door de annotatie “@SkipValidation” te gebruiken maar als er in één actie twee methodes op een verschillende manier gevalideert moete worden treedt er een probleem op. Dit probleem is met de wildcard mappings methode niet aanwezig.

7.4.4. Security

Security is in de prototypes niet aanbod gekomen. Daarnaast is er wel gekeken wat er over de security mogelijkheden te vinden is.

In Beehive is het mogelijk op basis van de rol van de gebruiker al dan niet toegang te geven. Het is met de property “login-required”en “rolesAllowed” bij de “@Jpf.Action” annotatie aan te geven welke rollen er een bepaalde actie mogen uitvoeren. Ook is het mogelijk om dit bij de “@Jpf.Controller” op te geven, dit geldt dan voor alle acties in de controller, tenzij de annotatie bij de actiemethode dit overschrijft.

Voor Spring is er het beveiligingsframework “Acegi”[19], of dit door de integratie die Struts 2 met Spring biedt ook voor Struts 2 gebruikt kan worden is niet bekend. “Acegi” is verder niet bekeken, dit valt te ver buiten de context en beschikbare tijd van het onderzoek.

[19] Zie voor informatie: http://www.acegisecurity.org/.

Bij Struts kan door gebruik te maken van de “RolesInterceptor” op basis van rollen. Hiermee is het mogelijk om per gedeclareerde actie aan te geven welke rollen toegang hebben tot de actie. Als de rol dit niet toe staat wordt er een 403 http error[20] gegeven. Het is niet mogelijk om per actiemethode aan te geven wie al dan niet toegang heeft. Als “DynamicMethodInvocation” wordt gebruikt of “Wildcard mappings” is het niet optimaal te beveiligen.

7.4.5. Conclusie

Onderdeel (wegingsfactor)	Beehive	Spring	Struts 2
GUI (1)	+	-	++
Validatie (1)	+	+	+
Security (1)	++	- / +	+
Totaal	+	- / +	+

De “- / +” bij Spring geeft aan dat Spring zelf geen security biedt, maar dat er wel security mogelijk is (Door gebruik van “Acegi”).
Uit dit onderdeel kan geconcludeerd worden dat Struts op GUI gebied de meeste functionaliteiten biedt. Voor validatie en security is de functionaliteit vrijwel gelijk. Alleen Beehive biedt iets meer geïntegreerde mogelijkheden.

[20] Zie voor informatie: http://en.wikipedia.org/wiki/HTTP_403.

7.5. Flexibiliteit

7.5.1. Inleiding

De flexibiliteit van het framework zal op een aantal punten vergeleken worden. Allereerst zal er gekeken worden hoeveel er aangepast moet worden in de code en configuratie als er een wijziging is. Hoe minder er in de code gewijzigd moet worden, hoe beter de score.

7.5.2. Beehive

In Beehive is niets aanpasbaar zonder dat er in de code aanpassingen moeten worden gemaakt. Dit komt doordat alles met annotaties wordt geregeld.

7.5.3. Spring

In Spring is de backend-koppeling in de configuratie regelbaar. De resultaat pagina die bij een actie hoort is op twee plekken regelbaar. De eerste is in de code, deze geeft de naam van de view terug. De tweede is in de configuratie, hierin wordt de naam van de view gekoppeld aan een fysieke pagina. Validatie wordt in de code gedaan, als dit veranderd moet er dus gecodeerd worden.

7.5.4. Struts

In Struts is vrijwel alles in de configuratie regelbaar. Alleen als een actie een ander resultaat terug moet geven (dit is niet direct een pagina) moeten er in de code aanpassingen worden gedaan. Ook validatie staat in een xml bestand dus hiervoor hoeft ook niet te worden gecodeerd.

7.5.5. Conclusie

De conclusie is dat Struts het meest flexibel is omdat er veel in de configuratie files staat. Als er iets moet wijzigen hoeven er niet direct wijzigingen in de code te worden gemaakt, terwijl dit bij de andere frameworks in veel gevallen wel noodzakelijk is.

Onderdeel (wegingsfactor)	**Beehive**	**Spring**	**Struts 2**
Aanpasbaarheid zonder code (3)	--	+	++
Flexibiliteit (2)	+	+	++
Koppeling controller/View (1)	++	--	++
Totaal	-	+	++

7.6. Performance

7.6.1. Inleiding

In dit onderdeel is de performance van de frameworks gemeten. Dit is gedaan met behulp van het programma jMeter[21]. Met dit programma is het mogelijk om verscheidene soorten testen uit te voeren. Voor dit vergelijkingsonderdeel is gebruik gemaakt van de mogelijkheid om HTTP requests uit te voeren en daarvan de reactie tijd op te slaan.

Voor de test worden de overzichtspagina's van klanten, boeken en uitleningen opgevraagd. Dit wordt per prototype getest. Nadat de tests zijn uitgevoerd wordt de applicatie server opnieuw opgestart. De tests zijn vanaf een andere machine dan de applicatie server uitgevoerd, dit om te zorgen dat de belasting door het testprogramma geen invloed heeft op de resultaten.

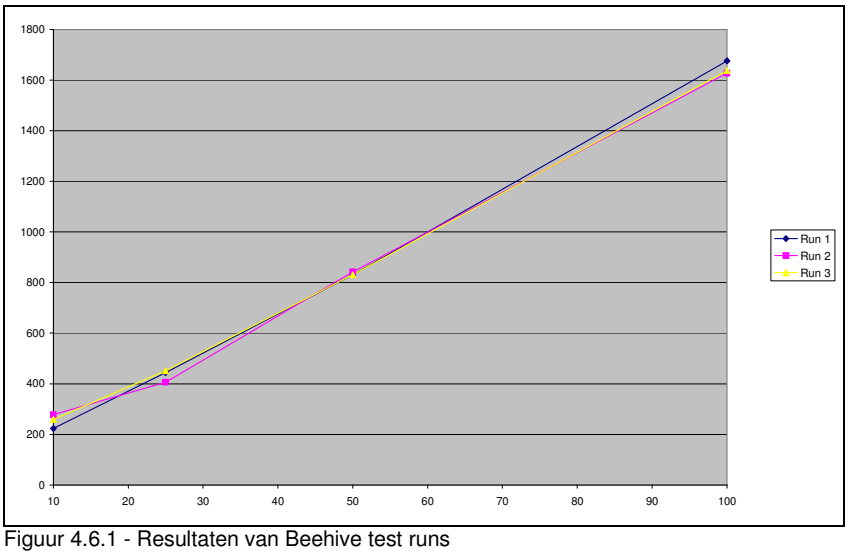
De test bestaat uit vier onderdelen, de eerste keer met 10 gebruikers (threads), daarna 25, 50 en als laatste 100 gebruikers. Er wordt eerst door 10 gebruikers de klanten pagina opgevraagd, daarna door 10 gebruikers de boeken pagina enz. Nadat alle pagina's zijn geweest wordt dit proces herhaald, tot dat er per pagina een totaal van 1000 requests is. In totaal komt dit op 3000 requests. Direct daarna wordt de test met 25 gebruikers uitgevoerd volgens dezelfde procedure, deze komt ook op totaal 3000 requests. Dit gaat zo verder voor alle tests. Om tijdelijke omstandigheden van buiten af uit te sluiten van het resultaat is de complete test drie keer (rond 9, 12 en 15 uur) uitgevoerd op één dag.

Uit elke test (en testonderdeel) komt een gemiddelde tijd waarin het resultaat is ontvangen. De resultaten zijn in de volgende grafieken weergegeven. Op de x-as staat het aantal gebruikers (threads), op de y-as het gemiddelde aantal milliseconden. De resultaten worden eerst per framework weergegeven, daarna volgt het totale plaatje, hierin staat het gemiddelde van de drie uitgevoerde testen per framework.

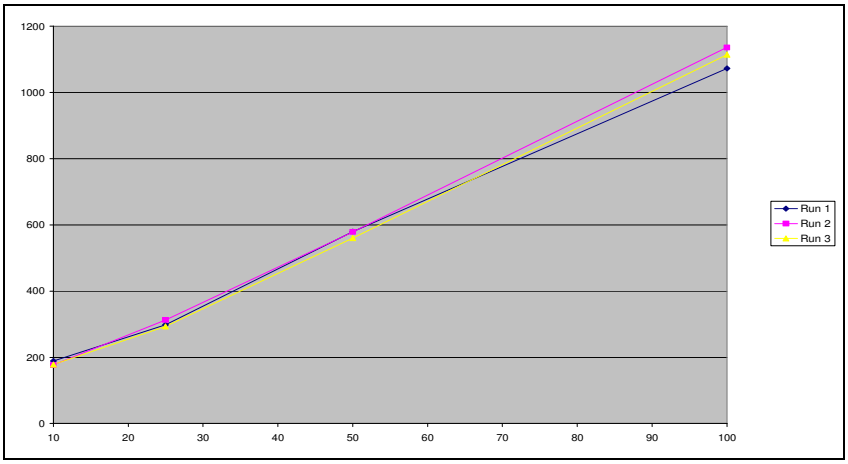
[21] Apache jMeter, http://jmeter.apache.org

7.6.2. Resultaten

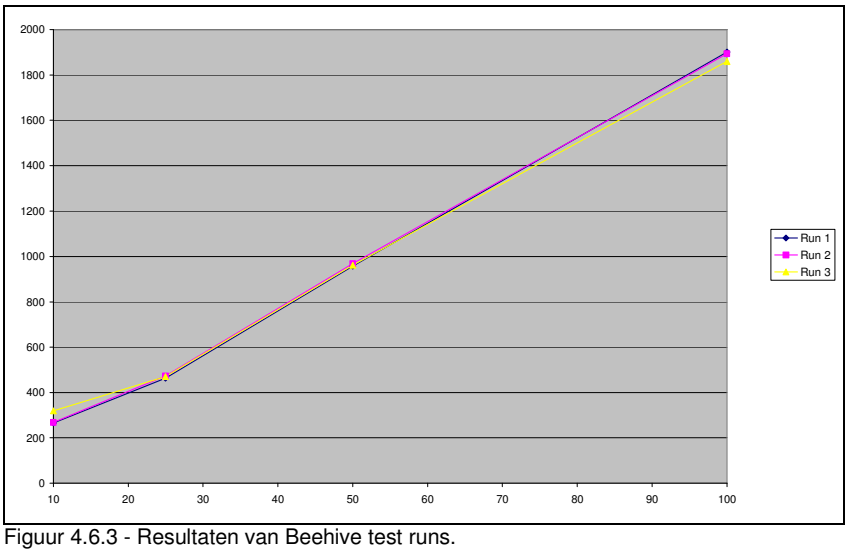
Hieronder staan de resultaten van de drie test runs grafisch weergegeven.



Figuur 4.6.1 - Resultaten van Beehive test runs

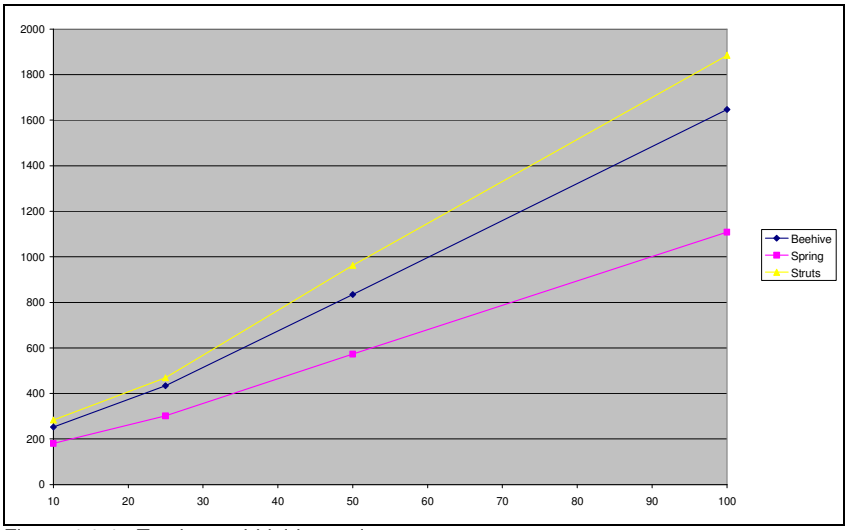


Figuur 4.6.2 - Resultaten van Spring test runs



Figuur 4.6.3 - Resultaten van Beehive test runs.

7.6.3. Conclusie



Figuur 4.6.4 - Totale gemiddelde resultaat.

Opvallend is dat alle frameworks de eerste keer even een soort opstarttijd nodig hebben. Waar dit precies mee te maken heeft is niet bekend, maar waarschijnlijk zal de applicatieserver na de eerste request de benodigde objecten in zijn cache hebben. De tijd neemt zo goed als lineair toe naar mate het aantal gebruikers hoger wordt.

Uit de test blijkt dat Spring naar mate het aantal gebruikers hoger wordt veruit het snelst is. Struts 2 daarin tegen is het langzaamst. Het verschil tussen Spring en Struts is een factor 1,7. Naarmate het aantal bezoekers toeneemt, is Struts dus een serieuze aanslag op de performance / benodigde capaciteit. Ook Beehive is beduidend langzamer dan Spring maar minder langzaam dan Struts 2, het scheelt een factor 1,4 met Spring.

Onderdeel (wegingsfactor)	**Beehive**	**Spring**	**Struts 2**
Performance	+	++	-

7.7. Testbaarheid

7.7.1. Inleiding

In dit onderdeel zal de testbaarheid van de frameworks buiten de applicatieserver worden bekeken.

Er zal op twee punten worden vergeleken, ten eerste hoe de algemene testbaarheid is en ten tweede hoe gemakkelijk het is om de backend te vervangen door een surrogaat.

7.7.2. Beehive

Beehive acties zijn gemakkelijk te testen. Er hoeven geen speciale parameters te worden meegegeven. Alleen door het feit dat backend in de code van een actie wordt geconfigureerd wordt het moeilijk om deze te vervangen door een surrogaat backend.

