



Noël Vaes

Java Trainer & Consultant



JavaServer Faces 2.2

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Vrijwel alle namen van software- en hardwareproducten die in deze cursus worden genoemd, zijn tegelijkertijd ook handelsmerken en dienen dienovereenkomstig te worden behandeld.

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand of openbaar worden gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de auteur. De enige uitzondering die hierop bestaat, is dat eventuele programma's en door de gebruiker te typen voorbeelden mogen worden ingevoerd opgeslagen en uitgevoerd op een computersysteem, zolang deze voor privé-doeleinden worden gebruikt, en niet bestemd zijn voor reproductie of publicatie.

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

Ondanks alle aan de samenstelling van deze tekst bestede zorg, kan de auteur geen aansprakelijkheid aanvaarden voor eventuele schade die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

03/02/2017

Copyright© 2017 Noël Vaes



Inhoudsopgave

Hoofdstuk 1. Inleiding.....	3
1.1. Design patterns.....	3
1.2. Model View Controller.....	3
1.3. Frameworks.....	5
1.4. JavaServer Faces.....	5
Hoofdstuk 2. Installatie & Configuratie.....	7
2.1. Wildfly.....	7
2.1.1. Installatie.....	7
2.1.2. Configuratie.....	7
2.1.3. Integratie met Eclipse.....	10
2.2. Installatie van JavaServer Faces.....	12
2.3. Een JSF-project maken.....	12
Hoofdstuk 3. JSF Overzicht.....	17
3.1. MVC algemeen.....	17
3.2. De JSF-architectuur.....	17
3.2.1. FacesServlet.....	18
3.2.2. Facelets.....	19
3.2.3. Managed Beans.....	20
3.3. Mijn eerste JSF-pagina.....	21
3.4. De JSF-levenscyclus.....	25
Hoofdstuk 4. Managed Beans.....	29
4.1. Inleiding.....	29
4.2. Backing beans.....	29
4.3. Configuratie van een managed bean.....	30
4.4. De scope van een bean.....	31
4.5. Afhankelijkheden tussen managed beans.....	32
Hoofdstuk 5. Expression language.....	37
5.1. Inleiding.....	37
5.2. Letterlijke waarden.....	38
5.3. Operatoren.....	38
5.4. Scope-objecten.....	39
5.5. Voorgedefinieerde objecten.....	39
5.6. Methoden oproepen.....	40
Hoofdstuk 6. Navigatie.....	41
6.1. Inleiding.....	41
6.2. Statische navigatie.....	41
6.3. Dynamische navigatie.....	48
6.4. Koppelingen en View Parameters.....	51
Hoofdstuk 7. Standaard JSF-tags.....	58
7.1. Inleiding.....	58
7.2. Core Tags.....	58
7.3. HTML Tags.....	60
7.3.1. Algemene attributen.....	60
7.3.2. Formulieren.....	62
7.3.3. Tekstvelden en tekstgebieden.....	62
7.3.4. Teksten en afbeeldingen tonen.....	63
7.3.5. Knoppen en koppelingen.....	64
7.3.6. Selecties.....	64
7.3.7. Scripts en stylesheets.....	69



