

---

# Onderzoek Framework

*Standaard framework webapplicatie*

**Afstuderen**

Bert Gritter  
Rob Juurlink  
2004

Laatste wijziging:  
maandag 19 april 2004  
15:32:02 uur.

---

# Onderzoek Framework

*Standaard framework webapplicatie*

## Versiebeheer

<i>Datum</i>	<i>auteur</i>	<i>Versie</i>	<i>Status/Wijziging</i>
25-03-2004	Rob	0.1	Start document
07-04-2004	Rob	0.2	Uitwerken HTTP Cache werking
08-04-2004	Rob	0.3	SQL Maps beschreven
10-04-2004	Rob	0.4	Documentatie WebWork ingevoegd. Werking cache uitgebreid en schema toegevoegd. Beschrijving Velocity beschrijvingstaal SQL Maps beschrijving herschreven Beschrijven Struts Framework
12-04-2004	Rob	0.5	Beschrijving Maverick framework
13-04-2004	Rob	0.6	Deel van conclusie geschreven en testplan uitgewerkt
10-04-2004	Bert	0.7	Beschrijving WebWork
13-04-2004	Bert, Rob	1.0	Afwerking van het document.
13-04-2004	Bert, Rob	1.1	Review

# INHOUDSOPGAVE

<b>1.</b>	<b>Inleiding</b>	<b>4</b>
<b>2.</b>	<b>Standaard framework</b>	<b>5</b>
2.1.	MVC framework	5
2.2.	Het MVC model	6
2.2.1.	Beschrijving van een aanvraag	6
2.2.2.	Apache Struts framework	7
	De Dispatcher	8
	De action controller	8
	Formulieren	9
	Validatie	9
2.2.3.	Maverick framework	10
	De Dispatcher	10
	De controller	10
	Het model	11
	De view	11
2.2.4.	WebWork framework	12
	Achtergrond	12
	Installatie	12
	Configureren	12
	Werking	12
2.3.	De view	14
2.3.1.	Java Server Pages (JSP)	14
	JSP 2.0	14
	Expression Language	14
2.3.2.	Velocity	15
2.4.	Persistentie laag	16
2.4.1.	iBatis SQL Maps	16
	Werking SQL Maps	16
	Een voorbeeld	17
2.4.2.	Apache data-sources	18
	server.xml	18
2.5.	caching	20
2.5.1.	Cache Control HTTP Headers	21
	HTTP 1.0 Expires header	21
	HTTP 1.1 Cache-Control header	21
	Valideren van een verlopen pagina - HTTP 1.0	22
	Valideren van een verlopen pagina - HTTP 1.1	22
2.5.2.	Aanpassingen aan het MVC Framework	23
	getLastModified()	23
	Aanpassingen aan de controller/actie	23
	Cache instellen	24
2.5.3.	Aanbevelingen bij gebruik van cache	25
2.5.4.	Prefetching	25
<b>3.</b>	<b>Ontwerp</b>	<b>26</b>
3.1.	Probleemstelling	26
3.2.	Ontwerp beslissingen	26
3.2.1.	CMP versus SQL Maps	26
3.3.	Analyse	27
3.3.1.	Verwoording	27
3.3.2.	Uitwerking	28
	Kernzinnen	28
	Datamodel	29
	Classediagram	29
	Applicatieschermen	30

<b>4.</b>	<b>Testen</b>	<b>32</b>
4.1.	Performance test	32
4.2.	Testplan en testrapport	33
4.2.1.	Framework	33
4.2.2.	Data persistentie	33
4.2.3.	Cache	34
<b>5.</b>	<b>Conclusie</b>	<b>35</b>
5.1.	Struts, Maverick of WebWork	35
5.2.	Performance	35
5.3.	Persistentie	35
5.4.	Caching	35
<b>6.</b>	<b>Referenties</b>	<b>36</b>



# 1. INLEIDING

Om de verantwoordelijkheden van de verschillende delen van de code te scheiden, wordt er gebruik gemaakt van meerdere lagen. Er wordt in de praktijk een tussenlaag gebruikt.

Er zijn bestaande implementaties van zo'n soort ontwerp. In dit document staan de resultaten van het onderzoek naar drie verschillende frameworken voor webapplicaties.

Er zijn een aantal lagen te onderscheiden, zo is er een persistentie laag en een view. De persistentie laag zorgt voor de opslag van data. De presentatielaag, ook wel view genoemd, zorgt voor het afbeelden van de gegevens in een nette vorm.

De presentatielaag moet niet meer code bevatten dan nodig is om een view af te beelden. Deze zogenaamde webtier (omdat in ons geval een webapplicatiei betreft) mag alleen logica bevatten voor webspecifieke delen. (Layout afbeelden, tabellen op het scherm zetten, opmaak enz). Er mag absoluut geen business code in voorkomen (bijvoorbeeld voor het ophalen van data uit de databank)

De uitgewerkte frameworken implementeren allen het MVC model. Details van dit model worden in dit document beschreven.

Verder is er door middel van een test gekeken of er performance verschil zit tussen de verschillende MVC implementaties Struts, Maverick en Webwork. In de praktijk is er praktisch geen verschil.

Daarnaast is het framework in totaal met twee verschillende views getest. Velocity en JSP icm JSTL. Ook tussen deze beide technieken is in de praktijk wat betreft de performance geen verschil waarneembaar.

Als het op de leesbaarheid van de code van de view aankomt, is Velocity de grote winnaar.

Tijdens het onderzoek naar het te gebruiken framework en bij het uitwerken van de test voorbeelden, bleek dat het cachten van informatie veel invloed op de performance had. Om die reden is er veel aandacht aan dit onderwerp besteed, terwijl het in de originele opdrachtomschrijving niet genoemd wordt. Er is onderzoek gedaan hoe HTTP caching werkt en er is een eigen aanpassing gemaakt om HTTP Cache Control te activeren in de bestaande frameworken.

---

## 2. STANDAARD FRAMEWERK

### 2.1. MVC FRAMEWERK

Het gebruik van een standaard framework maakt een eenvoudige applicatie in eerste instantie complexer, maar doordat het framework ook in toekomstige projecten gebruikt kan worden, is het toch een voordeel gebruik te maken van een framework.

Als het framework en de aanbevelingen consequent gevolgd worden, is door de strikte scheiding van taken altijd duidelijk waar wat gebeurt en waar een eventueel toekomstig probleem opgelost moet worden.

Door gebruik te maken van een open-source framework oplossing, wordt er kennis gebruikt waar ook door vele anderen over nagedacht is en aan bijgedragen. Ook is deze kant en klare oplossing meestal al voorbereid op toekomstige uitbreidingen waar de gebruiker zelf waarschijnlijk nog niet bij stil gestaan heeft, omdat hij of zij er simpelweg nog niet tegenaan gelopen is.

Een nadeel waar wel rekening mee gehouden moet worden is het feit dat de beschikbare frameworken nog niet zo lang bestaan en daardoor nog niet uitontwikkeld zijn. Er zal nog regelmatig een nieuwe versie uitgebracht worden en per uitgebrachte versie zal er nog veel veranderen.

Het is ondoenlijk om al geschreven code constant te actualiseren. Er zal op een gegeven moment een keuze gemaakt moeten worden over de gebruikte versie van het framework. Deze versie wordt dan gebruikt totdat de versie niet meer voldoet of tot het moment dat de laatste versie veel bruikbare functionaliteit toevoegt.

## 2.2. HET MVC MODEL

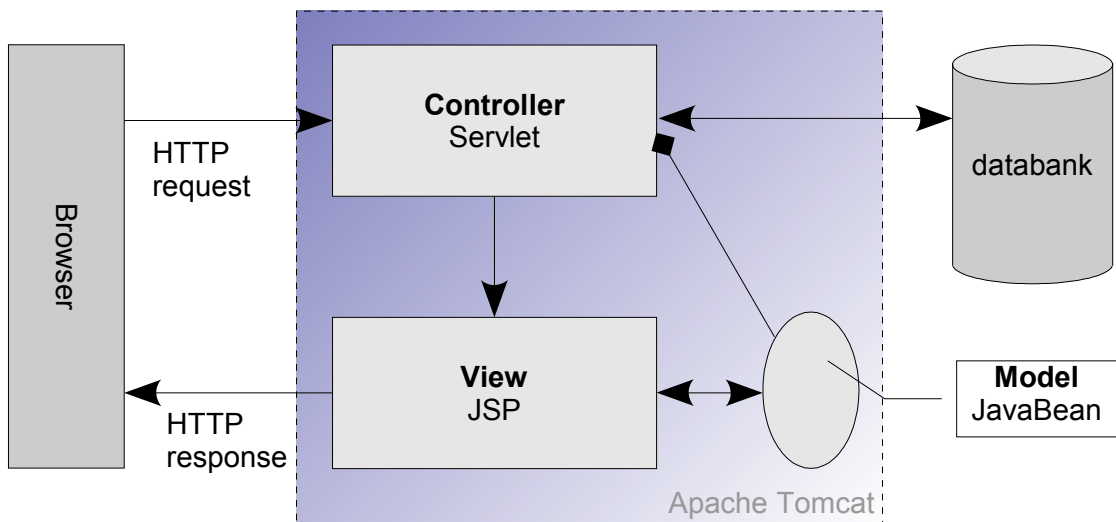
MVC, dat staat voor Model, View en Controller, is een design pattern welke een component verdeelt in drie verschillende delen. Te weten:

- **model** – Het model dat de data bevat.
- **view** – De view maakt het datamodel zichtbaar
- **controller** – Reageert op acties van gebruiker en werkt het datamodel bij.

Een ontwerp dat voldoet aan het MVC pattern wordt ook wel een “Model 2” ontwerp genoemd.

In een goed ontworpen webapplicatie is de web tier zo dun als mogelijk. De applicatie wordt gebouwd op goed gespecificeerde business interfaces. Een web tier zorgt feitelijk alleen voor het vertalen van de gebruikersopdrachten naar af te beelden datamodellen.