7.7.3. Spring

Spring acties zijn doordat er bij elke actie een HttpServletRequest en HttpServletReponse moet worden meegegeven moeilijker te testen. Hiervoor heeft Spring een oplossing bedacht, ze leveren bij het framework als surrogaat (Mock) objecten hiervoor, maar het blijft een extra stap. De backend daar in tegen is door de “Dependency Injection” eenvoudig te vervangen, hiervoor kan bijvoorbeeld het programma jMock gebruikt worden.

7.7.4. Struts

De Struts acties zijn doordat de actie methodes geen extra parameters nodig hebben eenvoudig te testen. Ook de backend is net als bij Spring door de “Dependency Injection” eenvoudig te vervangen.

7.7.5. Conclusie

De conclusie is dat Beehive doordat het niet gemakkelijk mogelijk is om de backend te vervangen slecht te testen is. Spring en Struts zijn goed te testen, Struts 2 zelfs heel goed door de eenvoud van de acties.

Onderdeel (wegingsfactor)	**Beehive**	**Spring**	**Struts 2**
Testbaarheid (1)	+	+	++
Vervangingbaarheid Backend (3)	--	++	++
Totaal	-	++	++

7.8. Toolondersteuning

Tool	Beehive	Spring	Struts 2
BEA Workshop for Weblogic	++	-	-
Spring IDE	--	++	--
MyEclipse	-	+	-
ExadelStudio (Wordt JBoss)	-	+	-
Totaal	++	++	-

De waarde "-" geeft aan dat er geen ondersteuning voor is maar dat je wel handmatig in de tool kan programmeren voor het framework. De waarde "--" geeft aan dat zelfs handmatig programmeren niet mogelijk is met de tool. Verder staat hier "+" voor ondersteuning en "++" voor zeer goede ondersteuning.

BEA Workshop for Weblogic[22] is een complete ontwikkelomgeving gebaseerd op Eclipse. Hierin worden alle functies van Beehive zowel grafisch als met wizards ondersteund. Daarnaast biedt deze ontwikkelomgeving ondersteuning voor Struts 1, JSF en EJB 3. Omdat het gebaseerd is op Eclipse kan er ook in ontwikkeld worden aan Spring en Struts 2 maar hiervoor worden geen grafische mogelijkheden en wizards geboden.

De Spring IDE[23] is een plugin voor Eclipse. Deze plugin biedt ondersteuning van het grafisch weergeven van de Spring configuratie en via wizard instellen van de configuratie. Deze plugin biedt geen ondersteuning op code niveau. Deze plugin is gratis.

MyEclipse[24] is net als BEA Workshop een complete ontwikkelomgeving gebaseerd op Eclipse. Deze ontwikkelomgevind biedt afhankelijk van de versie (Standaard of Professional) een grote lading aan functionaliteit. Onder andere biedt deze ondersteuning voor JSF, Struts 1, Spring en EJB 3. Daarnaast zijn er nog tools voor bijvoorbeeld UML[25] modelleren. Als er onder de gebruikers een grote vraag is naar Struts 2 ondersteuning zal dit ook worden gerealiseerd.
MyEclipse is niet gratis, de kosten zijn respectievelijk $ 29,95 voor de Standaard versie en $ 49,95 voor de Professional versie per jaar per licentie.

Exadel Studio[26] is ook een complete ontwikkelomgeving gebaseerd op Eclipse.
Exadel Studio zal in de loop van 2007 worden vervangen door een programma van JBoss. Daarmee kan het zijn dat er nieuwe functionaliteit geboden gaat worden, bijvoorbeeld ondersteuning voor Struts 2. Momenteel biedt Exadel Studio ondersteuning voor JSF, Struts 1, Spring, Hibernate, Facelets 1.0 en Struts Shale.

Uit dit onderdeel blijkt dat er voor Beehive en Spring goede tools zijn. Voor Struts 2 zijn deze er niet. Dit kan bij projecten een enorm nadeel zijn omdat een goede ontwikkelomgeving gewenst is.

[22] Zie voor informatie: http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/workshop.
[23] Zie voor informatie: http://springide.org/project.
[24] Zie voor informatie: http://www.myeclipseide.com/.
[25] Zie voor informatie: http://nl.wikipedia.org/wiki/Unified_Modeling_Language.
[26] Zie voor informatie: http://www.exadel.com/web/portal/products/ExadelStudioPro.

8. Conclusie

Het doel van dit onderzoek was het vergelijken van de drie frameworks op het gebied van configuratie, code, functionaliteit, flexibiliteit, performance, testbaarheid en toolondersteuning. In de volgende tabel staat het resultaat van dit onderzoek weergegeven.

Onderdeel (wegingsfactor)	**Beehive**	**Spring**	**Struts 2**
Configuratie (3)	+	-	++
Code (3)	-	+	++
Functionaliteit (1)	+	- / +	+
Flexibiliteit (2)	-	+	++
Performance (1)	+	++	-
Testbaarheid (2)	-	++	++
Toolondersteuning (4)	++	++	-
Totaal	- / +	+	+ / ++
Bruikbaar in project	+	+	-

Er kan niet worden geconcludeerd dat één framework het beste gekozen kan worden. Alle frameworks hun specifieke voor en nadelen.

Over het algemeen komt Struts 2 wel beste uit de bus, zeker op technisch vlak. Daarentegen schiet de toolondersteuning en performance bij Struts 2 te kort, toolondersteuning is binnen een project essentieel. Door het feit van de ontbrekende toolondersteuning en omdat Struts 2 pas recent een stabiele versie heeft opgeleverd is het nog niet verstandig om dit framework in een project voor een klant te gebruiken. Naarmate het framework stabieler wordt en de toolondersteuning groeit, zal dit framework ook zeker in een keuze meegenomen moeten worden. Of en zo ja wanneer de toolondersteuning beschikbaar zal zijn is niet aan te geven. Zodra er veel mensen vragen om deze tools zullen ze gemaakt worden.

Spring schiet op technisch gebied wat te kort, is redelijk complex. Maar deze scoort op toolondersteuning en performance heel hoog. Doordat dit framework zich in de loop van de tijd al heeft bewezen en omdat er een goede toolondersteuning is, kan dit framework goed worden ingezet voor een project. Maar naarmate Struts 2 in volwassenheid groeit is dit een serieuze concurrent voor Spring MVC.

Beehive springt er alleen op het onderdeel toolondersteuning uit. Verder komt het framework in vergelijking tot Spring MVC en Struts 2 in de breedte te kort. Door de beschikbare toolondersteuning en omdat ook dt framework zijn stabiliteit heeft bewezen kan dit framework ook goed in een project worden gebruikt.

De conclusie is dat per situatie aan de hand van bovenstaande tabel bekeken kan worden welk framework het meest geschikt is. Een ieder kan natuurlijk ook zijn eigen wegingsfactoren toepassen om zo een passend antwoord te krijgen. Als functionaliteit of performance juist heel zwaar weegt kan deze een hogere wegingsfactor worden gegeven (bijvoorbeeld 4) en de andere een lagere. Verder is de conclusie dat het gebruik van Struts 2 in een project nog niet aan te raden is.

Het is aan te bevelen om over een half jaar tot een jaar de frameworks nog een keer te bekijken. Het is dan niet de bedoeling om dit onderzoek opnieuw te doen, maar om bijvoorbeeld de minpunten van elk framework opnieuw te evalueren. Voor Beehive komt dit neer op het opnieuw bekijken van de hoeveelheid code door de annotaties. Voor Spring MVC is het aan te bevelen om de configuratie en de functionaliteiten te bekijken en voor Struts 2 is het van belang de toolondersteuning en performance opnieuw te evalueren.

Een manier om de frameworks nogmaals te onderzoeken is: De wijzigings logboeken raadplegen. Als in deze logboeken niets over de genoemde te onderzoeken punten vermeld staat is de kans groot dat er niets aan die situtatie veranderd is. Deze manier is niet van toepassing op het onderdeel toolondersteuning. Om te bepalen welke toolondersteuning er is zal er op internet

gezocht moeten worden, ook staat er op de website van het framework vaak informatie over beschikbare tools.

Het is ook aan te bevelen om over een half jaar tot een jaar het vooronderzoek de punten uit het vooronderzoek nogmaals te evalueren. Het aantal boeken en informatie op internet kan (en zal) veranderen, net als het aanbod en de vraag op de arbeidsmarkt. Hiermee kan onder andere opnieuw bekeken worden of het framework toekomst heeft. Hiermee kan tevens bekeken worden of deze drie frameworks nog steeds de beste keuze zijn. Het is namelijk ook mogelijk dat er nieuwe frameworks ontstaan zijn of dat de andere frameworks uit het vooronderzoek beter dan voorheen scoren.

9. Evaluatie

De eerste doelstelling van deze afstudeerstage was voor Capgemini: Inzicht krijgen in de werking van de verschillende frameworks, om zo aan klanten een onderbouwde aanbeveling te kunnen geven over het te gebruiken framework.

Ik denk dat deze doelstelling zeker gehaald is. Uit het onderzoek komt duidelijk naar voren hoe de frameworks werken. In de vergelijking staat per onderdeel een argumentatie en een score voor elk framework. Door eigen wegingsfactoren toe te passen is het mogelijk om een voor de klantsituatie specifieke aanbeveling te geven.

Mijn eigen doelstelling was: Kennis maken met een aantal nieuwe frameworks om zo mijn algemene kennis te verdiepen. Een volgende doelstelling was: Ervaring op doen met het uitvoeren van een onderzoek. Verder had ik de doelstelling: Mijn kennis en ervaring met plannen te vergroten.

Als ik mijn eigen doelstellingen ga evalueren is mijn eerste conclusie dat het voor mij erg leerzaam is geweest. Ik ben er achter gekomen dat ik het programmeren van software leuker vind dan het schrijven van documenten. Dit komt denk ik mede door het feit dat ik programmeren beter kan. Als ik wat ervaring en oefening heb met het schrijven van de documenten zou het me minder moeite en tijd kosten. Daardoor zou ik het schrijven daarvan ook leuker vinden. Het schrijven van documenten is een essentieel onderdeel van het werk, maar ik zal dit (zeker voorlopig) niet als hoofd werkzaamheid willen doen.

Doordat het schrijven van het onderzoeksplan langer duurde dan verwacht ben ik in dat opzicht uitgelopen op de planning. Hierdoor had ik minder tijd over voor het schrijven van de scriptie, maar omdat het onderzoeksrapport letterlijk in de scriptie kon worden opgenomen kostte dat minder tijd dan gepland. Ik denk dat ik een goede planning had gemaakt. In het begin van de planning ging het sneller dan ik had gepland, en aan het eind liep zoals gezegd het onderzoeksrapport wat uit. Ik denk dat ik nu wel weer iets meer ervaring met goed plannen heb.

Mijn algemene kennis heb ik naar mijn mening verdiept en zeker mijn kennis over J2EE webapplicaties.

Voor deze afstudeerstage had ik nog nooit een echt onderzoek uitgevoerd. Tijdens dit onderzoek heb ik gemerkt dat dit mij toch niet erg ligt. Het maken van software vind ik interessanter dan het onderzoeken hiervan en het uitvinden van waarom het werkt zoals het werkt.

Tijdens dit afstudeerproject heb ik ook geleerd dat ik het reviewen van documenten van tevoren moet plannen. Het gebeurde regelmatig dat ik op een onderdeel vastliep in een document en daarom een review nodig had. Maar omdat de begeleiders hun eigen werkzaamheden hebben is dat niet altijd direct mogelijk. Om dit op te lossen zou ik van te voren een aantal momenten in moeten plannen waarop gereviewed kan worden. Dit zou bijvoorbeeld één keer in de week kunnen zijn, zodat de begeleiders hier ook rekening mee kunnen houden en ik van te voren weet wanneer er wel of geen mogelijkheid voor een review is.

Al met al denk ik dat dit afstudeerproject geslaagd is en dat Capgemini een bruikbaar document heeft voor gebruik bij de keuze van een framework.

10. Bronvermelding

10.1. Boeken

Kanchanavallu, S., & Kunnal, M., (2005), *Pro Apache Beehive*, Apress, 1-590595-015-7

Kodali, R., & Wetherbee, J. (2006), *Beginning EJB 3 Application Development*: *From Novice to Professional*, Apress, 1-59059-671-4

Ladd, S., & Davison, D., & Devijver, S., et al. (2006), *Expert Spring MVC and Web Flow*, Apress, 1-59059-584-X

10.2. Documenten

10.3. Internetbronnen

Bea (NetUI), http://e-docs.bea.com/workshop/docs81/doc/en/core/index.html
Beehive, http://beehive.apache.org/
Spring, http://www.springframework.org/
Spring MVC Reference, http://www.springframework.org/docs/reference/mvc.html
Struts 1 vs. Struts 2, http://www.java-samples.com/showtutorial.php?tutorialid=200
Struts 2, http://struts.apache.org/2.x/