Het MVC model voor het web is niet gelijk aan het traditionele “push”-model in verband met de technische beperkingen van HTTP ten opzichte van een applicatie dat lokaal in een grafische gebruikersinterface draait. MVC voor het web is een zogenaamd “pull”-model. De gebruiker geeft een opdracht en de gegevens worden daarna uit het model “getrokken” en naar de webbrowser verzonden.



Figuur 1, het MVC Pull model voor het web, schematisch weergegeven.

In figuur 1 hierboven zijn de verschillende stappen afgebeeld die door een op MVC gebaseerde web applicatie achtereenvolgens uitgevoerd worden als er een aanvraag verwerkt wordt.

### 2.2.1. Beschrijving van een aanvraag

In een webapplicatie volgens het schema in figuur 1 worden achtereenvolgens de volgende handelingen uitgevoerd:

- De gebruiker typt het adres van de webapplicatie in in de browser. De browser verstuurt de aanroep naar de Apache Tomcat webserver waarin zich de webapplicatie bevindt.

- De controller bepaalt aan de hand van de gegevens in de aanroep welke code uitgevoerd dient te worden. Deze code wordt uitgevoerd en het datamodel wordt gecreëerd. Eventueel wordt er een databank aangesproken om het datamodel te vullen.
- De aanvraag wordt gedelegeerd naar de view die de pagina opbouwt. Voor het opbouwen van de pagina worden de gegevens uit het datamodel gebruikt. De view stuurt de reactie terug naar de browser van de gebruiker.
- De browser maakt de pagina zichtbaar voor de gebruiker.

Als data container worden JavaBeans gebruikt. Een JavaBean is een eenvoudig object dat variabelen bevat en methoden om deze variabelen een waarde te geven en uit te lezen.

Een goed ontworpen MVC model voor het web is moeilijk op te zetten, daarom gebruiken we hiervoor een bestaand framework. Het doel van een framework is om de code eenvoudiger en goed gescheiden te houden.

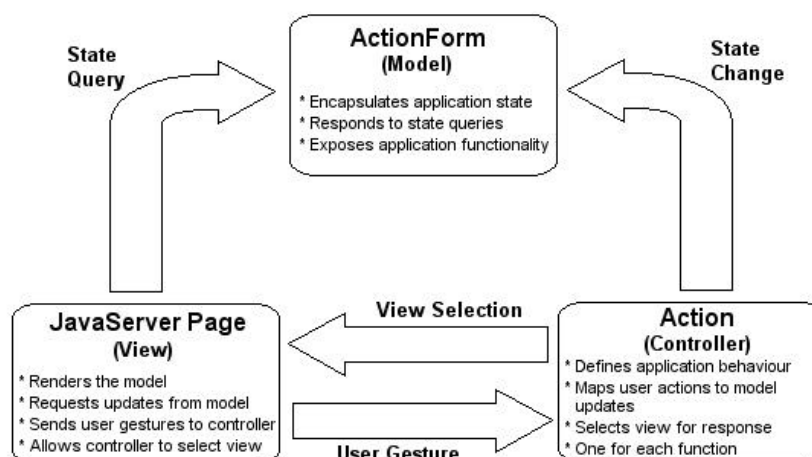
Er bestaan een aantal volgroeide open-source frameworken die zich in de praktijk hebben bewezen. Na een speurtocht op het internet blijkt dat onderstaande de meest gebruikte zijn:

- **Struts** – ontwikkeld door Apache
- **Maverick** – multi-platform, ook beschikbaar voor .NET en PHP.
- **WebWork** – Hetzelfde principe als Maverick

De verschillende frameworken worden achtereenvolgens in bovenstaande volgorde besproken.

## 2.2.2. Apache Struts framework

Apache Struts is een framework volgens het MVC model. In onderstaande figuur is het Struts MVC model schematisch weergegeven.



Figuur 2, Apache Struts MVC model.

## De Dispatcher

In de configuratie van de webmodule (welke zich altijd in het bestand `web.xml` bevindt dat op z'n beurt weer in de `WEB-INF/` map van een webapplicatie staat) wordt een zogenaamde `dispatcher` geconfigureerd. De Dispatcher is de class waarin de code staat die bepaalt welke `action controller` er uitgevoerd gaat worden. De `action controller` wordt bepaald aan de hand van de naam van de actie. De naam van de actie is gekoppeld aan de URL, zie onderstaand fragment uit de configuratie (`web.xml`):

```
<!-- Action Servlet Mapping -->
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>

<!-- Action Servlet Configuration -->
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

*web.xml, het configuratiebestand van de webapplicatie bevat de mapping die alle aanvragen met de extensie .do doorsturen naar een zogenaamde dispatcher class die bepaalt welke controller class er uitgevoerd wordt.*

## De action controller

De action controller (de C in MVC) is in Struts het onderdeel dat bepaalt welke code er uitgevoerd dient te worden. In het configuratiebestand van Struts (`struts-config.xml`) wordt gedefinieerd welke acties aan welke code gekoppeld wordt. In onderstaand voorbeeld is een inlogactie gekoppeld aan de Java class `LogonAction`. Een actie wordt afgeleid uit de URL.

```
<!-- Inloggen -->

<action path="/logon"
        type="nl.arsoftware.presentation.action.LogonAction"
        name="logonForm"
        input="logonView">
  <forward name="success"
          path="/ingelogd.jsp"/>
</action>
```

*Een actie configureren in Struts. Een actie bestaat uit een id (path), de class (type), een view (name) en een view waarop de data ingevuld en verzonden kan worden (input). Als de actie succesvol uitgevoerd wordt, wordt de view genaamd "success" uitgevoerd. Deze view toont de pagina ingelogd.jsp.*

## Formulieren

De velden op een HTML formulier kunnen gekoppeld worden aan een zogenaamde `formbean`. Een `formbean` is een 1 op 1 koppeling tussen de veldnamen op de HTML pagina en de `JavaBean`.

De `formbean` wordt geconfigureerd in de Struts configuratie. Doordat de waarden van de velden aanwezig zijn in de `JavaBean`, wordt validatie in combinatie met de Struts validatie plugin eenvoudig.

## Validatie

Validatie van een verzonden HTML formulier zit standaard ingebakken in Struts. Het valideren wordt uitgevoerd door het Struts framework. De validatie plugin moet dan van tevoren wel even geactiveerd worden in de configuratie van Struts. Dit gaat door het onderstaande aan `struts-config.xml` toe te voegen:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
    value="/WEB-INF/validator-rules.xml,
    /WEB-INF/validation.xml"/>
</plug-in>
```

*Aan Struts de mogelijkheid voor veld validatie toevoegen. Deze uitbreiding wordt geactiveerd door de plugin toe te voegen.*

De validatie regels moeten gedefinieerd worden in een speciaal XML configuratie bestand. Per veld wordt in dit configuratie bestand aangegeven waar het aan voldoen moet. Een deel van dit configuratiebestand ziet er als volgt uit. In onderstaand voorbeeld is het veld `totalTime` verplicht en moet het een geheel getal zijn.

```
<!-- Registratieformulier.
  Dit bestaat uit 1 veld dat wijzigbaar is door de gebruiker,
  dit veld moet numeriek zijn en mag niet leeg gelaten worden.
-->
<form name="registrationForm">
  <field property="totalTime"
    depends="required,integer">
    <arg0 key="registration.label.time"/>
  </field>
</form>
```

*In bovenstaande configuratie wordt aangegeven dat in het formulier genaamd `registrationForm` een veld `totalTime` aanwezig is. Dit veld is verplicht en het moet een geheel getal zijn.*

### 2.2.3. Maverick framework

Maverick is een framework volgens het MVC model. Volgens de makers van Maverick bestaat dit framework uit een combinatie van de beste onderdelen uit Struts, WebWork en Cocoon.

Kenmerken van Maverick:

- Een eenvoudige lichte API (in vergelijking met bijvoorbeeld Struts).
- Volledig te configureren door middel van XML bestanden.
- Geschikt voor verschillende views (JSP, Velocity, XML enz)
- Uitbreidbaar

#### De Dispatcher

In de configuratie van de webmodule wordt ook hier een zogenaamde dispatcher geconfigureerd. De Dispatcher bepaalt welk `commando` er uitgevoerd gaat worden. De naam van de actie is gekoppeld aan de URL, zie onderstaand fragment uit de configuratie (`web.xml`):

```
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.m</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>dispatcher</servlet-name>
  <display-name>Maverick Dispatcher</display-name>
  <servlet-class>org.infohazard.maverick.Dispatcher</servlet-class>
</servlet>
```

*web.xml, het configuratiebestand van de webapplicatie bevat de mapping die alle aanvragen met de extensie .m doorsturen naar een zogenaamde dispatcher class die bepaalt welke command class er uitgevoerd wordt.*

#### De controller

Zoals dat bij bijna elk framework het geval is, worden ook in Maverick de acties geconfigureerd in een centraal configuratiebestand. In Maverick is dit bestand `maverick.xml` genoemd. De structuur van dit bestand is eenvoudig te begrijpen, zie de code hieronder:

```
<maverick version="2.0" default-view-type="document"
  default-transform-type="document">
  <commands>
    <command name="overzicht">
      <controller class="nl.arsoftware.vastgoedonlinectl.ToonOverzicht"/>
      <view name="success" type="document" path="content/overzicht.jsp" />
      <view name="error" type="document" path="content/error.jsp" />
    </command>
  </commands>
</maverick>
```

Zoals in bovenstaande code te zien is, kunnen er commando's geconfigureerd worden. De naam van een commando is verwerkt in de URL. Aan elk commando is een class verbonden. In deze class bevindt zich de code die gestart wordt na het uitvoeren van het betreffende commando.

Een Controller is in Maverick ook meteen het model. De controller is in een class van het type `ThrowawayBean2` geïmplementeerd. Door van deze class te erven, kan de zogenaamde business code geïmplementeerd worden.