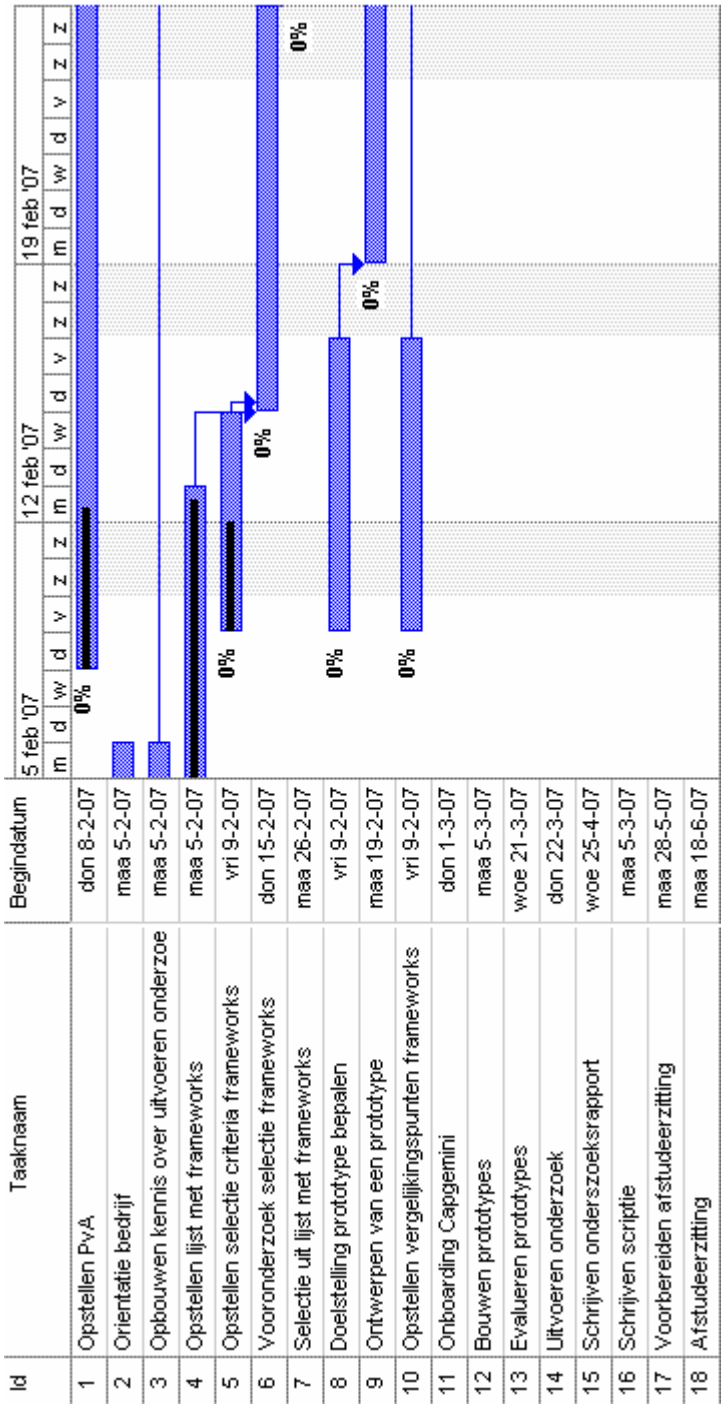
Bijlage 1 – Lijst met kandidaat frameworks

Naam	Url	Versie	Word nog ontwikkeld
JSF	http://java.sun.com/javaee/javaserverfaces/	1.2	Ja
Spring MVC	http://www.springframework.org/	2.0.5 en 1.2.8	Ja
Struts	http://struts.apache.org/	1.35 en 2.0.6	1.35 Nee 2.01 Ja
Tapestry	http://tapestry.apache.org/	4.0.2 (stable) en 4.1.1	Ja
WebWork	http://www.opensymphony.com/webwork/	2.2.4	Nee is Struts 2
Wicket	http://wicket.sourceforge.net/	1.2.4	Ja
Apache Beehive	http://beehive.apache.org/	1.0.2	Ja
Anvil	http://njet.org/		
Backbase	http://www.backbase.com		
Chrysalis	http://chrysalis.sourceforge.net/		
Cocoon	http://cocoon.apache.org/		
Echo	http://www.nextapp.com/products/echo/		
Jaffa	http://jaffa.sourceforge.net/		
JAT	http://www.myjavaserver.com/~jatopensource/		
Jucas	http://jucas.sourceforge.net/		
Maverick	http://mav.sourceforge.net/		
Millstone	http://www.millstone.org		
MyFaces	http://www.myfaces.org/		
RIFE	http://rife.dev.java.net		
Sofia	http://www.salmonllc.com/website/Jsp/vanity/Sofia.jsp		
Swinglets	http://www.javelinsoft.com/swinglets/		
Swingweb	http://swingweb.sourceforge.net		
Verge	http://www.inversoft.com/index.html		
Vraptor	http://www.vraptor.org/		
wingS	http://www.j-wings.org		

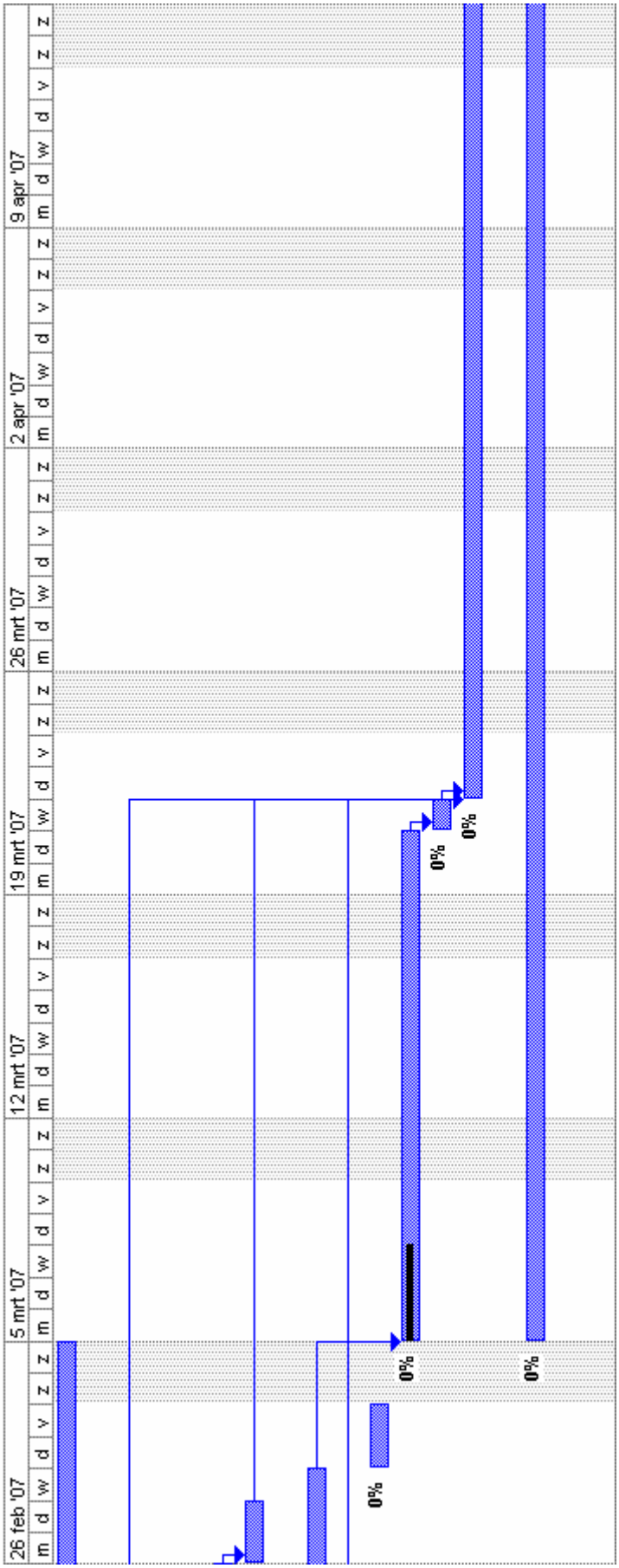
Bijlage 2 – Lijst selectie negen frameworks

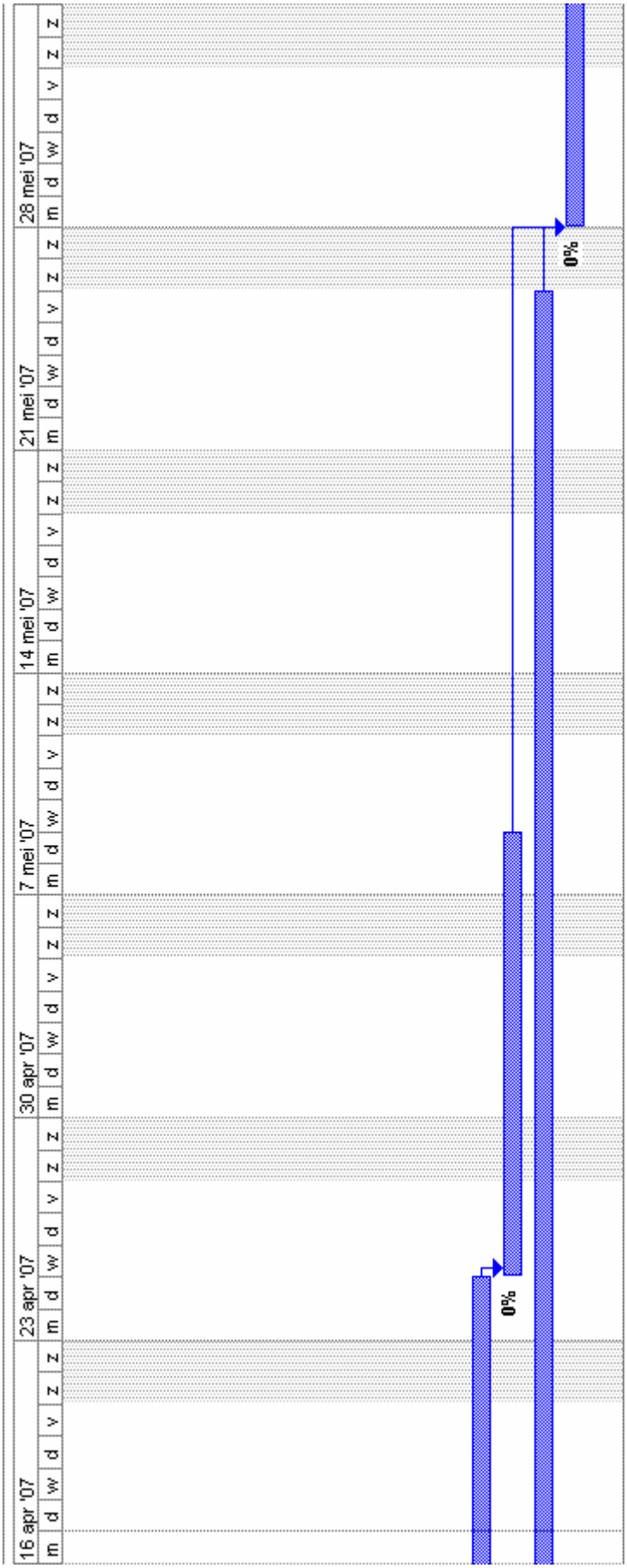
Naam	Url	Versie
Apache Beehive	http://beehive.apache.org/	1.0.2
Apache Cocoon	http://cocoon.apache.org/	2.1.10
JSF	http://java.sun.com/javaee/javaserverfaces/	1.2
Maverick	http://mav.sourceforge.net/	2.2.4
MyFaces	http://www.myfaces.org/	1.1.5
Spring MVC	http://www.springframework.org/	2.0.5
Struts Action 2	http://struts.apache.org/2.x/	2.0.6
Tapestry	http://tapestry.apache.org/	4.0.2
Wicket	http://wicket.sourceforge.net/	1.2.6

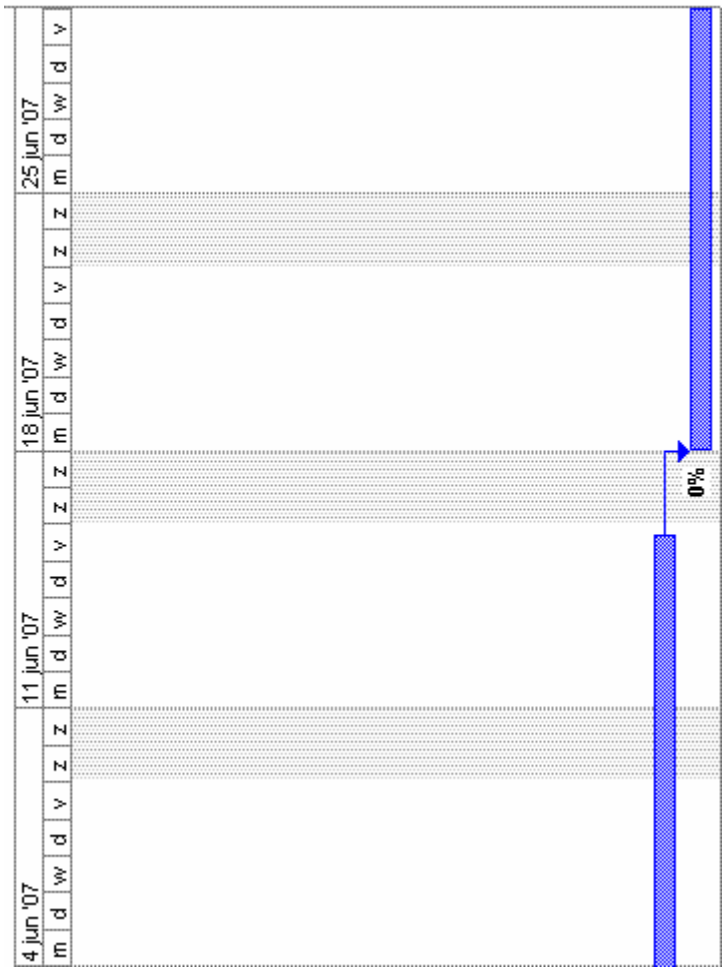
Bijlage 3 – Planning per activiteit



Id	Taaknaam	Begindatum
1	Opstellen PvA	don 8-2-07
2	Orientatie bedrijf	maa 5-2-07
3	Opbouwen kennis over uitvoeren onderzoe	maa 5-2-07
4	Opstellen lijst met frameworks	maa 5-2-07
5	Opstellen selectie criteria frameworks	vri 9-2-07
6	Vooronderzoek selectie frameworks	don 15-2-07
7	Selectie uit lijst met frameworks	maa 26-2-07
8	Doelstelling prototype bepalen	vri 9-2-07
9	Ontwerpen van een prototype	maa 19-2-07
10	Opstellen vergelijkingspunten frameworks	vri 9-2-07
11	Onboarding Capgemini	don 1-3-07
12	Bouwen prototypes	maa 5-3-07
13	Evalueren prototypes	woe 21-3-07
14	Uitvoeren onderzoek	don 22-3-07
15	Schrijven onderszoeksrapport	woe 25-4-07
16	Schrijven scriptie	maa 5-3-07
17	Voorbereiden afstudeerzitting	maa 28-5-07
18	Afstudeerzitting	maa 18-6-07


26 feb '07
5 mrt '07
12 mrt '07
19 mrt '07
26 mrt '07
2 apr '07
9 apr '07
0%
0%
0%
0%
0%


16 apr '07
23 apr '07
30 apr '07
7 mei '07
14 mei '07
21 mei '07
28 mei '07
0%
0%


4 jun '07
m d w d v z z
11 jun '07
m d w d v z z
18 jun '07
m d w d v z z
25 jun '07
m d w d v
0%

Bijlage 4 – Beehive code en configuratie

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
        version="2.4">

    <display-name>Beehive NetUI Web Application</display-name>

    <!-- Filter to perform Page Flow operations when JSPs are hit directly. -->
    <filter>
        <filter-name>PageFlowJspFilter</filter-name>
        <filter-class>org.apache.beehive.netui.pageflow.PageFlowJspFilter</filter-class>
    </filter>

    <filter>
        <filter-name>PageFlowForbiddenFilter</filter-name>
        <filter-class>org.apache.beehive.netui.pageflow.PageFlowForbiddenFilter</filter-class>
        <init-param>
            <param-name>response-code</param-name>
            <param-value>404</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>PageFlowJspFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>

    <!-- Configuration/init listener for NetUI framework -->
    <listener>
        <listener-class>org.apache.beehive.netui.pageflow.PageFlowContextListener</listener-class>
    </listener>

    <!-- Listener to create NetUI framework HttpSession mutex objects -->
    <listener>
        <listener-class>org.apache.beehive.netui.pageflow.HttpSessionMutexListener</listener-class>
    </listener>

    <!-- NetUI Servlet Configuration -->
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.beehive.netui.pageflow.PageFlowActionServlet</servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/_pageflow/struts-config.xml</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>

    <!-- Struts Action Servlet Mappings -->
    <!-- Note that because Struts takes the *last* mapping here as the extension to add to
         actions posted from forms, we must have *.do come after *.jpf (etc.). -->
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.jpf</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Controller.java

```
@Jpf.Controller(
  simpleActions={
    @Jpf.SimpleAction( name="begin", path="index.jsp"),
    @Jpf.SimpleAction( name="error", path="error.jsp")
  }
)
public class Controller extends PageFlowController {
  private transient final Logger logger = Logger.getLogger(getClass());
  private BookSession bookSession;
  private CustomerSession customerSession;
  private int id;
  private int bookid;
  private int customerid;
  private static final String SUCCESS = "success";
  private static final String ERROR = "error";
  private static final String PARAMERROR = "paramerror";
  private static final String BOOKSINPUT = "booksInput";

  @Jpf.Action(
    forwards = {
      @Jpf.Forward(
        name = SUCCESS,
        path="books.jsp",
        actionOutputs={ @Jpf.ActionOutput(name=BOOKSINPUT, type=BookEntity[].class, required=true) }
      )
    }
  )
  public Forward books() {
    logger.info("books()");
    Forward forward = new Forward(SUCCESS);

    try {
      prepare();
      BookEntity[] books = bookSession.retrieveAll();
      forward.addActionOutput(BOOKSINPUT, books);
    }
    catch(Exception e) {
      forward = new Forward(ERROR);
    }
    return forward;
  }

  @Jpf.Action(
    forwards = {
      @Jpf.Forward(
        name = SUCCESS,
        path="book.jsp",
        actionOutputs={ @Jpf.ActionOutput(name="bookInput", type=BookEntity.class, required=true) }
      ),
      @Jpf.Forward( name = PARAMERROR, action="books", redirect=true ),
      @Jpf.Forward( name = ERROR, action=ERROR, redirect=true )
    }
  )
  public Forward book() {
    logger.info("book()");
    Forward forward;
    try {
      prepare();
      int id = getId();
      if(id != 0) {
        forward           = new Forward(SUCCESS);
        BookEntity book = bookSession.retrieve(id);
        forward.addActionOutput("bookInput", book);
      }
      else { forward = new Forward(PARAMERROR); }
    }
    catch (NamingException e) { forward = new Forward(ERROR); }
    catch (BookNotFoundException e) { forward = new Forward(PARAMERROR); }
    return forward;
  }
```

```
@Jpf.Action(
  forwards = {
    @Jpf.Forward(
      name = SUCCESS,
      path="lends.jsp",
      actionOutputs={ @Jpf.ActionOutput( name=BOOKSINPUT, type=BookEntity[].class, required=true ) }
    )
  }
)
public Forward lends() {
  logger.info("lends()");
  Forward forward = new Forward(SUCCESS);
  try {
    prepare();
    BookEntity[] books = bookSession.retrieveLend();
    forward.addActionOutput(BOOKSINPUT, books);
    return forward;
  }
  catch(Exception e) { return null; }
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, path = "lendForm.jsp" ),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true )
  }
)
public Forward lend() {
  logger.info("lend()");
  Forward forward;
  try {
    prepare();
    LendBean bean = new LendBean();
    if(customerid!=0) { bean.setCustomerid(customerid); }
    if(bookid!=0) { bean.setBookid(bookid); }
    forward = new Forward(SUCCESS,bean);
  }
  catch (NamingException e) { forward = new Forward(ERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, action = "lends", redirect=true),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  }
)

public Forward saveLend(LendBean form){
  logger.info("saveLend("+form+")");
  Forward forward;
  try {
    prepare();
    bookSession.lend(form.getBookid(), form.getCustomerid());
    forward = new Forward(SUCCESS);
  }
  catch (Exception e) { forward = new Forward(ERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, path = "unlendForm.jsp" ),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true )
  }
)
public Forward unlend() {
  logger.info("unlend()");
  Forward forward;
  try {
    prepare();
    LendBean bean = new LendBean();
    if(bookid!=0) { bean.setBookid(bookid); }
    forward = new Forward(SUCCESS,bean);
  }
```

```
  catch (NamingException e) { forward = new Forward(ERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, action = "lends", redirect=true ),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true )
  }
)
public Forward saveUnlend(LendBean form) {
  logger.info("saveUnlend("+form+")");
  try {
    prepare();
    bookSession.unlend(form.getBookid());
    return new Forward(SUCCESS);
  }
  catch (NamingException e) { return new Forward(ERROR); }
  catch (BookNotFoundException e) { return new Forward(ERROR); }
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, action = "books", redirect=true),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  },
  validatableProperties={
    @Jpf.ValidatableProperty( propertyName="title", displayName="Field", validateRequired=@Jpf.ValidateRequired() ),
    @Jpf.ValidatableProperty( propertyName="author", displayName="Field", validateRequired=@Jpf.ValidateRequired() )
  },
  validationErrorForward=@Jpf.Forward(name="fail", path="bookForm.jsp")
)
public Forward saveBook(BookEntity form) {
  logger.info("saveBook("+form+")");
  try {
    prepare();
    bookSession.save(form);
    return new Forward(SUCCESS);
  }
  catch (NamingException e) { return new Forward(ERROR); }
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, path = "bookForm.jsp"),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  }
)
public Forward addBook() {
  logger.info("addBook()");
  try {
    prepare();
    BookEntity bookentity = bookSession.create();
    Forward forward = new Forward(SUCCESS,bookentity);
    forward.addActionOutput("id", bookentity.getId());
    return forward;
  }
  catch (NamingException e) {
    return new Forward(ERROR);
  }
}
```

```
@Jpf.Action(
  forwards = {
    @Jpf.Forward(name = SUCCESS, action = "books", redirect=true),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  }
)
public Forward deleteBook() {
  logger.info("deleteBook()");
  Forward forward;
  try {
    prepare();
    int id = getId();
    if(id!=0) {
      bookSession.delete(id);
      forward = new Forward(SUCCESS);
    }
    else { forward = new Forward(ERROR); }
  }
  catch (BookNotFoundException e) { forward = new Forward(SUCCESS); }
  catch(Exception e) { forward = new Forward(ERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, path = "bookForm.jsp" ),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true )
    }
  )

public Forward editBook() {
  logger.info("editBook()");
  Forward forward;
  try {
    prepare();
    int id = getId();
    if(id!=0) { forward = new Forward(SUCCESS,bookSession.retrieve(id)); }
    else { forward = new Forward(ERROR); }
  }
  catch(Exception e) { forward = new Forward(ERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward(
      name = SUCCESS,
      path="customers.jsp",
        actionOutputs={ @Jpf.ActionOutput( name="customersInput", type=CustomerEntity[].class, required=true ) }
    )
  }
)
public Forward customers() {
  logger.info("customers()");
  Forward forward = new Forward(SUCCESS);
  try {
    prepare();
    CustomerEntity[]customers = customerSession.retrieveAll();
    forward.addActionOutput("customersInput", customers);
    return forward;
  }
  catch(Exception e) {
    e.printStackTrace();
   return null;
  }
}
```

```
@Jpf.Action(
  forwards = {
    @Jpf.Forward(
      name = SUCCESS,
      path="customer.jsp",
      actionOutputs={ @Jpf.ActionOutput( name="customerInput",  type=CustomerEntity.class,  required=true ) }
    ),
    @Jpf.Forward( name = PARAMERROR, action="customers", redirect=true ),
    @Jpf.Forward( name = ERROR, action=ERROR, redirect=true )
  }
)
public Forward customer() {
  logger.info("customer()");
  Forward forward;
  try {
    prepare();
    int id = getId();
    if(id != 0) {
      forward = new Forward(SUCCESS);
      CustomerEntity cust = customerSession.retrieve(id);
      forward.addActionOutput("customerInput", cust);
    }
    else { forward = new Forward(PARAMERROR); }
  }
  catch (NamingException e) { forward = new Forward(ERROR); }
  catch (CustomerNotFoundException e) { forward = new Forward(PARAMERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, path="customerForm.jsp" ),
    @Jpf.Forward( name = ERROR, action=ERROR, redirect=true )
  }
)
public Forward addCustomer() {
  logger.info("addCustomer()");
  try {
    prepare();
    CustomerEntity ce = customerSession.create();
    Forward f =  new Forward(SUCCESS,ce);
    f.addActionOutput("id",ce.getId());
    return f;
  }
  catch (NamingException e) { return new Forward(ERROR); }
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, path = "customerForm.jsp"),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  }
)
public Forward editCustomer() {
  logger.info("editCustomer()");
  Forward forward;
  try {
    prepare();
    int id = getId();
    if(id!=0) {
      forward = new Forward(SUCCESS,customerSession.retrieve(id));
    }
    else {
      forward = new Forward(ERROR);
    }
  }
  catch(Exception e) { forward = new Forward(ERROR); }
  return forward;
}
```

```
@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, action = "customers", redirect=true),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  }
)
public Forward deleteCustomer() {
  logger.info("deleteCustomer()");
  Forward forward;
  try {
    prepare();
    int id = getId();
    if(id!=0) {
      customerSession.delete(id);
      forward = new Forward(SUCCESS);
    }
    else { forward = new Forward(ERROR); }
  }
  catch (CustomerNotFoundException e) { forward = new Forward(SUCCESS); }
  catch(Exception e) { forward = new Forward(ERROR); }
  return forward;
}

@Jpf.Action(
  forwards = {
    @Jpf.Forward( name = SUCCESS, action = "customers", redirect=true),
    @Jpf.Forward( name = ERROR, action = ERROR, redirect=true)
  },
  validatableProperties={
    @Jpf.ValidatableProperty( propertyName="firstName", displayName="Field", validateRequired=@Jpf.ValidateRequired() ),
    @Jpf.ValidatableProperty( propertyName="lastName", displayName="Field", validateRequired=@Jpf.ValidateRequired() ),
    @Jpf.ValidatableProperty( propertyName="birthDayDate", displayName="Field", validateRequired=@Jpf.ValidateRequired() )
  },
  validationErrorForward=@Jpf.Forward(name="fail", path="customerForm.jsp")
)
public Forward saveCustomer(CustomerEntity form) {
  logger.info("saveCustomer("+form+")");
  try {
    prepare();
    customerSession.save(form);
    return new Forward(SUCCESS);
  }
  catch (NamingException e) { return new Forward(ERROR); }
}

public void prepare() throws NamingException {
  logger.info("prepare()");
  String factory = "org.jnp.interfaces.NamingContextFactory";
  String url = "jnp://localhost:1099";
  Properties props = new Properties();
  props.setProperty(Context.INITIAL_CONTEXT_FACTORY, factory);
  props.setProperty(Context.PROVIDER_URL, url);
  InitialContext ctx = new InitialContext(props);
  Object ref = ctx.lookup("BookSessionBean/remote");
  bookSession = (BookSession) javax.rmi.PortableRemoteObject.narrow(ref,BookSession.class);

  Object ref2 = ctx.lookup("CustomerSessionBean/remote");
  customerSession = (CustomerSession) javax.rmi.PortableRemoteObject.narrow(ref2,CustomerSession.class);

  HttpServletRequest req = getRequest();
  String ids = req.getParameter("id");
  if(ids!="") {
    try {
      int id = Integer.parseInt(ids);
      setId(id);
    }
    catch(NumberFormatException e) { setId(0); }
  }
  else { setId(0); }
```

```
    String bookids = req.getParameter("bookid");
    if(ids!="") {
      try {
        int bookid = Integer.parseInt(bookids);
        setBookid(bookid);
      }
      catch(NumberFormatException e) { setBookid(0); }
    }
    else { setBookid(0); }

    String customerids = req.getParameter("customerid");
    if(ids!="") {
      try {
        int customerid = Integer.parseInt(customerids);
        setCustomerid(customerid);
      }
      catch(NumberFormatException e) { setCustomerid(0); }
    }
    else { setCustomerid(0); }
  }

  public int getId() { return id; }
  public void setId(int id) { this.id=id; }

  public int getBookid() { return bookid; }
  public void setBookid(int bookid) { this.bookid = bookid; }

  public int getCustomerid() { return customerid; }
  public void setCustomerid(int customerid) { this.customerid = customerid; }

  public BookSession getBookSession() { return bookSession; }
  public void setBookSession(BookSession bs) { this.bookSession = bs; }

  public CustomerSession getCustomerSession() { return customerSession; }
  public void setCustomerSession(CustomerSession cs) { this.customerSession = cs; }
}
```