### **Het model**

Het model, dat bij Maverick versmolten is met de controller, bewaart z'n data in instance variabelen die te vullen en uit te lezen zijn met getters en setters. Als er een (HTML) formulier verzonden wordt, wordt de data door Maverick automatisch gekopieerd naar het model. Daarbij wordt er rekening gehouden met het type (String, int, double enzovoort).

### **De view**

Aan de hand van de terugwaarde van de uitgevoerde code in de controller, wordt bepaald welke view afgebeeld moet worden. Een view kan geschreven zijn in de JSP taal, maar ook met een templating taal zoals bijvoorbeeld Velocity.

## 2.2.4. WebWork framework

### Achtergrond

Ook WebWork is een open-source MVC-framework. Het begrip “KISS” is hier het hoofdmotto. “Keep It Simple Stupid”. Struts is een framework dat heel uitgebreid is, daarom hebben enkele mensen WebWork ontworpen. WebWork heeft dan ook alleen de benodigde functies om een complex webapplicatie op te bouwen.

Daarnaast heeft WebWork ook een aantal gereedschappen die gebruikt kunnen worden bij het bouwen van de applicatie. Hierdoor is de applicatie makkelijker te onderhouden. WebWork ondersteunt, net als Maverick, meerdere talen voor het afbeelden van de view.

### Installatie

De installatie van WebWork is heel eenvoudig. Het programma kan worden gedownload via de onderstaande link:

<https://webwork.dev.java.net/files/documents/693/1790/webwork-1.4.zip>

In dit zip-file zit de `webwork.jar` die je in de `WEB-INF/lib` zet van de server of van je applicatie. In principe kun je nu gebruiken maken van het framework. Ook zitten er een aantal voorbeelden van applicaties die gebruiken maken van het framework. Deze voorbeelden zijn geplaatst in de war-file. Deze war-file kun je deployen op een server (tomcat).

### Configureren

WebWork is te configureren via een aantal properties-bestanden of XML-bestanden. Via deze bestanden kan aangegeven worden hoe WebWork zich moet gedragen. Bij de standaard instellingen kijkt WebWork naar twee bestanden, namelijk `webwork.properties` en `default.properties`.

### Werking

WebWork heeft twee verschillende manieren om acties te configureren. Dit kan via een properties bestand of via een XML bestand. Na deze twee mogelijkheden bekeken te hebben, bleek een XML-bestand overzichtelijker dan een properties bestand. Bij het gebruik van XML is per actie te zien wat de gevolgen zijn.

```
<action name="vastgoed.overzicht.VastgoedOverzicht" alias="overzicht">
  <view name="success">/content_jsp/VastgoedOverzicht.jsp</view>
</action>
```

Name geeft de naam van de class die aangeroepen moet worden. En “alias” is te gebruiken om de acties te laten uitvoeren.

In een properties bestand kan in tegenstelling tot een XML bestand alles door elkaar gezet worden, waardoor het geheel niet meer overzichtelijk is.

```
overzicht.action=vastgoed.overzicht.VastgoedOverzicht
overzicht.success=/content_jsp/VastgoedOverzicht.jsp
```

---

De werking van beide bestanden zijn precies hetzelfde. Ook voor de snelheid maakt het niets uit. Dus het hangt af van de ontwikkelaar wat hij of zij makkelijkst vindt. Voor dit project wordt sowieso gekozen voor een XML bestand.

WebWork heeft een eigen bibliotheek. Om zo'n actie te definiëren moet er een class gemaakt worden die erft van de class `ActieSupport`. In deze class moeten twee methoden beschreven met de naam `doExecute` en `doValidate`. De execute wordt uitgevoerd als de actie wordt aangeroepen. Maar voordat `doExecute` wordt aangeroepen wordt eerst de `doValidate` uitgevoerd. Deze kan een aantal zelf-gedefinieerde controles uitvoeren. Als dit niet hoeft, kan men de class leeg laten. Daarna gaat de `doExecute` de acties uitvoeren die een ontwerper wil laten doen. De class geeft een string terug. Deze string moet overeenkomen met de waarde die gedefinieerd wordt in het xml bestand. Standaard wordt `success`, `input` en `error` gebruikt. Verder heeft webwork ook een aantal eigen tags. Deze tags zijn niet getest, omdat ze toch een beetje te omslachtig zijn. Er is gebruik gemaakt van JSTL (Java Standard TagLibrary), JSTL is de standaard taglib bibliotheek waarvan Apache een implementatie heeft. De bibliotheek van WebWork en JSTL hebben allebei dezelfde werking. Er wordt ook gebruik gemaakt van velocity, een templating taal van apache, zie bladzijde 15.

Dus WebWork is een simpel framework voor het opbouwen van een applicatie. Simpel en snel. Het is wel lastig te begrijpen, heb je met systeem door, wordt het bouwen van een applicatie makkelijker en overzichtelijker.

## 2.3. DE VIEW

### 2.3.1. Java Server Pages (JSP)

JavaServer Pages, kortweg JSP, is een technologie die is gebaseerd op Java. JSP wordt gebruikt om dynamische, door de server gecreëerde website te beschrijven. JSP bestanden zien er uit als HTML bestanden met daarin speciale tags die Java code kunnen bevatten.

JSP is als laag bovenop Servlet technology gebouwd. Feitelijk is een JSP een HTML pagina met extra tags ingebouwd. Als extensie voor een JSP bestand wordt doorgaans \*.jsp gebruikt ipv \*.html.

De eerste keer dat het bestand wordt aangeroepen door de webserver, wordt het JSP bestand geparst(omgezet) naar een Java Servlet bronbestand. Dit bronbestand wordt daarna gecompileerd, dit gebeurt alleen de eerste keer dat het bestand wordt aangeroepen. De volgende keren wordt de gecompileerde code gebruikt. Dit is de reden dat de eerste keer dat het bestand aangeroepen wordt, dit merkbaar langzamer gaat.

Het is mogelijk om in de JSP pagina Java Code op te nemen. Omdat JSP hier alleen gebruikt wordt als view voor het MVC model, is er afgesproken geen Java code in een JSP pagina op te nemen. Alle zogenaamde business code staat in de controller.

#### JSP 2.0

De gebruikte webcontainer Apache Tomcat heeft versie 2.0 van de Java Server Pages geïmplementeerd. Het opvallendste verschil met de vorige JSP versie is de standaard ondersteuning van de expression language EL. De expression language is een krachtige taal waarmee objecten die zich in de pagina-, request- of sessie-scope bevinden rechtstreeks benaderd en bewerkt kunnen worden.

#### Expression Language

Met behulp van de expression language, afgekort tot EL, kan data die zich in de sessie van een webserver bevindt, benaderd worden. EL is gebaseerd op een combinatie van EcmaScript en de XML Path taal (XPath). EL wordt gebruikt om objecten te benaderen en er eenvoudige acties op uit te voeren. De kracht van EL is dat het met een simpele korte commando's complexe acties uitgevoerd kunnen worden.

Het eenvoudigste is om EL te demonstreren net een aantal voorbeelden. In onderstaande code wordt vanuit de user javabean de properties firstName en lastName afgedrukt.

```
${user.firstName} ${user.lastName}
```

Geneste properties kunnen rechtstreeks aangeroepen worden. In onderstaand voorbeeld is employees een map die een bean bevat met de key "rob". Deze bean bevat een propertie "lastName".

```
${work.employees.rob.lastName}
```

Naast properties uitvragen kan EL ook rekenkundige operaties uitvoeren. Type conversie wordt automatisch toegepast indien noodzakelijk. Een aantal typen objecten zijn altijd impliciet aanwezig. Dit zijn de scoped variabelen: session, page, request. In onderstaand voorbeeld is een iets complexere expressie afgedrukt.

```
#{item.price * (1 + taxRate[user.address.zipcode])}
```

In de ontworpen applicatie wordt veelvuldig gebruik gemaakt van EL om JavaBeans aan te spreken. In de praktijk is EL een prettige en krachtige uitbreiding van JSP gebleken die de code duidelijker maakt.

## 2.3.2. Velocity

Velocity is een zogenaamde templating taal om een view voor het MVC framework te beschrijven. Het verschil met JSP is dat velocity eenvoudiger is en puur alleen voor het weergeven van content is. Een Velocity document bevat HTML en de Velocity code.

Velocity bevat een beperkt aantal keywords. In onderstaande tabel is een overzicht gegeven van alle in Velocity aanwezige "directives", een voorbeeld en een beschrijving.

<b>Directive</b>	<b>Syntax</b>	<b>Beschrijving</b>
#foreach	#foreach (\$item in \$collection) item is \$item #end	Itereert over een collectie, array, of map.
#if #else #elseif	#if (\$order.total == 0) No charge #end	Een conditie.
#parse	#parse("header.vm")	Laadt en parst het opgegeven document.
#macro	#macro(currency \$amount) #{formatter.currency(\$amount)} #end	Definieert een nieuwe directive.
#include	#include("disclaimer.txt")	Voegt een bestand in zoals het is, zonder de inhoud er van te parsen.
#set	#set (\$customer = \${order.customer})	Wijst een waarde toe aan een string in de huidige context.
#stop	#if (\$debug) #stop #end	Stop het uitvoeren van de code. (meestal voor het debuggen)

Hieronder is een triviaal voorbeeld waarin een naam wordt afgebeeld in een cel van een tabel:

```
<td>
  #if ($customer.firstName == "Rob")
    Hallo Rob Juurlink
  #else
    Hallo $customer.firstName met onbekende achternaam
  #end
</td>
```

*Een triviaal voorbeeld waarin de Velocity beschrijvingstaal toegepast wordt.*

## 2.4. PERSISTENTIE LAAG

Er is geprobeerd om de verbinding met de databank zo flexibel, eenvoudig en onderhoudbaar als mogelijk te maken. Om dit te kunnen bereiken is er een vorm van abstractie nodig. Door deze extra abstractie-laag wordt de applicatie onafhankelijk van de gebruikte database en wordt voorkomen dat er (onoverzichtelijke) SQL databank-code rechtstreeks in de applicatie komt te staan.