LendBean.java

```
package formBean;

public class LendBean {
  private int bookid;
  private int customerid;

  public int getBookid() { return bookid; }
  public void setBookid(int bookid) { this.bookid = bookid; }

  public int getCustomerid() { return customerid; }
  public void setCustomerid(int customerid) { this.customerid = customerid; }
}
```

Bijlage 5 – Beehive jsp pagina's

book.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <title>Book</title>
  </head>
  <netui:body>
    <h2>Book details</h2>
    <table>
      <tr>
        <th align="left">Id:</th>
        <td>${pageInput.bookInput.id}</td>
      </tr>
      <tr>
        <th align="left">Title:</th>
        <td>${pageInput.bookInput.title}</td>
      </tr>
      <tr>
        <th align="left">Author:</th>
        <td>${pageInput.bookInput.author}</td>
      </tr>
      <c:choose>
        <c:when test="${!pageInput.bookInput.available}">
          <tr>
            <th align="left">Available:</th>
            <td>No</td>
          </tr>
          <tr>
            <th align="left">Lend to:</th>
            <td>${pageInput.bookInput.lendTo.fullName}</td>
          </tr>
          <tr>
            <th align="left"></th>
            <td><netui:anchor action="unlend">
                Return book<netui:parameter name="bookid" value="${pageInput.bookInput.id}"/>
              </netui:anchor>
            </td>
          </tr>
        </c:when>
        <c:otherwise>
          <tr>
            <th align="left">Available:</th>
            <td>Yes</td>
          </tr>
          <tr>
            <th align="left"></th>
            <td><netui:anchor action="lend">
                Lend book
                <netui:parameter name="bookid" value="${pageInput.bookInput.id}"/>
              </netui:anchor>
            </td>
          </tr>
        </c:otherwise>
      </c:choose>
    </table>
    <netui:anchor action="books">Back to books</netui:anchor>
  </netui:body>
</netui:html>
```

bookForm.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <c:choose>
      <c:when test="${pageInput.id == 0}">
        <title>Add new book</title>
      </c:when>
      <c:otherwise>
        <title>Edit book</title>
      </c:otherwise>
    </c:choose>
  </head>
  <netui:body>
    <c:choose>
      <c:when test="${pageInput.id == 0}">
        <h2>Add new book</h2>
      </c:when>
      <c:otherwise>
        <h2>Edit book</h2>
      </c:otherwise>
    </c:choose>
    <netui:form action="saveBook" tagId="bookForm">
      <table>
        <tr>
          <th align="left">Title:</th>
          <td><netui:textBox tagId="title" dataSource="actionForm.title" /></td>
          <td><netui:error key="title"></netui:error></td>
        </tr>
        <tr>
          <th align="left">Author:</th>
          <td><netui:textBox tagId="author" dataSource="actionForm.author" /></td>
          <td><netui:error key="author"></netui:error></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><netui:button type="submit" value="Submit" /></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><netui:button type="submit" action="books" styleClass="text" value="Cancel" /></td>
        </tr>
      </table>
      <netui:hidden tagId="id" dataSource="actionForm.id"/>
    </netui:form>
  </netui:body>
</netui:html>
```

books.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <title>Books</title>
  </head>
  <netui:body>
    <h2>Books in the library</h2>
    <netui-data:dataGrid name="BooksGrid" dataSource="pageInput.booksInput" border="1">
      <netui-data:configurePager pageAction="books"/>
      <netui-data:header>
        <netui-data:headerCell headerText="Id" cellAlign="left"/>
        <netui-data:headerCell headerText="Title" cellAlign="left"/>
        <netui-data:headerCell headerText="Author" cellAlign="left"/>
        <netui-data:headerCell headerText="Available" cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
      </netui-data:header>
      <netui-data:rows>
        <netui-data:spanCell value="${container.item.id}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.title}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.author}"></netui-data:spanCell>
        <c:choose>
          <c:when test="${!container.item.available}">
            <netui-data:spanCell value="No"></netui-data:spanCell>
          </c:when>
          <c:otherwise>
            <netui-data:spanCell value="Yes"></netui-data:spanCell>
          </c:otherwise>
        </c:choose>
        <netui-data:anchorCell value="Edit" action="editBook">
          <netui:parameter name="id" value="${container.item.id}"/>
        </netui-data:anchorCell>
        <netui-data:anchorCell value="Delete" action="deleteBook">
          <netui:parameter name="id" value="${container.item.id}"/>
        </netui-data:anchorCell>
        <netui-data:anchorCell value="Details" action="book">
          <netui:parameter name="id" value="${container.item.id}"/>
        </netui-data:anchorCell>
        <c:choose>
          <c:when test="${!container.item.available}">
            <netui-data:anchorCell value="Return book" action="unlend">
              <netui:parameter name="bookid" value="${container.item.id}"/>
            </netui-data:anchorCell>
          </c:when>
          <c:otherwise>
            <netui-data:anchorCell value="Lend book" action="lend">
              <netui:parameter name="bookid" value="${container.item.id}"/>
            </netui-data:anchorCell>
          </c:otherwise>
        </c:choose>
      </netui-data:rows>
    </netui-data:dataGrid>
    <netui:anchor action="addBook">Add a book</netui:anchor>
    <br />
    <netui:anchor action="begin">Back to main</netui:anchor>
  </netui:body>
</netui:html>
```

customer.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <title>Customer</title>
  </head>
  <netui:body>
    <h2>Customer details</h2>
    <table>
      <tr>
        <th align="left">Id:</th>
        <td>${pageInput.customerInput.id}</td>
      </tr>
      <tr>
        <th align="left">Firstname:</th>
        <td>${pageInput.customerInput.firstName}</td>
      </tr>
      <tr>
        <th align="left">Lastname:</th>
        <td>${pageInput.customerInput.lastName}</td>
      </tr>
      <tr>
        <th align="left">Birthday:</th>
        <td>
          <netui:label value="${pageInput.customerInput.birthDayDate}">
            <netui:formatDate pattern="dd-MM-yyyy" />
          </netui:label>
        </td>
      </tr>
    </table>
    <h2>Lend books</h2>
    <netui-data:dataGrid name="LendBooksGrid" dataSource="pageInput.customerInput.books" border="1">
      <netui-data:configurePager pageAction="customer"/>
      <netui-data:header>
        <netui-data:headerCell headerText="Id" cellAlign="left"/>
        <netui-data:headerCell headerText="Title" cellAlign="left"/>
        <netui-data:headerCell headerText="Author" cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
      </netui-data:header>
      <netui-data:rows>
        <netui-data:spanCell value="${container.item.id}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.title}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.author}"></netui-data:spanCell>
        <netui-data:anchorCell value="Return Book" action="unlend">
          <netui:parameter name="bookid" value="${container.item.id}"/>
        </netui-data:anchorCell>
      </netui-data:rows>
    </netui-data:dataGrid>
    <netui:anchor action="customers">Back to customers</netui:anchor>
  </netui:body>
</netui:html>
```

customerForm.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <c:choose>
      <c:when test="${pageInput.id == 0}">
        <title>Add new customer</title>
      </c:when>
      <c:otherwise>
        <title>Edit customer</title>
      </c:otherwise>
    </c:choose>
  </head>
  <netui:body>
    <c:choose>
      <c:when test="${param.id == 0 || empty param.id}">
        <h2>Add new customer</h2>
      </c:when>
      <c:otherwise>
        <h2>Edit customer</h2>
      </c:otherwise>
    </c:choose>
    <netui:form action="saveCustomer" tagId="customerForm">
      <table>
        <tr>
          <th align="left">Firstname:</th>
          <td><netui:textBox tagId="firstName" dataSource="actionForm.firstName" size="30" styleClass="text"/></td>
          <td><font color="red"><netui:error key="firstName" /></font></td>
        </tr>
        <tr>
          <th align="left">Lastname:</th>
          <td><netui:textBox tagId="lastName" dataSource="actionForm.lastName" size="30" styleClass="text"/></td>
          <td><font color="red"><netui:error key="lastName" /></front</td>
        </tr>
        <tr>
          <th align="left">Birthday:</th>
          <td>
            <netui:textBox tagId="birthDayDate" dataSource="actionForm.birthDayDate" size="10" styleClass="text">
              <netui:formatDate pattern="dd-MM-yyyy" />
            </netui:textBox>
          </td>
          <td><font color="red"><netui:error key="birthDayDate" /></font></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><netui:button type="submit" styleClass="text" value="Submit" /></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><netui:button type="submit" action="customers" styleClass="text" value="Cancel" /></td>
        </tr>
      </table>
      <netui:hidden tagId="id" dataSource="actionForm.id"/>
    </netui:form>
  </netui:body>
</netui:html>
```

customers.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>

<netui:html>
  <head>
    <netui:base/>
    <title>Customers</title>
  </head>
  <netui:body>
    <h2>Customers in the library</h2>
    <netui-data:dataGrid name="CustomersGrid" dataSource="pageInput.customersInput" border="1">
      <netui-data:configurePager pageAction="customers"/>
      <netui-data:header>
        <netui-data:headerCell headerText="Id" cellAlign="left"/>
        <netui-data:headerCell headerText="Firstname" cellAlign="left"/>
        <netui-data:headerCell headerText="Lastname" cellAlign="left"/>
        <netui-data:headerCell headerText="Birthday" cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
      </netui-data:header>
      <netui-data:rows>
        <netui-data:spanCell value="${container.item.id}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.firstName}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.lastName}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.birthDayDate}">
          <netui:formatDate pattern="dd-MM-yyyy" />
        </netui-data:spanCell>
        <netui-data:anchorCell value="Edit" action="editCustomer">
          <netui:parameter name="id" value="${container.item.id}"/>
        </netui-data:anchorCell>
        <netui-data:anchorCell value="Delete" action="deleteCustomer">
          <netui:parameter name="id" value="${container.item.id}"/>
        </netui-data:anchorCell>
        <netui-data:anchorCell value="Details" action="customer">
          <netui:parameter name="id" value="${container.item.id}"/>
        </netui-data:anchorCell>
      </netui-data:rows>
    </netui-data:dataGrid>
    <netui:anchor action="addCustomer">Add a customer</netui:anchor>
    <br />
    <netui:anchor action="begin">Back to main</netui:anchor>
  </netui:body>
</netui:html>
```

error.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <title>Books</title>
  </head>
  <netui:body>
    <H2>Error!</H2>
  </netui:body>
</netui:html>
```

index.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<netui:html>
  <head>
    <title>Library</title>
    <netui:base/>
  </head>
  <netui:body>
    <h2>Welcome to the mini-library</h2>
    <netui:anchor action="customers">Customers</netui:anchor>
    <br />
    <netui:anchor action="books">Books</netui:anchor>
    <br />
  <netui:anchor action="lends">Lends</netui:anchor>
  </netui:body>
</netui:html>
```

lendForm.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <title>Lend a book</title>
  </head>
  <netui:body>
    <h2>Lend book</h2>
    <netui:form action="saveLend" tagId="lendForm">
      <table>
        <tr>
          <th align="left">Book:</th>
          <td><netui:textBox tagId="bookid" dataSource="actionForm.bookid" /></td>
          <td><netui:error key="book.error.title"></netui:error></td>
        </tr>
        <tr>
          <th align="left">Customer:</th>
          <td><netui:textBox tagId="customerid" dataSource="actionForm.customerid" /></td>
          <td><netui:error key="book.error.author"></netui:error></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><netui:button type="submit" styleClass="text" value="Submit" /></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><netui:button type="submit" action="lends" styleClass="text" value="Cancel" /></td>
        </tr>
      </table>
    </netui:form>
  </netui:body>
</netui:html>
```

lends.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<netui:html>
  <head>
    <netui:base/>
    <title>Lend books</title>
  </head>
  <netui:body>
    <h2>Lend books in the library</h2>
    <netui:anchor action="lend">Lend a book</netui:anchor>
    <netui:anchor action="unlend">Return a book</netui:anchor><br />
    <netui-data:dataGrid name="BooksGrid" dataSource="pageInput.booksInput" border="1">
      <netui-data:configurePager pageAction="lends" />
      <netui-data:header>
        <netui-data:headerCell headerText="Id" cellAlign="left"/>
        <netui-data:headerCell headerText="Title" cellAlign="left"/>
        <netui-data:headerCell headerText="Author" cellAlign="left"/>
        <netui-data:headerCell headerText="Lend To" cellAlign="left"/>
        <netui-data:headerCell headerText=" " cellAlign="left"/>
      </netui-data:header>
      <netui-data:rows>
        <netui-data:spanCell value="${container.item.id}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.title}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.author}"></netui-data:spanCell>
        <netui-data:spanCell value="${container.item.lendTo.fullName}"></netui-data:spanCell>
        <netui-data:anchorCell value="Return Book" action="unlend">
          <netui:parameter name="bookid" value="${container.item.id}"/>
        </netui-data:anchorCell>
      </netui-data:rows>
    </netui-data:dataGrid>
    <netui:anchor action="begin">Back to main</netui:anchor>
  </netui:body>
</netui:html>
```

unlendForm.jsp

```
<%@ page language="java" contentType="text/html;charset=ISO-8859-1"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<netui:html>
  <head>
    <netui:base/>
    <title>Return a book</title>
  </head>
  <netui:body>
     <h2>Return book</h2>
     <netui:form action="saveUnlend" tagId="unlendForm">
       <table>
         <tr>
           <th align="left">Book:</th>
           <td><netui:textBox tagId="bookid" dataSource="actionForm.bookid" /></td>
           <td><netui:error key="book.error.title"></netui:error></td>
         </tr>
         <tr>
           <th align="left"></th> <td></td>
           <td align="right"><netui:button type="submit" value="Submit" /></td>
         </tr>
        <tr>
          <th align="left"></th> <td></td>
          <td align="right"><netui:button type="submit" action="lends" value="Cancel" /></td>
        </tr>
      </table>
    </netui:form>
  </netui:body>
</netui:html>
```

Bijlage 6 – Spring MVC configuratie

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
 version="2.4"
>
  <display-name>PrototypeSpring</display-name>
  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>