De data wordt vanuit de database op objecten afgebeeld volgens het DAO pattern. Er bestaan open-source bibliotheken die DAO geïmplementeerd hebben. Er is een bestaande volwassen framework met bibliotheek van iBATIS.

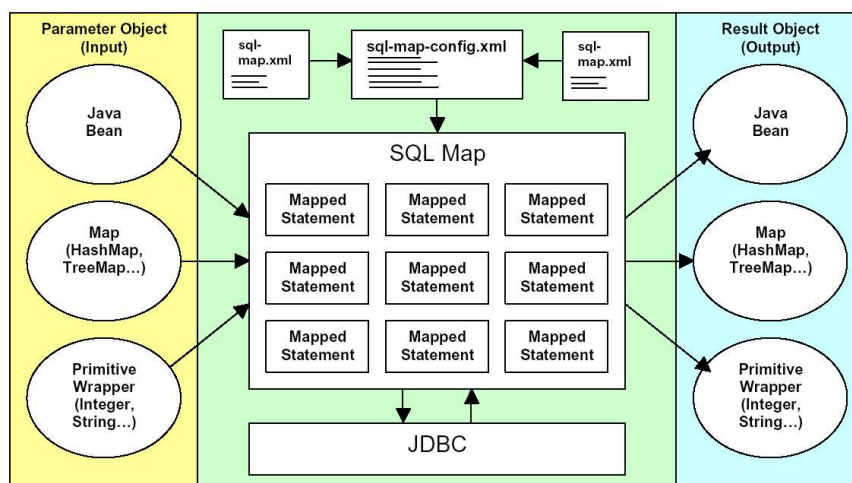
### 2.4.1. iBatis SQL Maps

SQL Maps is een framework dat als doel heeft om de hoeveelheid code die nodig is om een databank aan te spreken, te beperken. Het framework koppelt standaard JavaBean objecten aan SQL statements door middel van een eenvoudig XML configuratiebestand. Uitgangspunt van SQL Maps is eenvoudigheid.

#### Werking SQL Maps

In figuur 3 zijn de verschillende stappen weergegeven die uitgevoerd dienen te worden om objectgegevens uit te vragen.

1. Bied een object aan als parameter. Deze parameter wordt gebruikt om de waarden in een SQL update statement of de waarden in een SQL WHERE regel te zetten.
2. Voer het statement uit. Het SQL Maps framework creëert een `PreparedStatement` instantie en stelt de parameters in door gebruik te maken van het object dat we mee gegeven hebben. Voer het statement uit.
3. Als er een update uitgevoerd is, wordt het aantal tabelrijen dat veranderd is als een getal teruggegeven. In het geval van een `query` komt het resultaat terug. Dit resultaat is een JavaBean, een Map of een ander soort object.



Figuur 3, een schematisch overzicht van de stappen die uitgevoerd dienen te worden om een update uit te voeren of om data op te vragen. Er wordt gebruik gemaakt van SQL Maps.

## Een voorbeeld

In onderstaande code wordt in XML aangegeven hoe een JavaBean van het type VastgoedObject gekoppeld is aan de overeenkomstige data in de databank.

```
<select id="getVastgoedObject" resultClass="VastgoedObject">
    SELECT
        id,
        objecttype,
        straatnaam,
        straatnummer,
        straatbijvoegsel,
        plaatsnaam,
        postcode,
        vraagprijs,
        status,
        categorie,
        bouwjaar,
        aantalKamers,
        perceeloppervlakte
    FROM vastgoedobjecten
    WHERE id = #value#;
</select>
```

*In een XML configuratiebestand wordt beschreven hoe een JavaBean (VastgoedObject) z'n data is vastgelegd in de databank.*

Om nu de objectgegevens uit de databank op te vragen, moet de onderstaande code uitgevoerd worden:

```
Integer id = new Integer(5);
VastgoedObject object = (VastgoedObject) sqlMap.executeQueryForObject
    ("getVastgoedObject", id);
```

*In de bovenstaande code wordt een JavaBean gevuld met de gegevens uit de databank.*

Het is ook mogelijk om meer geavanceerde opties in te stellen zoals transactie management, cache en lazyloading<sup>1</sup>, maar zoals eigenlijk bijna altijd het geval is, is het beter om eenvoudig te beginnen en de configuratie en code uit te werken naar een meer geavanceerde variant in de toekomst als dat nodig mocht blijken.

Als de instellingen zoals de gebruikte databank, het cache model enz. later gewijzigd worden, hoeft daarvoor de Java code niet gewijzigd te worden.

<sup>1</sup> Nog niet alle data voor een JavaBean wordt uitgevraagd uit de databank, maar pas op het moment dat de waarde daadwerkelijk uitgelezen wordt, wordt de databank query uitgevoerd.

## 2.4.2. Apache data-sources

Om in een webapplicatie een databank te kunnen benaderen, kan de databank het beste globaal voor de applicatieserver geconfigureerd worden. Dit kan door in Apache een `datasource` te configureren. De referentie naar deze `datasource` is in een webapplicatie op te vragen door middel van de JNDI naam. In het onderstaande voorbeeld is de JNDI naam "jdbc/VastgoedOnlineDB".

Er wordt vanuit gegaan dat de JDBC databank driver in het classpath van de applicatieserver aanwezig is en dat de databank opgestart is en te benaderen via een TCP verbinding. In het geval van MySQL moet het JAR archief `mysql-connector-java-3.0.11-stable-bin.jar`

### server.xml

Om de databank verbindingen te creëren en beschikbaar te stellen in Apache Tomcat 5, moet de volgende code toegevoegd worden aan de configuratie in `conf/server.xml`.

```
<!-- De MySQL databank instellen. -->
<Context path="/maverick" docBase="webapp"
    debug="5" reloadable="true" crossContext="true">

    <Resource name="jdbc/VastgoedOnlineDB"
        auth="Container"
        type="javax.sql.DataSource"/>

    <ResourceParams name="jdbc/VastgoedOnlineDB">
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>

<!-- MySQL dB username and password for dB connections -->
        <parameter>
            <name>username</name>
            <value>java</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>javal23</value>
        </parameter>

<!-- Class name for the official MySQL Connector/J driver -->
        <parameter>
            <name>driverClassName</name>
            <value>com.mysql.jdbc.Driver</value>
        </parameter>

<!-- The JDBC connection url for connecting to your MySQL dB -->
        <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost:3306/vastgoedonline</value>
        </parameter>
    </ResourceParams>
</Context>
```

*Een databank configureren en data-source voor JNDI instellen in server.xml. Het configuratiebestand van Apache Tomcat.*

---

Nadat de referentie naar de databank via JNDI beschikbaar gesteld is, kan deze uitgevraagd en gebruikt worden door SQL Maps. De data-source is als volgt te configureren:

```
<transactionManager type="JDBC">
  <dataSource type="JNDI">
    <property name="DBFullJndiContext"
      value="java:comp/env/jdbc/VastgoedOnlineDB"/>
  </dataSource>
</transactionManager>
```

*Een data-source configureren in SQL Maps*

## 2.5. CACHING

Een Cache is een korte termijn geheugen dat snel toegankelijk is.

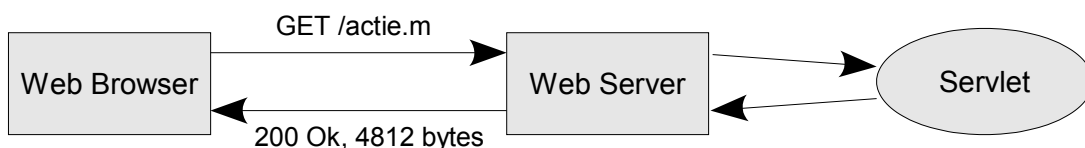
Wanneer informatie opgevraagd wordt van een langzamer medium en het bekend is dat die informatie enige tijd niet zal veranderen, kan het resultaat onthouden worden in cache geheugen. Bij een volgende aanroep wordt de versie uit het cache opgehaald, het langzamere medium hoeft niet aangesproken te worden. Dit principe heet caching.

Ook webbrowsers maken gebruik van deze techniek: Als een website in de browser getoond wordt, worden de afgebeelde pagina's inclusief de plaatjes ook opgeslagen in het lokale geheugen. Als een eerder opgevraagde pagina daarna nogmaals opgevraagd wordt, kan de versie die in het geheugen van de webbrowser aanwezig is meteen weergegeven worden. De browser hoeft deze keer de pagina niet nogmaals vanaf het langzamere internet in te laden.

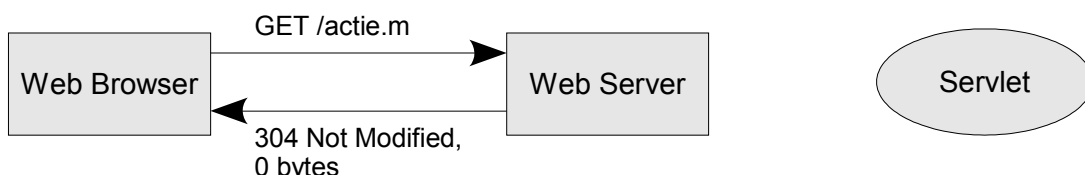
Met speciale HTTP Header velden kan de webserver per pagina aangeven hoelang deze 'houdbaar' is en wanneer de browser verplicht is te controleren of er een nieuwe versie van het document op de server aanwezig is.

Naast de webbrowser zou ook een proxyserver die zich tussen de gebruiker en de applicatieserver bevindt, gebruik kunnen maken van de mogelijkheden van HTTP Cache.

In figuur 4 en 5 is weergegeven hoe het principe van caching werkt in de browser.