  <!-- The Usual Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

springapp-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

<!--  - Application context definition for "springapp" DispatcherServlet.  -->
<beans>
  <bean id="methodNameResolver" class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
    <property name="paramName">
      <value>action</value>
    </property>
  </bean>
  <bean id="propsMethodNameResolver"
    class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver">
    <property name="mappings">
      <value>
        /books.action=list
        /customers.action=list
      </value>
    </property>
  </bean>

  <bean id="BookController" class="controllers.BookController">
    <property name="methodNameResolver" ref="methodNameResolver"/>
    <property name="bs" ref="BookSession" />
  </bean>

  <bean id="BooksController" class="controllers.BookController">
    <property name="methodNameResolver" ref="propsMethodNameResolver"/>
    <property name="bs" ref="BookSession" />
  </bean>

  <bean id="LendsController" class="controllers.LendsController">
    <property name="bs" ref="BookSession" />
  </bean>

  <bean id="CustomersController" class="controllers.CustomerController">
    <property name="methodNameResolver" ref="propsMethodNameResolver"/>
    <property name="cs" ref="CustomerSession" />
  </bean>
```

```
 <bean id="CustomerController" class="controllers.CustomerController">
  <property name="methodNameResolver" ref="methodNameResolver"/>
  <property name="cs" ref="CustomerSession" />
 </bean>

 <bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
   <props>
    <prop key="/book.action">BookController</prop>
    <prop key="/books.action">BooksController</prop>
    <prop key="/bookForm.action">BookFormController</prop>
    <prop key="/lends.action">LendsController</prop>
    <prop key="/customers.action">CustomersController</prop>
    <prop key="/customer.action">CustomerController</prop>
    <prop key="/customerForm.action">CustomerFormController</prop>
    <prop key="/lendForm.action">LendFormController</prop>
    <prop key="/unlendForm.action">UnlendFormController</prop>
   </props>
  </property>
 </bean>

 <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass"><value>org.springframework.web.servlet.view.JstlView</value></property>
  <property name="prefix"><value>/WEB-INF/jsp/</value></property>
  <property name="suffix"><value>.jsp</value></property>
 </bean>

 <bean id="BookSession" class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName" value="BookSessionBean/remote" />
 </bean>

 <bean id="CustomerSession" class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName" value="CustomerSessionBean/remote" />
 </bean>

 <bean id="BookFormController" class="controllers.BookFormController">
  <property name="bs" ref="BookSession" />
  <property name="formView" value="bookForm" />
  <property name="commandName" value="book" />
  <property name="successView" value="redirect:books.action"/>
  <property name="cancelView" value="redirect:books.action"/>
  <property name="validator"><bean class="validators.BookValidator" /></property>
 </bean>

 <bean id="CustomerFormController" class="controllers.CustomerFormController">
  <property name="cs" ref="CustomerSession" />
  <property name="formView" value="customerForm" />
  <property name="commandName" value="customer" />
  <property name="successView" value="redirect:customers.action"/>
  <property name="cancelView" value="redirect:customers.action"/>
  <property name="validator"><bean class="validators.CustomerValidator" /></property>
 </bean>

 <bean id="LendFormController" class="controllers.LendFormController">
  <property name="bs" ref="BookSession" />
  <property name="formView" value="lendForm" />
  <property name="commandName" value="lend" />
  <property name="successView" value="redirect:lends.action"/>
  <property name="cancelView" value="redirect:lends.action"/>
 </bean>

 <bean id="UnlendFormController" class="controllers.UnlendFormController">
  <property name="bs" ref="BookSession" />
  <property name="formView" value="unlendForm" />
  <property name="commandName" value="unlend" />
  <property name="successView" value="redirect:lends.action"/>
  <property name="cancelView" value="redirect:lends.action"/>
 </bean>

 <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
  <property name="basename" value="message" />
 </bean>
</beans>
```

Bijlage 7 – Spring MVC code

BookController.java

```
package controllers;

public class BookController extends MultiActionController {
  protected final Log logger = LogFactory.getLog(getClass());
  private BookSession bs;
  private BookEntity book;
  private BookEntity[] books;

  public ModelAndView detail(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("detail()");
    if(ServletRequestUtils.getStringParameter(request, "id")!=null) {
        int id = ServletRequestUtils.getIntParameter(request, "id");
        if(id!=0) {
          try {
            book = bs.retrieve(id);
            Map<String, BookEntity> myModel = new HashMap<String,BookEntity>();
            myModel.put("book", book);
            return new ModelAndView("book","model",myModel);
          }
          catch (BookNotFoundException e) { return new ModelAndView(new RedirectView("book.action?action=list")); }
        }
        else { return new ModelAndView(new RedirectView("book.action?action=list")); }
    }
    else { return new ModelAndView(new RedirectView("book.action?action=list")); }
  }

  public ModelAndView list(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("list()");
    books = bs.retrieveAll();
    Map<String, BookEntity[]> myModel = new HashMap<String,BookEntity[]>();
    myModel.put("books", books);
    return new ModelAndView("books","model",myModel);
  }

  public ModelAndView delete(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("delete()");
    if(ServletRequestUtils.getStringParameter(request, "id")!=null) {
      int id = ServletRequestUtils.getIntParameter(request, "id");
      if(id!=0) {
        try { bs.delete(id); }
        catch (BookNotFoundException e) {}
      }
    }
    return new ModelAndView(new RedirectView("book.action?action=list"));
  }

  public BookEntity[] getBooks() { return books; }
  public void setBooks(BookEntity[] books) { this.books = books; }

  public BookEntity getBook() { return book; }
  public void setBook(BookEntity book) { this.book = book; }

  public BookSession getBs() { return bs; }
  public void setBs(BookSession bs) { this.bs = bs; }
}
```

BookFormController.java

```
package controllers;

public class BookFormController extends CancellableFormController {
  protected final Log logger = LogFactory.getLog(getClass());
  private BookSession bs;
  private BookEntity book;

  public BookFormController() { }

  protected @Override Object formBackingObject(HttpServletRequest arg0) throws Exception {
    logger.info("formBackingObject()");
    BookSession bse = this.getBs();
    if(ServletRequestUtils.getStringParameter(arg0, "id")!=null) {
      int id = ServletRequestUtils.getIntParameter(arg0, "id");
      if(id!=0) {
        try { this.setBook(bse.retrieve(id)); }
        catch(Exception e) { this.setBook(bse.create()); }
      }
      else { this.setBook(bse.create()); }
    }
    else { this.setBook(bse.create()); }
    return this.getBook();
  }


  protected ModelAndView onSubmit(Object command) throws ServletException{
    logger.info("onSubmit()");
    BookSession bse = this.getBs();
    BookEntity be = (BookEntity) command;
    bse.save(be);
    return new ModelAndView(getSuccessView());
  }

  public BookEntity getBook() { return book; }
  public void setBook(BookEntity book) { this.book = book; }
  public BookSession getBs() { return bs; }
  public void setBs(BookSession bs) { this.bs = bs; }
}
```

CustomerController.java

```
package controllers;

public class CustomerController extends MultiActionController {
  protected final Log logger = LogFactory.getLog(getClass());
  private CustomerSession cs;
  private CustomerEntity customer;
  private CustomerEntity[] customers;

  public ModelAndView detail(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("handleRequest()");
    if(ServletRequestUtils.getStringParameter(request, "id")!=null) {
      int id = ServletRequestUtils.getIntParameter(request, "id");
      if(id!=0) {
        try {
          customer = cs.retrieve(id);
          Map<String, CustomerEntity> myModel = new HashMap<String,CustomerEntity>();
          myModel.put("customer", customer);
          return new ModelAndView("customer","model",myModel);
        }
        catch (CustomerNotFoundException e) { return new ModelAndView(new RedirectView("customers.action")); }
      }
      else { return new ModelAndView(new RedirectView("customers.action")); }
    }
    else { return new ModelAndView(new RedirectView("customers.action")); }
  }

  public ModelAndView delete(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("handleRequest()");
    if(ServletRequestUtils.getStringParameter(request, "id")!=null) {
      int id = ServletRequestUtils.getIntParameter(request, "id");
      if(id!=0) {
        try { cs.delete(id); }
        catch (CustomerNotFoundException e) {}
      }
    }
    return new ModelAndView(new RedirectView("customers.action"));
  }

  public ModelAndView list(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("handleRequest()");
    customers = cs.retrieveAll();
    Map<String, CustomerEntity[]> myModel = new HashMap<String,CustomerEntity[]>();
    myModel.put("customers", customers);
    return new ModelAndView("customers","model",myModel);
  }

  public CustomerSession getCs() { return cs; }
  public void setCs(CustomerSession cs) { this.cs = cs; }

  public CustomerEntity getCustomer() { return customer; }
  public void setCustomers(CustomerEntity customer) { this.customer = customer; }
}
```

CustomerFormController.java

```
package controllers;

public class CustomerFormController extends CancellableFormController {
  protected final Log logger = LogFactory.getLog(getClass());
  private CustomerSession cs;
  private CustomerEntity customer;
  private int id;

  public CustomerFormController() { }

  protected Object formBackingObject(HttpServletRequest arg0) throws Exception {
    logger.info("formBackingObject()");
    if(ServletRequestUtils.getStringParameter(arg0, "id")!=null) {
      int id = ServletRequestUtils.getIntParameter(arg0, "id");
      if(id!=0) {
        try { this.setCustomer(cs.retrieve(id)); }
        catch(Exception e) { this.setCustomer(cs.create()); }
      }
      else { this.setCustomer(cs.create()); }
    }
    else { this.setCustomer(cs.create()); }
    return this.getCustomer();
  }

  protected void initBinder(HttpServletRequest req,
    ServletRequestDataBinder binder) throws Exception {
    logger.info("initBinder()");
    CustomDatePropertyEditor cd = new CustomDatePropertyEditor();
    binder.registerCustomEditor(Date.class,cd);
  }

  protected ModelAndView onSubmit(Object command) throws ServletException{
    logger.info("onSubmit()");
    CustomerEntity ce = (CustomerEntity) command;
    cs.save(ce);
    return new ModelAndView(getSuccessView());
  }

  public int getId() { return id; }
  public void setId(int id) { this.id = id; }

  public CustomerSession getCs() { return cs; }
  public void setCs(CustomerSession cs) { this.cs = cs; }

  public CustomerEntity getCustomer() { return customer; }
  public void setCustomer(CustomerEntity customer) { this.customer = customer; }
}
```

LendFormController.java

```
package controllers;

public class LendFormController extends CancellableFormController {
  protected final Log logger = LogFactory.getLog(getClass());
  private BookSession bs;

  public LendFormController() { }

  protected Object formBackingObject(HttpServletRequest arg0) throws Exception {
    logger.info("formBackingObject()");
    LendCommand lendcmd = new LendCommand();
    if(ServletRequestUtils.getStringParameter(arg0, "bookid")!=null) {
      int bookid = ServletRequestUtils.getIntParameter(arg0, "bookid");
      if(bookid!=0) { lendcmd.setBookid(bookid); }
    }

    if(ServletRequestUtils.getStringParameter(arg0, "customerid")!=null) {
      int customerid = ServletRequestUtils.getIntParameter(arg0, "customerid");
      if(customerid!=0) { lendcmd.setCustomerid(customerid); }
    }
    return lendcmd;
  }

  protected ModelAndView onSubmit(Object command) throws ServletException{
    logger.info("onSubmit()");
    LendCommand lend = (LendCommand) command;
    try {
      bs.lend(lend.getBookid(), lend.getCustomerid());
      return new ModelAndView(getSuccessView());
    }
    catch(Exception e) {  return new ModelAndView(getSuccessView()); }
  }

  public BookSession getBs() { return bs; }
  public void setBs(BookSession bs) { this.bs = bs; }
}
```

LendsController.java

```
package controllers;

public class LendsController implements Controller {
  protected final Log logger = LogFactory.getLog(getClass());
  private BookSession bs;
  private BookEntity[] books;

  public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)throws Exception {
    logger.info("LendsController - handleRequest");
    books = bs.retrieveLend();
    Map<String, BookEntity[]> myModel = new HashMap<String,BookEntity[]>();
    myModel.put("books", books);
    return new ModelAndView("lends","model",myModel);
  }

  public BookEntity[] getBooks() { return books; }
  public void setBooks(BookEntity[] books) { this.books = books; }

  public BookSession getBs() { return bs; }
  public void setBs(BookSession bs) { this.bs = bs; }
}
```

UnlendFormController.java

```
package controllers;

public class UnlendFormController extends CancellableFormController {

  protected final Log logger = LogFactory.getLog(getClass());
  private BookSession bs;

  public UnlendFormController() { }

  protected Object formBackingObject(HttpServletRequest arg0) throws Exception {
    logger.info("formBackingObject()");
    LendCommand lendcmd = new LendCommand();
    if(ServletRequestUtils.getStringParameter(arg0, "bookid")!=null) {
      int bookid = ServletRequestUtils.getIntParameter(arg0, "bookid");
      if(bookid!=0) { lendcmd.setBookid(bookid); }
    }
    return lendcmd;
  }

  protected ModelAndView onSubmit(Object command) throws ServletException{
    logger.info("onSubmit()");
    LendCommand lend = (LendCommand) command;
    try {
      bs.unlend(lend.getBookid());
      return new ModelAndView(getSuccessView());
    }
    catch(Exception e) { return new ModelAndView(getSuccessView()); }
  }

  public BookSession getBs() { return bs; }
  public void setBs(BookSession bs) { this.bs = bs; }
}
```

CustomDatePropertyEditor.java

```
package editors;

public class CustomDatePropertyEditor extends PropertyEditorSupport{
  private Pattern pattern = Pattern.compile("(0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.](19|20)\\d\\d");

  @Override
  public void setAsText(String text) throws IllegalArgumentException{
    if(!StringUtils.hasText(text)) {
      throw new IllegalArgumentException("Date must not be empty or null");
    }
    else {
      Matcher matcher = pattern.matcher(text);
      if(matcher.matches()) {
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        try {
          Date d = sdf.parse(text);
          setValue(d);
        }
        catch (ParseException e) { throw new IllegalArgumentException("Date not parseble using pattern dd-MM-yyyy"); }
      }
      else { throw new IllegalArgumentException("Date does not match pattern dd-MM-yyyy"); }
    }
  }

  @Override
  public String getAsText() {
    if(getValue() != null) {
      SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
      return sdf.format(getValue());
    }
    else { return ""; }
  }
}
```

BookValidator.java

```
package validators;

public class BookValidator implements Validator {
  public boolean supports(Class clazz) {
    return BookEntity.class.equals(clazz);
  }

  public void validate(Object obj, Errors e) {
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "title", "field.required");
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "author", "field.required");
  }
}
```

CustomerValidator.java

```
package validators;

public class CustomerValidator implements Validator {

  public boolean supports(Class clazz) {
    return CustomerEntity.class.equals(clazz);
  }

  public void validate(Object obj, Errors e) {
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "firstName", "field.required");
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "lastName",  "field.required");
  }
}
```

Bijlage 8 – Spring MVC jsp pagina's

include.jsp

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

index.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Library</title>
  </head>
  <body>
    <h2>Welcome to the mini-library</h2>
    <a href="customers.action">Customers</a>
    <br />
    <a href="book.action?action=list">Books</a>
    <br />
    <a href="lends.action">Lends</a>
  </body>
</html>
```

book.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Book</title>
  </head>
  <body>
    <h2>Book details</h2>
    <table>
      <tr>
        <th align="left">Id:</th>
        <td>${model.book.id}</td>
      </tr>
      <tr>
        <th align="left">Title:</th>
        <td>${model.book.title}</td>
      </tr>
      <tr>
        <th align="left">Author:</th>
        <td>${model.book.author}</td>
      </tr>
      <c:choose>
        <c:when test="${!model.book.available}">
          <tr> <th align="left">Available:</th> <td>No</td> </tr>
          <tr> <th align="left">Lend to:</th> <td>${model.book.lendTo.fullName}</td> </tr>
          <tr> <th align="left"></th> <td><a href="unlendForm.action?bookid=${model.book.id}">Return book</a></td> </tr>
        </c:when>
        <c:otherwise>
          <tr> <th>Available:</th> <td>Yes</td> </tr>
          <tr> <th align="left"></th> <td><a href="lendForm.action?bookid=${model.book.id}">Lend book</a></td> </tr>
        </c:otherwise>
      </c:choose>
    </table>
    <a href="book.action?action=list">Back to books</a>
  </body>
</html>
```

bookForm.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <c:choose>
      <c:when test="${book.id == 0}">
        <title>Add new book</title>
      </c:when>
      <c:otherwise>
        <title>Edit book</title>
      </c:otherwise>
    </c:choose>
  </head>
  <body>
    <c:choose>
      <c:when test="${book.id == 0}">
        <h2>Add new book</h2>
      </c:when>
      <c:otherwise>
        <h2>Edit book</h2>
      </c:otherwise>
    </c:choose>
    <form:form method="post" commandName="book" action="bookForm.action">
      <table>
        <tr>
          <th align="left">Title:</th>
          <td><form:input path="title" /></td>
          <td><font color="red"><form:errors path="title" /></font></td>
        </tr>
        <tr>
          <th align="left">Title:</th>
          <td><form:input path="author" /></td>
          <td><font color="red"><form:errors path="author" /></font></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" value="Submit"></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" name="_cancel" value="Cancel"></td>
        </tr>
      </table>
      <form:hidden path="id" />
    </form:form>
  </body>
</html>
```

books.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Books</title>
  </head>
  <body>
    <h2>Books in the library</h2>
    <table border="1">
      <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Author</th>
        <th>Available</th>
        <th> </th>
        <th> </th>
        <th> </th>
        <th> </th>
      </tr>
      <c:forEach items="${model.books}" var="book">
        <tr>
          <td><c:out value="${book.id}"/></td>
          <td><c:out value="${book.title}"/></td>
          <td><c:out value="${book.author}"/></td>
          <c:choose>
            <c:when test="${!book.available}">
              <td>No</td>
            </c:when>
            <c:otherwise>
              <td>Yes</td>
            </c:otherwise>
          </c:choose>
          <td><a href="bookForm.action?id=<c:out value="${book.id}"/>">Edit</a></td>
          <td><a href="book.action?id=<c:out value="${book.id}"/>&amp;action=delete">Delete</a></td>
          <td><a href="book.action?id=<c:out value="${book.id}"/>&amp;action=detail">Details</a></td>
          <c:choose>
            <c:when test="${!book.available}">
              <td><a href="unlendForm.action?bookid=<c:out value="${book.id}"/>">Return book</a></td>
            </c:when>
            <c:otherwise>
              <td><a href="lendForm.action?bookid=<c:out value="${book.id}"/>">Lend book</a></td>
            </c:otherwise>
          </c:choose>
        </tr>
      </c:forEach>
    </table>
    <a href="bookForm.action">Add a book</a><br />
    <a href="index.jsp">Back to main</a>
  </body>
</html>
```

customer.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Customer</title>
  </head>
  <body>
    <h2>Customer details</h2>
    <table>
      <tr> <th align="left">Id:</th> <td>${model.customer.id}</td> </tr>
      <tr> <th align="left">Firstname:</th> <td>${model.customer.firstName}</td> </tr>
      <tr> <th align="left">Lastname:</th> <td>${model.customer.lastName}</td> </tr>
      <tr>
        <th align="left">Birthday:</th>
        <td><fmt:formatDate value="${model.customer.birthDayDate}" pattern="dd-MM-yyyy"/></td>
      </tr>
    </table>
    <h2>Lend books</h2>
    <table border="1">
      <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Author</th>
        <th>Available</th>
      </tr>
      <c:forEach items="${model.customer.books}" var="book">
        <tr>
          <td><c:out value="${book.id}"/></td>
          <td><c:out value="${book.title}"/></td>
          <td><c:out value="${book.author}"/></td>
          <td><a href="unlendForm.action?bookid=<c:out value="${book.id}"/>">Return book</a></td>
        </tr>
      </c:forEach>
    </table>
    <a href="customers.action">Back to customers</a>
  </body>
</html>
```

customerForm.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <c:choose>
      <c:when test="${customer.id == 0}">
        <title>Add new customer</title>
      </c:when>
      <c:otherwise>
        <title>Edit customer</title>
      </c:otherwise>
    </c:choose>
  </head>
  <body>
    <c:choose>
      <c:when test="${customer.id == 0}">
        <h2>Add new customer</h2>
      </c:when>
      <c:otherwise>
        <h2>Edit customer</h2>
      </c:otherwise>
    </c:choose>
    <form:form method="post" action="customerForm.action" commandName="customer">
      <table>
        <tr>
          <th align="left">Firstname:</th>
          <td><form:input path="firstName" size="30" /></td>
          <td><font color="red"><form:errors path="firstName" /></font></td>
        </tr>
        <tr>
          <th align="left">Lastname:</th>
          <td><form:input path="lastName" size="30"/></td>
          <td><font color="red"><form:errors path="lastName" /></font></td>
        </tr>
        <tr>
          <th align="left">Birthday:</th>
          <td><form:input path="birthDayDate" size="10"/></td>
          <td><font color="red"><form:errors path="birthDayDate" /></font></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" value="Submit"></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" name="_cancel" value="Cancel"></td>
        </tr>
      </table>
      <form:hidden path="id" />
    </form:form>
  </body>
</html>
```

customers.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Customers</title>
  </head>
  <body>
    <h2>Customers in the library</h2>
    <table border="1">
      <tr>
        <th>Id</th>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Birthday</th>
        <th> </th>
        <th> </th>
        <th> </th>
      </tr>
      <c:forEach items="${model.customers}" var="customer">
        <tr>
          <td><c:out value="${customer.id}"/></td>
          <td><c:out value="${customer.firstName}"/></td>
          <td><c:out value="${customer.lastName}"/></td>
          <td><fmt:formatDate value="${customer.birthDayDate}" pattern="dd-MM-yyyy"/></td>
          <td><a href="customerForm.action?id=<c:out value="${customer.id}"/>">Edit</a></td>
          <td><a href="customer.action?id=<c:out value="${customer.id}"/>&amp;action=delete">Delete</a></td>
          <td><a href="customer.action?id=<c:out value="${customer.id}"/>&amp;action=detail">Details</a></td>
        </tr>
      </c:forEach>
    </table>
    <a href="customerForm.action">Add a customer</a><br />
    <a href="index.jsp">Back to main</a>
  </body>
</html>
```

lendForm.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Lend a book</title>
  </head>
  <body>
    <h2>Lend book</h2>
    <form:form method="post" action="lendForm.action" commandName="lend">
      <table>
        <tr>
          <th align="left">Book:</th>
          <td><form:input path="bookid"/></td>
          <td><font color="red"><form:errors path="bookid" /></font></td>
        </tr>
        <tr>
          <th align="left">Customer:</th>
          <td><form:input path="customerid"/></td>
          <td><font color="red"><form:errors path="customerid" /></font></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" value="Submit"/></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" name="_cancel" value="Cancel"/></td>
        </tr>
      </table>
    </form:form>
  </body>
</html>
```

lends.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Lend books</title>
  </head>
  <body>
    <h2>Lend books in the library</h2>
    <a href="lendForm.action">Lend a book</a> <a href="unlendForm.action">Return a book</a>
    <table border="1">
      <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Author</th>
        <th>Lend to</th>
        <th> </th>
      </tr>
      <c:forEach items="${model.books}" var="book">
        <tr>
          <td><c:out value="${book.id}"/></td>
          <td><c:out value="${book.title}"/></td>
          <td><c:out value="${book.author}"/></td>
          <td><c:out value="${book.lendTo.fullName}"/></td>
          <td><a href="unlendForm.action?bookid=<c:out value="${book.id}"/>">Return book</a></td>
        </tr>
      </c:forEach>
    </table>
    <a href="index.jsp">Back to main</a>
  </body>
</html>
```

unlendForm.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<html>
  <head>
    <title>Return a book</title>
  </head>
  <body>
    <h2>Return book</h2>
    <form:form method="post" action="unlendForm.action" commandName="unlend">
      <table>
        <tr>
          <th align="left">Book:</th>
          <td><form:input path="bookid"/></td>
          <td><font color="red"><form:errors path="bookid" /></font></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" value="Submit"/></td>
        </tr>
        <tr>
          <th align="left"></th>
          <td></td>
          <td align="right"><input type="submit" name="_cancel" value="Cancel"/></td>
        </tr>
      </table>
    </form:form>
  </body>
</html>
```

Bijlage 9 – Struts 2 configuratie

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
 version="2.4"
>
  <display-name>PrototypeStruts</display-name>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!-- The Usual Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
</web-app>
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="bookSession" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="BookSessionBean/remote" />
  </bean>
  <bean id="customerSession" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="CustomerSessionBean/remote" />
  </bean>
  <bean
      id="org.apache.struts2.dispatcher.ServletDispatcherResult"
      class="org.apache.struts2.dispatcher.ServletDispatcherResult">
  </bean>
</beans>
```

struts.properties

```
struts.devMode = false
struts.enable.DynamicMethodInvocation = true
struts.objectFactory = spring
struts.objectFactory.spring.autoWire = name
```

struts.xml

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <!-- Include Struts 2 default (from Struts 2 JAR). -->
  <include file="struts-default.xml"/>

  <!-- Configuration for the default package. -->
  <package name="StrutsSample" extends="struts-default">
    <default-interceptor-ref name="defaultStack"/>

    <action name="customers" class="actions.CustomerAction" method="list">
      <result>/customers.jsp</result>
    </action>

    <action name="customer" class="actions.CustomerAction" method="detail">
        <result name="success" type="redirect-action">customers</result>
        <result name="detail">/customer.jsp</result>
        <result name="input">/customerForm.jsp</result>
        <result name="error">error.jsp</result>
    </action>

    <action name="books" class="actions.BookAction" method="list">
        <result>/books.jsp</result>
    </action>

    <action name="book" class="actions.BookAction" method="detail">
      <result name="success" type="redirect-action">books</result>
      <result name="detail">/book.jsp</result>
      <result name="input">/bookForm.jsp</result>
      <result name="error">error.jsp</result>
    </action>

    <action name="lend" class="actions.LibraryAction" method="lend">
      <result name="success" type="redirect-action">lends</result>
      <result name="input">/lendForm.jsp</result>
      <result name="error">error.jsp</result>
    </action>

    <action name="unlend" class="actions.LibraryAction" method="unlend">
      <result name="success" type="redirect-action">lends</result>
      <result name="input">/unlendForm.jsp</result>
      <result name="error">error.jsp</result>
    </action>

    <action name="lends" class="actions.LibraryAction" method="list">
      <result>/lends.jsp</result>
    </action>
  </package>
</struts>
```

Bijlage 10 – Struts 2 code

BookAction.java

```
package actions;

public class BookAction extends ActionSupport {

  protected final Logger logger = Logger.getLogger(getClass());
  private BookSession bookSession;
  private BookEntity[] books;
  private BookEntity book;
  private CustomerEntity customer;
  private int id;

  public String list() {
    logger.info("list()");
    setBooks(bookSession.retrieveAll());
    return SUCCESS;
  }

  @SkipValidation
  public String detail() {
    logger.info("detail()");
    if(id != 0) {
      try {
        book = bookSession.retrieve(id);
        return "detail";
      } catch (BookNotFoundException e) { return ERROR; }
    }
    else { return ERROR; }
  }

  @SkipValidation
  public String add() {
    logger.info("add()");
    book = bookSession.create();
    return INPUT;
  }

  @SkipValidation
  public String edit() {
    logger.info("edit()");
    try {
      book = bookSession.retrieve(id);
      return INPUT;
    } catch (BookNotFoundException e) { return ERROR; }
  }

  @SkipValidation
  public String delete() {
    logger.info("delete()");
    try {
      bookSession.delete(id);
      return SUCCESS;
    }  catch (BookNotFoundException e) { return ERROR; }
  }

  public String save() {
    logger.info("save()");
    if(book != null) {
      try {
        BookEntity btemp = bookSession.retrieve(book.getId());
        book.setLendTo(btemp.getLendTo());
      } catch (BookNotFoundException e) { book.setId(0); }
      bookSession.save(book);
      return SUCCESS;
    }
    else { return ERROR; }
  }
```