*Figuur 4, een browser roept voor het eerst de actie.m pagina op. De server delegeert de actie naar de Servlet die de dynamische pagina genereert en terugstuurt via de webserver naar de browser. Het ontvangen document in het antwoord is 4812 bytes lang.*



*Figuur 5, een browser roept voor de tweede maal de actie.m pagina op. De server controleert en ziet dat de dynamische pagina niet gewijzigd is en stuurt een antwoord terug dat het document niet gewijzigd is. Er hoeft nu geen document gegenereerd te worden door de Servlet en het ontvangen antwoord is kort omdat het geen document bevat.*

Door gebruik te maken van deze mogelijkheid die het HTTP protocol biedt, kan door twee soorten caches voorkomen worden dat de webserver onnodig aangesproken wordt, te weten de **browser cache** en de **proxy cache**. Het resultaat van dit alles is dat de webserver en het netwerk uiteindelijk minder zwaar belast worden en dat de webapplicatie of website sneller z'n pagina's weergeeft.

Standaard is de mogelijkheid van HTTP caching niet geïmplementeerd in de bij ons bekende MVC frameworken Struts, Maverick en WebWork. Echter door de code uit te breiden(de frameworken zijn open-source) is het toch mogelijk gebruik te maken van de mogelijkheden van HTTP Cache.

## 2.5.1. Cache Control HTTP Headers

In de HTTP 1.0 en 1.1 specificaties zijn zogenaamde Cache Control Headers gedefinieerd. In HTTP versie 1.0 bestaat de `Expires` header en in HTTP versie 1.1 wordt het cache gedrag ingesteld met de header `Cache-Control`.

Een pagina zonder de hierboven genoemde headers wordt meestal niet in de cache opgeslagen. Elke aanvraag van de browser wordt dan door de server opnieuw verwerkt en de inhoud verstuurd. Als de cache niet ingesteld wordt door middel van de cache headers, zijn we voor de standaard instelling van het cache gedrag afhankelijk van de gebruikte browser.

### HTTP 1.0 Expires header

In HTTP 1.0 wordt door middel van de `Expires` header een GMT<sup>2</sup> tijd ingesteld. Deze header bevat een datum en een tijd die aangeven wanneer het document niet meer "houdbaar" is en geactualiseerd dient te worden. De `Expires` HTTP 1.0 header ziet er als volgt uit:

```
Expires: Mon, 12 Apr 2004 17:35:42 GMT
```

Om te voorkomen dat de browser de data lokaal in de cache van de browser opslaat, kan de datum in het verleden gezet worden. Een waarde van -1 wordt volgens de specificatie ook gezien als een datum uit het verleden.

### HTTP 1.1 Cache-Control header

HTTP 1.1 kent de `Cache-Control` header. Deze header biedt meer mogelijkheden. Er zijn meerdere parameters voor deze header mogelijk die individueel of in combinatie met elkaar gebruikt kunnen worden.

- `max-age`. Geeft het aantal seconden aan dat de pagina niet geactualiseerd hoeft te worden. In deze tijd hoeft er geen contact met de server plaats te vinden om te controleren of de pagina veranderd of ververs is.
- `no-cache`. Geeft aan dat een proxyserver bij elke aanroep van een browser moet controleren of de inhoud van de originele pagina gewijzigd is. De proxyserver hoeft niet de complete pagina opnieuw in te laden om in z'n cache te zetten.
- `must-revalidate`. Dwingt de proxyserver om de pagina te controleren op een gewijzigde versie nadat de inhoud verlopen is. Wordt deze parameter niet meegegeven, dan kan de proxyserver zelf gaan beslissen of het wil controleren of de pagina gewijzigd is.
- `private`. De inhoud van de pagina is bestemd voor één enkele gebruiker en dient daarom niet opgeslagen te worden door de publieke proxyserver. De pagina mag wel in de cache bewaard worden van de browser van de gebruiker.

<sup>2</sup> GMT is de afkorting van Greenwich Meridian Time ook wel: Greenwich Mean Time. GMT is één van de tijd standaarden. Greenwich is een klein plaatsje ten zuiden van Londen in Engeland en ligt op 0° oosterbreedte. Het moment dat in Greenwich de zon op het hoogste punt staat (in het zuiden), heet twaalf uur. De aanduiding is dan 12:00:00 GMT.

## Valideren van een verlopen pagina - HTTP 1.0

De cache kan volgens de specificaties van HTTP versie 1.0 controleren of een pagina gewijzigd is door met een nieuwe aanvraag (een HTTP request) een `If-Modified-Since` header mee te sturen. De server kan dan antwoorden dat de inhoud van de pagina niet gewijzigd is. De aanvraag van de browser en het antwoord van de server zien er dan als volgt uit: (niet relevante headers zijn in onderstaande voorbeeld niet afgedrukt)

```
GET /index.html HTTP/1.0
If-Modified-Since: Thu, 08 Apr 2004 12:38:16 GMT
```

```
HTTP/1.1 304 Not Modified
Last-Modified: Thu, 08 Apr 2004 12:38:16 GMT
Date: Wed, 08 Apr 2004 22:03:30 GMT
```

*Een HTTP 1.0 aanvraag waarin aangegeven wordt door de browser wat de versie van document in de lokale cache is (boven). Het antwoord dat terug komt van de server geeft aan dat het document niet gewijzigd is (onder).*

Als een webserver een mogelijkheid heeft om snel te kunnen controleren of de inhoud van een pagina gewijzigd is of om snel de datum en tijd van het document te bepalen, biedt het gebruik van cache een enorme snelheidswinst. Daarbij wordt ook nog eens de hoeveelheid netwerkverkeer verkleind en de server minder belast.

In de praktijk is caching alleen mogelijk bij gebruik van HTTP GET aanroepen. Bij HTTP POST aanroepen blijft de URL vaak hetzelfde, terwijl de inhoud van opgeroepen document (meestal) anders is.

Het is ook mogelijk om het cache gedrag van de browser te bepalen door speciale tags in de HTML op te nemen, echter hebben cache instellingen in de header voorrang boven de cache instellingen die in de HTML van het document zelf staan. Daarnaast lezen proxyservers de inhoud van een document niet, waardoor alleen de Cache headers invloed hebben op het cache gedrag van een proxyserver.

## Valideren van een verlopen pagina - HTTP 1.1

De HTTP 1.1 specificatie beschrijft een nieuwe methode om te controleren of een bepaalde pagina nog "vers" is. Hiervoor kan in HTTP versie 1.1 de `ETag` in combinatie met de `If-None-Match` header gebruikt worden.

Een `ETag` bevat een code die moet wijzigen als de inhoud van een document veranderd is. Hoe deze code er uit moet zien, is niet vastgelegd in de specificatie, als deze code maar wijzigt. Door middel van deze `ETag` bepaalt de webbrowser, die deze code van de webserver ontvangt, of de versie van de pagina in z'n eigen cache nog gelijk is aan de versie op de webserver.

Als in de HTML de beide headers `If-None-Match` en `If-Modified-Since` tegelijk voorkomen, heeft de (nieuwere) `If-None-Match` header voorrang boven de (oudere) `If-Modified-Since` header.

De aanvraag van de browser en het antwoord van de server zien er bij het gebruik van de Etag voor de validatie dan als volgt uit: (niet relevante headers zijn in onderstaande voorbeeld niet afgedrukt)

```
GET /maverick/details.m?objectId=2 HTTP/1.0
If-None-Match: date-1081461844000
```

```
HTTP/1.1 304 Not Modified
Last-Modified: Thu, 08 Apr 2004 22:04:04 GMT
ETag: date-1081461844000
Date: Thu, 08 Apr 2004 22:04:51 GMT
```

*Een HTTP 1.1 aanvraag waarin met een unieke code, de zogenaamde ETag, aangegeven wordt wat de versie van het document in de lokale cache is (boven). Het antwoord dat terug komt van de server geeft aan dat het document niet gewijzigd is (onder).*

## 2.5.2. Aanpassingen aan het MVC Framework

In een Servlet is het voldoende om de standaard methode `getLastModified()` te "overriden". In de standaard versie geeft deze methode als waarde een `-1` terug. Een waarde van `-1` heeft als resultaat dat de inhoud van een pagina altijd opnieuw gegenereerd en verzonden wordt. Bij een andere waarde wordt gecontroleerd of de versie van de browser ouder (HTTP1.0) of anders (HTTP1.1) is.

### **getLastModified()**

Om te kunnen bepalen of een op te roepen pagina is gewijzigd, roept de webcontainer de methode `getLastModified(HttpServletRequest request)` van de class `HttpServletRequest` aan.

De methode is `protected`, dus kan het overschreven worden door eigen code die erft van de `HttpServletRequest` class.

Om de methode werkend te maken, moet de implementatie er van voldoen aan de volgende punten:

- De code die bepaalt wanneer de pagina gewijzigd is, moet snel uit te voeren zijn. Als deze code er net zo lang over doet als de tijd die nodig is om een nieuwe inhoud te genereren, werkt het gebruik van cache functionaliteit alleen maar vertragend.
- De waarde van de methode mag alleen wijzigen als de te tonen data gewijzigd is.
- Er moet geen authenticatie proces aan de gang zijn. In zo'n geval is het gebruik van cache alleen maar storend.

### **Aanpassingen aan de controller/actie**

We willen graag per actie of controller kunnen achterhalen wanneer de data voor het laatst gewijzigd is, daarom moet er per controller een methode komen die bepaalt wanneer de data voor de als parameter meegegeven aanvraag voor het laatst gewijzigd is.

In de frameworken wordt de standaard code van een Servlet niet gebruikt, daarom is de afhandeling van cache zelf geïmplementeerd in de code.

Als de browser een aanroep doet, komt de vraag wanneer de pagina voor het laatst gewijzigd is, als eerst bij de dispatcher van het MVC framework uit. De dispatcher kan niet bepalen wanneer de laatste wijziging was en moet daarom de aanvraag doorsturen naar een controller. Om te bepalen naar welke controller de vraag gedelegeerd dient te worden, moet de dispatcher uitzoeken welke actie aan welke controller gekoppeld is.