```
  public BookEntity getBook() { return book; }
  public void setBook(BookEntity book) { this.book = book; }

  public BookEntity[] getBooks() { return books; }
  public void setBooks(BookEntity[] books) { this.books = books; }

  public CustomerEntity getCustomer() { return customer; }
  public void setCustomer(CustomerEntity customer) { this.customer = customer; }

  public int getId() { return id; }
  public void setId(int id) { this.id = id; }

  public void setBookSession(BookSession bs) { this.bookSession = bs; }
}
```

CustomerAction.java

```
package actions;

public class CustomerAction extends ActionSupport implements Preparable {

 protected final Logger logger = Logger.getLogger(getClass());

  private CustomerSession customerSession;
  private CustomerEntity[] customers;
  private CustomerEntity customer;
  private Collection<BookEntity> books = new TreeSet<BookEntity>();
  private int id;

  public String list() throws Exception {
    logger.info("list()");
    setCustomers(customerSession.retrieveAll());
    return SUCCESS;
  }

  @SkipValidation
  public String detail() {
    logger.info("detail()");
    try {
      customer = customerSession.retrieve(id);
      books = customer.getBooks();
      return "detail";
    } catch (CustomerNotFoundException e) { return ERROR; }
  }

  @SkipValidation
  public String add() {
    logger.info("add()");;
    customer = customerSession.create();
    return INPUT;
  }

  @SkipValidation
  public String edit() {
    logger.info("edit()");
    try {
      customer = customerSession.retrieve(id);
      return INPUT;
    } catch (CustomerNotFoundException e) { return ERROR; }
  }

  @SkipValidation
  public String delete() {
    logger.info("delete()");
    try {
      customerSession.delete(id);
    } catch (CustomerNotFoundException e) { }
    return SUCCESS;
  }
```

```
  public String save() {
    logger.info("save()");
    if(customer != null) {
      customerSession.save(customer);
    }
    else { return ERROR; }
    return SUCCESS;
  }

  public void setCustomers(CustomerEntity[] customers) { this.customers = customers; }
  public CustomerEntity[] getCustomers() { return customers; }

  public void setId(int id){ this.id = id; }
  public int getId() { return id; }

  public void setCustomer(CustomerEntity customer) { this.customer = customer; }
  public CustomerEntity getCustomer() { return customer; }

  public void setBooks(Collection<BookEntity> books) { this.books = books; }
  public Collection<BookEntity> getBooks() { return books; }

  public void setCustomerSession(CustomerSession cs) { this.customerSession = cs; }
}
```

LibraryAction.java

```
package actions;

public class LibraryAction  extends ActionSupport {
  protected final Logger logger = Logger.getLogger(getClass());
  private BookEntity[] lendbooks;
  private BookSession bookSession;
  private int bookid;
  private int customerid;
  private String errorMessage;

  @SkipValidation
  public String lend() {
    logger.info("lend()");
    return INPUT;
  }

  @SkipValidation
  public String unlend() {
    logger.info("unlend()");
    return INPUT;
  }

  public String savelend() {
    logger.info("savelend()");
    try {
      bookSession.lend(bookid, customerid);
      return SUCCESS;
    }
    catch(Exception e) {
      errorMessage = e.getMessage();
      return INPUT;
    }
  }

  public String saveunlend() {
    logger.info("saveunlend()");
    try {
      bookSession.unlend(bookid);
      return SUCCESS;
    }
    catch(Exception e) {
      errorMessage = e.getMessage();
      return INPUT;
    }
  }
```

```
  @SkipValidation
  public String list() {
    logger.info("list()");
    lendbooks = bookSession.retrieveLend();
    return SUCCESS;
  }

  public int getBookid() { return bookid; }
  public void setBookid(int bookid) { this.bookid = bookid; }

  public int getCustomerid() { return customerid; }
  public void setCustomerid(int customerid) { this.customerid = customerid; }

  public String getErrorMessage() { return errorMessage; }
  public void setErrorMessage(String errorMessage) { this.errorMessage = errorMessage; }

  public BookEntity[] getLendbooks() { return lendbooks; }
  public void setLendbooks(BookEntity[] lendbooks) { this.lendbooks = lendbooks; }

  public void setBookSession(BookSession bookSession) { this.bookSession = bookSession; }
}
```

Bijlage 11 – Struts 2 jsp pagina's

book.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Book</title>
  </head>
  <body>
    <h2>Book details</h2>
    <table>
      <tr>
        <th align="left">Id:</th>
        <td><s:property value="book.id"/></td>
      </tr>
      <tr>
        <th align="left">Title:</th>
        <td><s:property value="book.title"/></td>
      </tr>
      <tr>
        <th align="left">Author:</th>
        <td><s:property value="book.author"/></td>
      </tr>
      <s:if test="!book.available">
        <tr>
          <th align="left">Available:</th>
          <td>No</td>
        </tr>
        <tr>
          <th align="left">Lend to:</th>
          <td><s:property value="book.lendTo.fullName"/></td>
        </tr>
      </s:if>
      <s:else>
        <tr>
          <th align="left">Available:</th>
          <td>Yes</td>
        </tr>
      </s:else>
      <s:if test="!book.available">
        <tr>
          <td></td>
          <td>
            <s:url id="url2" action="unlend"><s:param name="bookid" value="id" /></s:url>
            <a href="<s:property value="#url2"/>">Return book</a>
          </td>
        </tr>
      </s:if>
      <s:else>
        <tr>
          <td></td>
          <td>
            <s:url id="url2" action="lend"><s:param name="bookid" value="id" /></s:url>
            <a href="<s:property value="#url2"/>">Lend book</a>
          </td>
        </tr>
      </s:else>
    </table>
    <s:url id="url" action="books"  />
    <a href="<s:property value="#url"/>">Back to books</a>
  </body>
</html>
```

bookForm.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <s:head theme="ajax"/>
    <s:if test="book==null || book.Id == 0">
      <title>Add new book</title>
    </s:if>
    <s:else>
      <title>Edit book</title>
    </s:else>
  </head>
  <body>
    <s:if test="book==null || book.Id == 0">
      <h2>Add new book</h2>
    </s:if>
    <s:else>
      <h2>Edit book</h2>
    </s:else>
    <s:form action="book!save" method="post">
      <s:textfield key="book.title" label="Title" size="40" required="true"/>
      <s:textfield key="book.author" label="Author" size="40" required="true"/>
      <s:hidden key="book.Id" />
      <s:submit value="Submit"/>
      <s:submit value="Cancel" name="redirect-action:books"/>
    </s:form>
  </body>
</html>
```

books.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Books</title>
  </head>
  <body>
    <h2>Books in the library</h2>
    <table border="1">
      <tr>
        <th>Id</th><th>Title</th> <th>Author</th> <th>Available</th> <th> </th> <th> </th> <th> </th>
        <th> </th>
      </tr>
      <s:iterator value="books">
        <tr>
          <td><s:property value="id"/></td>
          <td><s:property value="title"/></td>
          <td><s:property value="author"/></td>
          <s:if test="!available">
            <td>No</td>
          </s:if>
          <s:else>
            <td>Yes</td>
          </s:else>
          <td>
            <s:url id="url" action="book!edit"><s:param name="id" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Edit</a>
          </td>
          <td>
            <s:url id="url" action="book!delete"><s:param name="id" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Delete</a>
          </td>
          <td>
            <s:url id="url" action="book!detail"><s:param name="id" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Details</a>
          </td>
```

```
        <s:if test="!available">
          <td>
            <s:url id="url" action="unlend"><s:param name="bookid" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Return book</a>
          </td>
        </s:if>
        <s:else>
          <td>
            <s:url id="url" action="lend"><s:param name="bookid" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Lend book</a>
          </td>
        </s:else>
      </tr>
    </s:iterator>
  </table>
  <s:url id="url" action="book!add"  />
  <a href="<s:property value="#url"/>">Add a book</a>
  <br />
  <a href="index.jsp">Back to main</a>
 </body>
</html>
```

customer.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
 <head>
  <title>Customer</title>
 </head>
 <body>
  <h2>Customer details</h2>
  <table>
   <tr>
    <td>Id:</td>
    <td><s:property value="customer.id"/></td>
   </tr>
   <tr>
    <td>Firstname:</td>
    <td><s:property value="customer.firstName"/></td>
   </tr>
   <tr>
    <td>Lastname:</td>
    <td><s:property value="customer.lastName"/></td>
   </tr>
   <tr>
    <td>Birthday:</td>
    <td><s:date name="customer.birthDayDate" format="dd-MM-yyyy" /></td>
   </tr>
  </table>
  <h2>Lend books</h2>
  <table>
   <tr>
    <th>Id</th>
    <th>Title</th>
    <th>Lastname</th>
    <th>Author</th>
    <th> </th>
   </tr>
   <s:iterator value="books">
    <tr>
     <td><s:property value="id"/></td>
     <td><s:property value="title"/></td>
     <td><s:property value="author"/></td>
     <td>
      <s:url id="url2" action="unlend"  includeParams="none"><s:param name="bookid" value="id" /></s:url>
      <a href="<s:property value="#url2"/>">Return Book</a>
     </td>
    </tr>
   </s:iterator>
  </table>
  <s:url id="url" action="customers"  includeParams="none" />
  <a href="<s:property value="#url"/>">Back to customers</a>
 </body>
</html>
```

customerForm.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <s:head theme="ajax"/>
    <s:if test="customer==null || customer.Id == 0">
      <title>Add new customer</title>
    </s:if>
    <s:else>
      <title>Edit customer</title>
    </s:else>
  </head>
  <body>
    <s:if test="customer==null || customer.id == 0">
      <h2>Add new customer</h2>
    </s:if>
    <s:else>
      <h2>Edit customer </h2>
    </s:else>
    <s:form action="customer!save" method="post">
      <s:textfield key="customer.firstName" label="Firstname" size="40" required="true"/>
      <s:textfield key="customer.lastName" label="Lastname" size="40" required="true"/>
      <s:date id="bdate"name="customer.birthDayDate" format="dd-MM-yyyy" />
      <s:datetimepicker name="customer.birthDayDate" value="%{#bdate}" displayFormat="dd-MM-yyyy" label="Birthday"
        required="true"/>
      <s:hidden key="customer.Id"/>
      <s:submit value="Submit"/>
      <s:submit value="Cancel" name="redirect-action:customers"/>
    </s:form>
  </body>
</html>
```

customers.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Customers</title>
  </head>
  <body>
    <h2>Customers in the library</h2>
    <table border="1">
      <tr>
        <th>Id</th>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Birthday</th>
        <th> </th>
        <th> </th>
        <th> </th>
      </tr>
      <s:iterator value="customers">
        <tr>
          <td><s:property value="id"/></td>
          <td><s:property value="firstName"/></td>
          <td><s:property value="lastName"/></td>
          <td><s:date name="birthDayDate" format="dd-MM-yyyy" /></td>
          <td>
            <s:url id="url" action="customer!edit"><s:param name="id" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Edit</a>
          </td>
          <td>
            <s:url id="url" action="customer!delete"><s:param name="id" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Delete</a>
          </td>
          <td>
            <s:url id="url" action="customer!detail"><s:param name="id" value="id" /></s:url>
            <a href="<s:property value="#url"/>">Details</a>
          </td>
        </tr>
      </s:iterator>
    </table>
    <s:url id="url" action="customer!add"  />
```

```
    <a href="<s:property value="#url"/>">Add a customer</a>
    <br />
    <a href="index.jsp">Back to main</a>
  </body>
</html>
```

index.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Library</title>
  </head>
  <body>
    <h2>Welcome to the mini-library</h2>
    <s:url id="url" action="customers"  />
    <a href="<s:property value="#url"/>">Customers</a>
    <br />
    <s:url id="url" action="books"  />
    <a href="<s:property value="#url"/>">Books</a>
    <br />
    <s:url id="url" action="lends"  />
    <a href="<s:property value="#url"/>">Lends</a>
  </body>
</html>
```

lendForm.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <s:head theme="ajax"/>
    <title>Lend a book</title>
  </head>
  <body>
    <h2>Lend book</h2>
    <s:if test="errorMessage!=null">
      <font color="red"><strong><s:property value="errorMessage"/></strong></font>
    </s:if>
    <s:form action="lend!savelend" method="post">
      <s:textfield name="bookid" value="%{bookid}" label="Book" size="10" required="true"/>
      <s:textfield name="customerid" value="%{customerid}" label="Customer" size="10" required="true"/>
      <s:submit value="Submit"/>
      <s:submit value="Cancel" name="redirect-action:lends"/>
    </s:form>
  </body>
</html>
```

lends.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Lend books</title>
  </head>
  <body>
    <h2>Lend books in the library</h2>
    <s:url id="url" action="lend"  />
    <a href="<s:property value="#url"/>">Lend a book</a>
    <s:url id="url" action="unlend"  />
    <a href="<s:property value="#url"/>">Return a book</a>
    <br />
    <table border="1">
      <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Author</th>
        <th>Lend to</th>
        <th> </th>
      </tr>
      <s:iterator value="lendbooks">
      <tr>
        <td><s:property value="id"/></td>
        <td><s:property value="title"/></td>
        <td><s:property value="author"/></td>
        <td><s:property value="lendTo.fullName"/></td>
        <td>
          <s:url id="url" action="unlend"><s:param name="bookid" value="id" /></s:url>
          <a href="<s:property value="#url"/>">Return book</a>
        </td>
      </tr>
    </s:iterator>
    </table>
    <a href="index.jsp">Back to main</a>
  </body>
</html>
```

unlendForm.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <s:head theme="ajax"/>
    <title>Return a book</title>
  </head>
  <body>
    <h2>Return book</h2>
    <s:if test="errorMessage!=null">
      <font color="red"><strong><s:property value="errorMessage"/></strong></font>
    </s:if>
    <s:form action="unlend!saveunlend" method="post">
      <s:textfield name="bookid" value="%{bookid}" label="Book" required="true"/>
      <s:submit value="Submit"/>
      <s:submit value="Cancel" name="redirect-action:lends"/>
    </s:form>
  </body>
</html>
```