Als de dispatcher weet welke actie aan welke controller gekoppeld is, weet het ook aan welke controller het de vraag moet stellen wanneer de data van de request voor het laatst gewijzigd is.

## Cache instellen

Om eenvoudig aan te kunnen geven hoelang een bepaalde pagina maximaal in de cache van de proxy of van de browser bewaard mag worden, is er een speciale methode gemaakt.

De parameter `pSeconds` geeft het aantal seconden aan dat de browser niet bij de server hoeft te controleren of het document gewijzigd is. `pMustRevalidate` geeft aan dat een browser wel of niet moet valideren bij de server of een document gewijzigd is.

Deze methode die wordt aangeroepen voordat de view afgebeeld wordt, ziet er als volgt uit:

```
/**
 * De HTTP Cache Control headers instellen.
 *
 * @param pResponse De response via welke de headers benaderd worden.
 * @param pSeconds Het aantal seconden dat de inhoud "vers" is.
 * @param pMustRevalidate Verplicht controleren op wijziging van de pagina
 * na verlopen van de datum?
 */
public static void cacheForSeconds(HttpServletRequestResponse pResponse,
    int pSeconds, boolean pMustRevalidate) {

    String lVal = "max-age=" + pSeconds;
    if (pMustRevalidate) {
        lVal += ", must-revalidate";
    }
    pResponse.setHeader("Cache-Control", lVal);
    pResponse.setDateHeader("Expires", System.currentTimeMillis() +
        pSeconds * 1000L);
}
```

*Een methode waarin met HTTP 1.0 en HTTP 1.1 header velden wordt aangegeven hoelang een document door de browser ge-"cache"-d mag worden. Daarnaast is aan te geven of de browser na het verlopen van de houdbaarheid van een pagina bij een volgende aanroep verplicht is te controleren of het document op de server gewijzigd is.*

### 2.5.3. Aanbevelingen bij gebruik van cache

Naast het gebruik van een “houdbaarheidsdatum” en validatie, zijn er een aantal andere belangrijke punten die een webapplicatie cache vriendelijker maken.

- **Consistente URL's** – Dit is eigenlijk het belangrijkste onderdeel om cache goed te laten werken. Een bepaalde URL moet altijd dezelfde pagina opleveren. Caching heeft geen nut als een bepaalde inhoud elke keer een andere URL oplevert. HTTP Get parameters zijn ook onderdeel van een URL.
- **Plaatjes op een vaste plaats** – Gebruik voor de plaatjes een standaard bestandsmap en refereer door de gehele webapplicatie voor de plaatjes altijd naar deze map. Dit heeft als resultaat dat een plaatje slechts eenmaal geladen hoeft te worden door de browser.
- **Onderdelen die praktisch nooit wijzigen krijgen een hoge max-age** – Plaatjes en inhoud waarvan van te voren bekend is dat ze nauwelijks wijzigen, kunnen een hoge waarde meekrijgen voor de `max-age` Cache Control. De `max-age` zorgt er voor dat de browser voor een bepaalde tijd niet gaat controleren of het gewijzigd is.
- **Sessies alleen als het echt nodig is** – Het gebruik van sessies breekt vaak de cache doordat de URL bij elke aanroep anders lijkt. Als een pagina ook zonder sessies kan, de sessie uitschakelen.

### 2.5.4. Prefetching

Met “link prefetching” wordt bedoeld het downloaden van bepaalde documenten door de browser tijdens de tijd dat de browser niets te doen heeft (de `idle` tijd). Dit zijn de documenten die de gebruiker in de toekomst zou kunnen bezoeken. De data wordt opgeslagen in de cache van de browser.

In een web pagina kan voor een verwijzing een extra attribuut aan de tag toegevoegd worden die aangeeft dat het document gedownload mag worden tijdens de tijd dat de browser verder niets te doen heeft. Mocht een gebruiker nu even later beslissen wel de verwijzing aan te klikken, dan kan deze verwijzing meteen uit de cache getoond worden.

Een fragment van de HTML met daarin een prefetch verwijzing:

```
<link rel="prefetch" href="/images/big.jpeg">
```

*In het bovenste voorbeeld is een verwijzing naar een groot plaatje opgenomen. Het attribuut `rel="prefetch"` geeft aan dat het plaatje al ingeladen mag worden in de cache voordat het bezocht is.*

```
Link: </images/big.jpeg>; rel=prefetch
```

*Het is ook mogelijk in een HTTP Header aan te geven welke plaatjes voordat ze bezocht zijn alvast ingeladen mogen worden in de cache.*

Prefetching is niet toegepast door ons, maar omdat het een nuttige techniek lijkt die eventueel toegepast kan worden in de toekomst, wordt het hier beschreven. Prefetching wordt op dit moment alleen ondersteund door op Mozilla gebaseerde browsers.

## 3. ONTWERP

### 3.1. PROBLEEMSTELLING

Om het ontwerp duidelijk te houden en de verantwoordelijkheden tussen de verschillende delen van de code goed te scheiden wordt ge-eist dat er een standaard applicatie framework gebruikt wordt.

Welk framework er gebruikt gaat worden, staat nog niet vast. Er zijn een aantal mogelijkheden. In een vooronderzoek wordt bepaald welk framework het meest geschikt is.

Het kiezen van een framework is het onderdeel dat door de opdrachtgever niet uitgebreid beschreven is. Voor de ontwikkelaars is het het belangrijkste dat er een goed framework gebruikt wordt. De uitwerking van deze deel-opdracht wordt niet bepaald door de opdrachtgever, maar door de ontwikkelaar zelf.

### 3.2. ONTWERP BESLISSINGEN

Voordat er gestart is met het ontwerpen en de implementatie, heeft er natuurlijk een vooronderzoek plaatsgevonden om te kijken welke framework het beste gebruik kan worden en welke manier van data opslag het beste is.

Hoe de structuur van een webapplicatie er uit moet zien is niet vastgelegd in de J2EE specificatie. Wat wel is vastgelegd is hoe data opgeslagen kan worden. Er wordt geadviseerd om voor de data opslag gebruik te maken Enterprise JavaBeans. Echter in theorie klinkt de techniek hierachter heel mooi en duidelijk, in de praktijk valt dat helaas wat tegen, vandaar dat voor de opslag van data een andere keuze gemaakt is.

#### 3.2.1. CMP versus SQL Maps

Tijdens het vooronderzoek is gebleken dat J2EE een mooie standaard is, maar dat het op dit moment nog een standaard is die aan veel veranderingen onderhevig is. De grootste ontwerpfout in J2EE's leek wel het gebruik van Enterprise JavaBeans en dan met name de specificatie van voor J2EE1.3, waar Enterprise JavaBeans alleen remote aangesproken konden worden.

In theorie was dit een mooi ontwerp, maar in de praktijk bleek het niet te werken. Het maakte de applicatie complexer en het feit dat Enterprise JavaBeans alleen konden communiceren door middel van geserialiseerde objecten, kwam de performance zeker niet ten goede.

In latere specificatie is het probleem van slechte performance met Enterprise JavaBeans opgelost door het lokaal aanroepen van EJB's toe te staan.

Het probleem met performance in combinatie met Enterprise JavaBeans is daarmee opgelost, alleen kan nu afgevraagd worden wat nog de toegevoegde waarde van het gebruik van EJB's is. Het enige wat we nu kunnen bedenken is de zogenaamde Container Managed Persistence, de communicatie met een database laten afhandelen door de EJB container en de ondersteuning van transacties.

Voor communicatie met een database zijn er tegenwoordig ook een aantal andere oplossingen in ontwikkeling. Een bekende methode is “data mapping”. SQL Maps van iBatis<sup>3</sup> bijvoorbeeld is een bibliotheek die deze functionaliteit implementeert.

## 3.3. ANALYSE

Door de twee wekelijkse iteraties is het gehele systeem in overzichtelijke deels losstaande brokken opgedeeld. Door dit opdelen van de applicatie in kleinere delen wordt bereikt dat de analyse eenvoudiger is.

gedurende het onderzoek wordt er een test applicatie ontwikkeld. Deze test applicatie die in verschillende frameworken gerealiseerd wordt, moet vergelijkbaar zijn, daarom wordt er een eenvoudige test applicatie bedacht, die tijdens het project verder uitgewerkt kan worden om ook echt bruikbaar te zijn in de afstudeeropdracht.

De verwoording van deze test applicatie ziet er als volgt uit:

### 3.3.1. Verwoording

Toon een lijst met objecten op het scherm. Deze objectdata komt uit een databank. Er hoeven voor deze test van het framework slechts vier objecten in de databank aanwezig te zijn. Bij elk afgebeeld object is een verwijzing aanwezig naar een pagina waarop de details van het object afgedrukt worden.

In het lijstoverzicht moet per object een adres (straatnaam, -nummer, -bijvoegsel, postcode en plaatsnaam), een titelfoto(de eerste foto in de lijst), een objecttype, een vraagprijs, een status en een verwijzing afgedrukt worden. De vraagprijs is een geheel getal die het aantal Euro's aangeeft. Een straatnaam, plaatsnaam en de status zijn verplicht. De data komt uit de databank, behalve de foto zelf, die op een door een webbrowser te bereiken locatie staat. De naam van de foto komt wel uit de databank. De status vraagprijs en de status zijn beide teksten.

Het detailoverzicht bevat dezelfde gegevens als het lijstoverzicht. Daarnaast bevat het detailoverzicht extra gegevens: categorie, bouwjaar, totaal aantal kamers en perceeloppervlakte en naast de titelfoto extra foto's. Deze gegevens zijn niet verplicht. Er is geen maximum aan het aantal foto's dat toegevoegd kan worden. De foto's hebben een volgorde en een beschrijving. Van deze extra foto's wordt de preview versie afgedrukt. Er is een verwijzing naar de grotere versie van de foto.

Aan elk object is een id gekoppeld. Dit id is een nummer. Welk nummer aan welk object gekoppeld is, wordt bepaald door de VastgoedOnline client en ligt dus al vast. Het id is een verplicht veld.

Er moet een zoekscherm aanwezig zijn. In dit zoekscherm moet een zoekopdracht opgegeven kunnen worden. Zoeken op vraagprijs, type, categorie en plaatsnaam. De vraagprijs wordt opgegeven in twee velden (van t/m). De prijs hoeft niet getypt te worden, maar wordt geselecteerd uit een dropdown. Ook type en categorie kunnen uit een lijst geselecteerd worden. De functionaliteit waarin de zoekopdracht uitgevoerd wordt, hoeft in deze testapplicatie niet aanwezig te zijn.

---

3 SQL Maps van iBATIS. Open-source bibliotheek voor het koppelen van Java objecten met een relationele database. <http://www.ibatis.com>.

Hoe de gegevens in de database terecht gekomen zijn, is even niet van belang binnen dit onderzoek. Ook de data voor de foto's voor in het lijst- en detailoverzicht bevinden zich al op de server.

Voor de naamgeving van de foto's is het volgende afgesproken: In de databank wordt de originele naam van een foto bewaard. De kleinste preview versie van de foto heeft als toevoeging aan het einde “\_preview”. De middelste en de grootste versie respectievelijk “\_middel” en “\_groot”. Van een foto met de naam “id34521.jpg” heet de kleinste versie dus “id34521\_preview.jpg”.

De opmaak van de pagina doet er niet toe binnen dit onderzoek, eenvoudige HTML om de werking te demonstreren is voldoende.

Deze test applicatie wordt geïmplementeerd volgens de richtlijnen van het gebruikte framework. De data wordt niet rechtstreeks uit de database gehaald door SQL in de code op te nemen, maar wordt via een speciaal design pattern daarvoor op een object afgebeeld.

### 3.3.2. Uitwerking

#### Kernzinnen

- Een object heeft een id
- Een object heeft een objecttype
- Een object heeft een adres
- Een adres heeft een straatnaam
- Een adres heeft een huisnummer
- Een adres heeft een huisnummer bijvoegsel
- Een adres heeft een plaatsnaam
- Een adres heeft een postcode
- Aan een object zijn foto's gekoppeld
- Een foto heeft een naam
- Een foto heeft een beschrijving
- Een foto heeft een volgorde
- Een object heeft een vraagprijs
- Een object heeft een status
- Een object heeft een categorie
- Een object heeft een bouwjaar
- Een object heeft een totaal aantal kamers
- Een object heeft een perceeloppervlakte

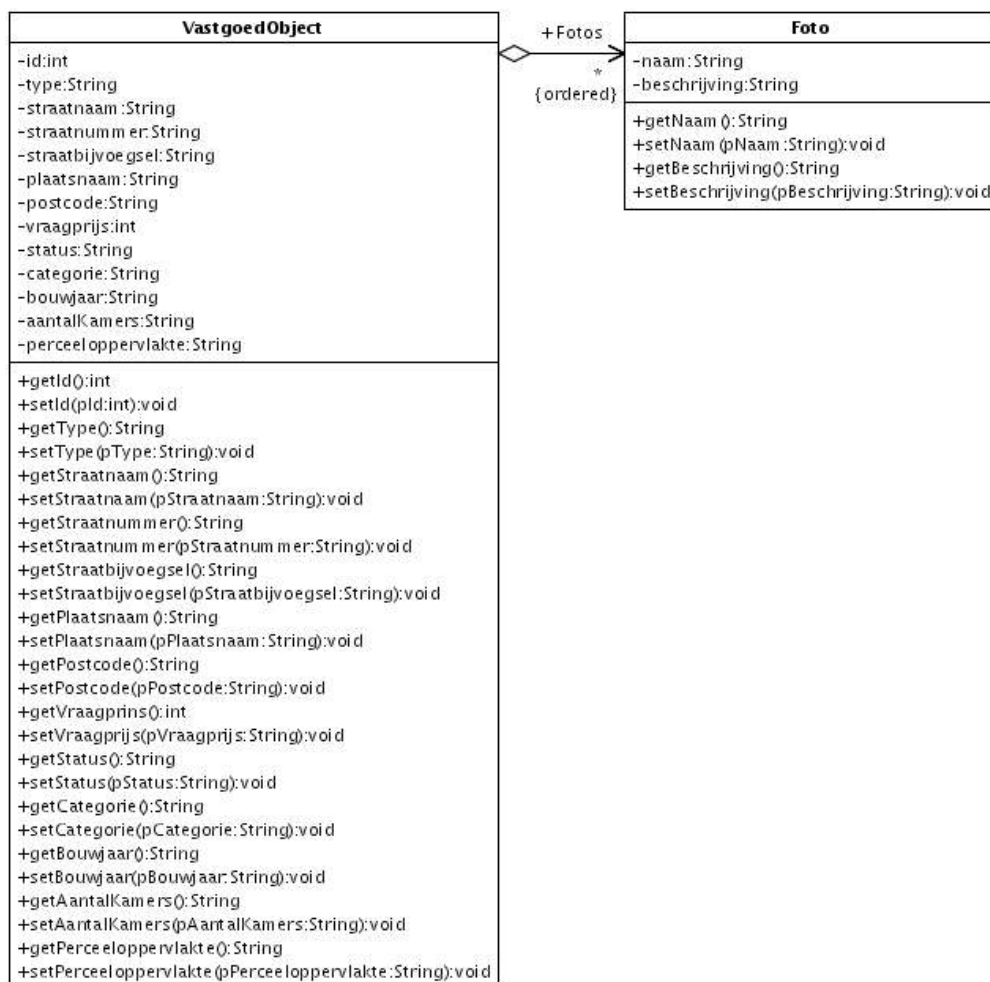
## Datamodel

In het datamodel worden de gegevens zoveel als mogelijk samengevoegd in één object. Door dit samenvoegen kan voor het verzenden van de gegevens hetzelfde object gebruikt worden. Bij het verzenden is het van belang dat dit in zo min mogelijk stappen en zo groot mogelijke brokken gebeurt.

Aan een object kan slechts één adres gekoppeld zijn. De adresgegevens die uit teksten (Strings) bestaan, worden daarom rechtstreeks in het object opgenomen. Dit scheelt straks bij het benaderen van de gegevens in de database ook een extra aanroep.

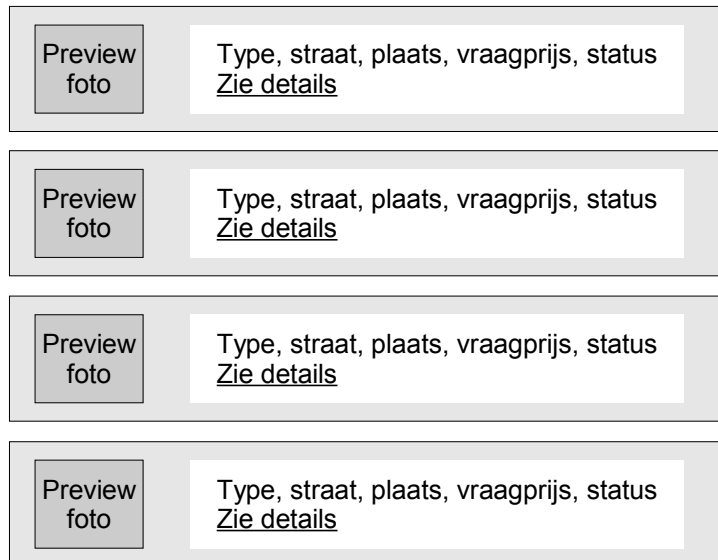
## Classediagram

De uitwerking van het datamodel heeft de volgende classes opgeleverd. De tabellen in de databank zijn een één op één afspiegeling van objectclasses. Ook de naamgeving van de velden is hetzelfde gehouden.

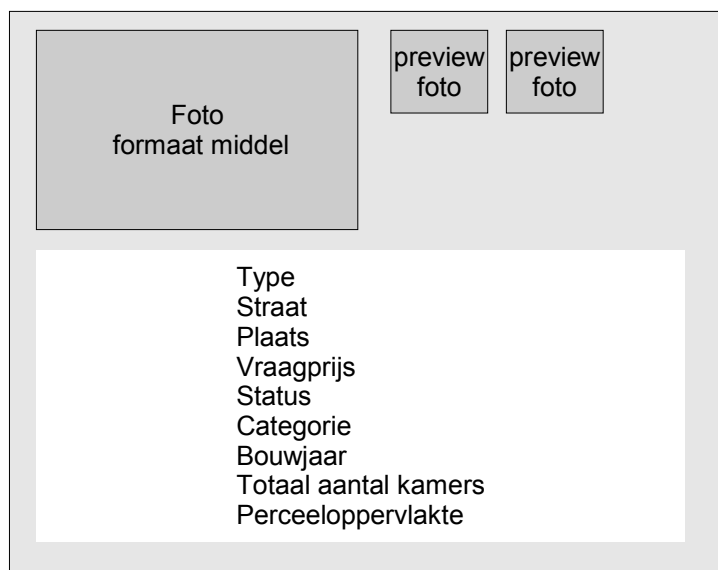


## Applicatieschermen

Als we de verwoording uitwerken blijkt dat de webapplicatie uit drie te onderscheiden schermen bestaat. Deze schermen zien er schematisch gezien als volgt uit. Aan het zoekscherm is nog geen functionaliteit gekoppeld. Het gaat er in dit eenvoudige ontwerp om dat de data van het formulier verwerkt kan worden door het framework.



Figuur 6, schematische weergave van het lijstoverzicht



Figuur 7, schematische weergave detailweergave

---

Vraagprijs:  ▼  ▼

Type:  ▼

Categorie:  ▼

Plaats:

*Figuur 8, het scherm waarin de zoekopdracht opgegeven kan worden.*

## 4. TESTEN

### 4.1. PERFORMANCE TEST

Een niet onbelangrijk onderdeel is de performance van een framework. Om de performance goed te kunnen testen zijn er zoals eerder in dit document verteld drie verschillende implementaties gemaakt van dezelfde applicatie. Bij elke implementatie is een ander framework en/of view gebruikt.

In de praktijk is er nauwelijks verschil in snelheid tussen de verschillende frameworken. Onderstaand overzicht toont de testresultaten. Bij deze test zijn 10 gelijktijdige gebruikers gesimuleerd die allemaal 100 aanroepen uitvoeren.

De testen zijn een aantal maal uitgevoerd en daarvan is de gemiddelde waarde genoteerd. De uitvoer van deze testen ziet er als volgt uit.

Zoals te zien in de uitvoer duurt het verwerken van 1000 aanroepen zo'n 8 seconden. Tijdens het verwerken is dan een lijst afgebeeld. Het detailscherm doet er zo'n 5 seconden over om 1000 keer gegenereerd en afgebeeld te worden.

```

Overzicht:
=====
JSTL + JSP (Webwork)

Transactions:                1000 hits
Availability:                100.00 %
Elapsed time:                 7.69 secs
Data transferred:           1955000 bytes
Response time:               0.07 secs
Transaction rate:            130.04 trans/sec
Throughput:                  254226.27 bytes/sec
Concurrency:                  9.04

Velocity (Webwork)

Transactions:                1000 hits
Availability:                100.00 %
Elapsed time:                 8.44 secs
Data transferred:           1701000 bytes
Response time:               0.08 secs
Transaction rate:            118.48 trans/sec
Throughput:                  201540.29 bytes/sec
Concurrency:                  9.41

JSTL (maverick)

Transactions:                1000 hits
Availability:                100.00 %
Elapsed time:                 4.55 secs
Data transferred:           3118000 bytes
Response time:               0.04 secs
Transaction rate:            219.78 trans/sec
Throughput:                  685274.70 bytes/sec
Concurrency:                  9.47

```

```

Successful transactions:      1000
Failed transactions:         0

Detail:
=====
JSTL + JSP

Transactions:                1000 hits
Availability:                100.00 %
Elapsed time:                5.32 secs
Data transferred:           1655000 bytes
Response time:              0.01 secs
Transaction rate:           187.97 trans/sec
Throughput:                  311090.22 bytes/sec
Concurrency:                 1.00

Velocity

Transactions:                1000 hits
Availability:                100.00 %
Elapsed time:                5.29 secs
Data transferred:           2791404 bytes
Response time:              0.01 secs
Transaction rate:           189.04 trans/sec
Throughput:                  527675.62 bytes/sec
Concurrency:                 0.99

```

## 4.2. TESTPLAN EN TESTRAPPORT

In dit hoofdstuk zijn de eisen en wensen uitgewerkt in een aantal zogenaamde test cases. Deze test cases zijn individueel testbaar. Alle test cases hebben betrekking op de eis met nummer 13; Standaard framework.

### 4.2.1. Framework

Bij de verschillende versies van de testapplicatie ging het niet om de functionaliteit, maar om te kunnen ervaren welk framework het prettigste werkte. Daarnaast ook een stuk performance test. De resultaten daarvan zijn een hoofdstuk hierboven afgebeeld.

### 4.2.2. Data persistentie

Er wordt getest of een nieuw object opgeslagen wordt. Het object daarna opgevraagd kan worden en of de gegevens van een object gewijzigd kunnen worden.

Een opgevraagd object moet ook die namen van de gekoppelde foto's bevatten. Fysiek bevinden deze namen van foto's zich in een andere tabel.

<b>Beschrijving</b>	<b>Instructies</b>	<b>Verwachte uitvoer</b>	<b>Check</b>
Schrijven	SQLTest: Maak een nieuw object aan en schrijf deze weg via SQL Maps.	Alle data van het object staat in de databank.	√

<i>Beschrijving</i>	<i>Instructies</i>	<i>Verwachte uitvoer</i>	<i>Check</i>
Opvragen	Vraag aan de hand van een id het object uit. Roep de toString methode aan.	Alle data die ook in de databank aanwezig is wordt afgedrukt.	√
Wijzigen	Vraag een object uit, wijzig de straatnaam en plaatsnaam en plaats het object opnieuw in de databank onder hetzelfde id.	In de databank zijn nog precies evenveel objecten aanwezig. De straatnaam en de plaatsnaam van een bestaand object zijn gewijzigd.	√

### 4.2.3. Cache

Caching is getest met verschillende browsers. Mozilla versie 1.6, Internet Explorer versie 6.0, Konqueror en Opera.

De aanroepen en de antwoorden zijn gecontroleerd met een proxysniffer.

<i>Beschrijving</i>	<i>Instructies</i>	<i>Verwachte uitvoer</i>	<i>Check</i>
Bij eerste aanroep wordt het volledige pagina ingeladen. Document bevat cache header, datumveld, en ETag	Verwijder alle cache. Roep de detailpagina op.	Kijk in de log of de reactie een ETag en een Last-Modified veld bevat.	√
Bij volgende aanroep binnen 30 seconden (van zelfde document), zoekt browser geen contact met server	Herlaad dezelfde pagina.	Controleer in de log of de webserver niet aangesproken wordt.	√
Bij volgende aanroep na 30 seconden (van zelfde document), zoekt browser alleen contact met server om te valideren.	Wacht 30 seconden en herlaad dezelfde pagina.	Controleer in de log of de webserver aangesproken wordt. In de aanroep bevindt zich een ETag. De server antwoordt met een 304 en het antwoord bevat verder geen data.	√
Nadat document gewijzigd is en de browser opnieuw valideert, 'ziet' deze dat de pagina gewijzigd is, en laadt nieuwe versie in	Verander de straatnaam van een object. Herlaad dezelfde pagina in de browser.	Controleer in de log of de webserver aangesproken wordt. In de aanroep bevindt zich een ETag. De server antwoordt met een status 200 en het antwoord bevat verder de nieuwe ETag en een nieuwe datum voor het veld Last-Modified.	√

---

## 5. CONCLUSIE

### 5.1. STRUTS, MAVERICK OF WEBWORK

In het gebruik zijn de drie frameworken die we onderzocht hebben nauwelijks verschillend. Het grootste verschil zit in de uitwerking van de bibliotheken en de naamgeving van de classes. Eén groot technisch verschil tussen Struts en de andere twee is het gebruik van een Singleton controller. De code in een controller moet daarom altijd synchronized zijn.

Voor het gevoel werken Maverick en Webwork het meest prettig. Vaak is de naamgeving in Struts niet logisch. Dit lijkt te komen doordat struts “backwards compatible” wil zijn met z'n oudere versies. Veel methoden in de bibliotheek zijn daardoor ook deprecated.

De definitieve keuze is gevallen op WebWork, omdat WebWork een hele handige methode aanbiedt om voordat de code van een actie uitgevoerd wordt “interceptors” uit te voeren. Dit is handig om op een eenvoudige manier caching functionaliteit te toe te voegen zonder dat de code van het framework aangepast hoeft te worden.

### 5.2. PERFORMANCE

In de praktijk blijkt er geen performance verschil te zitten tussen de verschillende frameworken. Ook de gebruikte view heeft geen invloed op de performance, een bepaalde view is niet sneller dan een andere view. Om dit te kunnen vaststellen is er een view geschreven in Velocity en een view in JSP met gebruik making van de standaard taglib van Java.

### 5.3. PERSISTENTIE

De abstractie met de databank is gerealiseerd door middel van SQL Maps. SQL Maps is een DAO implementatie van iBatis. SQL Maps werkte zoals verwacht.

Een extra functionaliteit van SQL Maps, lazy loading, kan niet gebruikt worden, omdat dit de methode beïnvloed die hashcode bepaalt. De hashcode wordt gebruikt om te kunnen vergelijken of objecten verschillend zijn.

### 5.4. CACHING

Caching zorgt er voor dat een eerder aangeroepen pagina die in de tussentijd niet gewijzigd is, aanmerkelijker sneller weergegeven wordt. In de praktijk duurt het zo'n 5mS om te testen of een pagina gewijzigd is, terwijl het opnieuw genereren en versturen van een pagina al snel zo'n 30mS in beslag neemt.

Tijdens het testen is gebleken dat het cachen goed werkt in verschillende moderne browsers, waaronder Internet Explorer, Mozilla, Opera en Konqueror.

## 6. REFERENTIES

- [1] SUN MICROSYSTEMS, *JSP Access Models*, maart 2004,  
<http://java.sun.com/developer/onlineTraining/JSPIntro/contents.html>
- [2] ROD JOHNSON, *Expert One-on-One J2EE Design and Development*,  
ISBN 0764543857 Wrox, Oktober 2002.
- [3] CLINTON BEGIN, *iBatis; DAO framework en SQL Maps*, april 2004,  
<http://www.ibatis.com>
- [4] APACHE STRUTS, *Struts user and developer guides*, april 2004,  
<http://jakarta.apache.org/struts>
- [5] MARK NOTTINGHAM, *Caching tutorial for Web Authors and Webmasters*,  
april 2004, [http://www.mnot.net/cache\\_docs/](http://www.mnot.net/cache_docs/)
- [6] O'REILLY JAVA AUTHORS, *Servlet Best Practices*, april 2004,  
[http://www.onjava.com/pub/a/onjava/excerpt/jebp\\_3/index2.html](http://www.onjava.com/pub/a/onjava/excerpt/jebp_3/index2.html)
- [7] INTERNET ENGINEERING TASK FORCE, *Hypertext Transfer Protocol –  
HTTP/1.1 RFC 2616 (Sectie 14.9 Cache-Control)*, april 2004,  
<http://www.ietf.org/rfc/rfc2616.txt>
- [8] KRIS THOMPSON, *Building With WebWork*, november 2003,  
<http://www.theserverside.com/articles/article.tss?l=WebWork2>
- [9] JEFF SCHNITZER, SCOTT HERNANDEZ, JIM MOORE, *Maverick manual*, april  
2004, <http://mav.sourceforge.net/maverick-manual.html>