7.3.8.Panels.....	69
7.3.9.Gegevenstabellen.....	74
7.3.10.HTML-tags en HTML-editors.....	81
7.4.JSF en JavaScript.....	82
Hoofdstuk 8.Internationalisation (I18N).....	86
8.1.Inleiding.....	86
8.2.Het Locale object.....	86
8.3.JSF Message Bundles.....	87
Hoofdstuk 9.Conversie en validatie.....	90
9.1.Inleiding.....	90
9.2.Conversie.....	91
9.2.1.Ingebouwde conversie.....	91
9.2.1.1.Conversie van getallen.....	91
9.2.1.2.Conversie van datums en tijden.....	96
9.2.2.Aangepaste conversie.....	96
9.3.Validatie.....	99
9.3.1.Ingebouwde validators.....	99
9.3.1.1.Verplichte velden.....	99
9.3.1.2.De lengte van een veld.....	100
9.3.1.3.Getalbegrenzingsen.....	100
9.3.1.4.Reguliere uitdrukkingen.....	101
9.3.2.Aangepaste validators.....	101
9.3.3.Validatie door een backing bean.....	103
9.3.4.Gecombineerde validatie door een backing bean.....	105
9.3.5.Conversie en validatie overslaan.....	107
9.3.6.Parameters doorgeven aan converters en validators.....	108
9.3.7.Bean validation (JSR-303).....	110
Hoofdstuk 10.Event Handling.....	116
10.1.Inleiding.....	116
10.2.Value Change Events.....	117
10.3.Action Events.....	121
10.4.PropertyActionListener.....	123
10.5.ViewActions.....	128
Hoofdstuk 11.JSF & Ajax.....	132
11.1.Inleiding.....	132
11.2.XmlHttpRequest.....	132
11.3.JSF en Ajax.....	134
Hoofdstuk 12.Templates.....	140
12.1.Inleiding.....	140
12.2.UI-tags voor gebruik in templates.....	141
12.3.Andere UI-tags.....	144



Hoofdstuk 1. Inleiding

1.1. Design patterns

Java Servlets en *Java Server Pages* (JSP) hebben het mogelijk gemaakt dynamische websites te ontwikkelen op basis van de Java-technologie. Dergelijke webapplicaties beantwoorden aan het WORA-principe: *Write Once Run Anywhere*.

Servlets bieden de mogelijkheid om programmatisch pagina's *realtime* te genereren. *Servlets* hebben echter het nadeel dat de volledige HTML-code programmatisch gegenereerd moet worden. HTML-pagina's bestaan echter voor het grootste gedeelte uit statische gegevens. Het gebruik van *servlets* is daarom vrij omslachtig. Een wijziging van de website vergt een wijziging van de Java-code. Bovendien moet de webdesigner de Java-programmeertaal machtig zijn. Hij moet zowel een programmeur als een grafisch ontwerper zijn.

Sommige van deze problemen werden opgelost met *Java Server Pages* (JSP). Dit was een makkelijke manier om een *servlet* te genereren. De rollen werden hierbij omgekeerd. JSP-pagina's bestaan gewoon uit HTML-syntax waardoor het makkelijk is de statische inhoud van de pagina te genereren en te onderhouden. De dynamische gegevens worden hieraan toegevoegd door middel van *scriptlets* en *custom tags*. De webcontainer zorgt voor de omzetting van de JSP-pagina naar een *servlet*. Een JSP-pagina is daarom ook niet meer dan een makkelijke manier om een *servlet* te genereren.

Met de introductie van JSP werden niet alle problemen opgelost. De JSP pagina's verworden al snel tot een onleesbare en moeilijk onderhoudbare mengeling van statische HTML-code en *scriptlets*. Als er bovendien ook nog (*clientside*-)JavaScript-code aan wordt toegevoegd, wordt de verwarring nog groter. Ook hier moet de webontwikkelaar zowel een ontwerper als een programmeur zijn.

Dit soort problemen heeft geleid tot het ontstaan van een aantal *design patterns* voor de ontwikkeling van webapplicaties. Een *design pattern* is een beproefde manier of patroon om een bepaald soort problemen adequaat op te lossen. *Design patterns* ontstaan in de gemeenschap van (web)ontwikkelaars doordat ze allen dezelfde problemen tegenkomen en hiervoor oplossingen zoeken. Deze oplossingen worden uitgewisseld, getest en bijgestuurd tot ze een zekere maturiteit bekomen.

Voor het ontwikkelen van grafische applicaties is het *design pattern Model View Controller* zeer in trek. We kunnen dit toepassen bij dynamische webapplicaties en het zal ons helpen een goede scheiding te maken tussen de presentatielogica en applicatielogica.

1.2. Model View Controller

Het *design pattern Model View Controller* (MVC) vindt zijn oorsprong in de programmeertaal *SmallTalk*.

Bij het ontwikkelen maakt men een onderscheid tussen drie verschillende componenten:

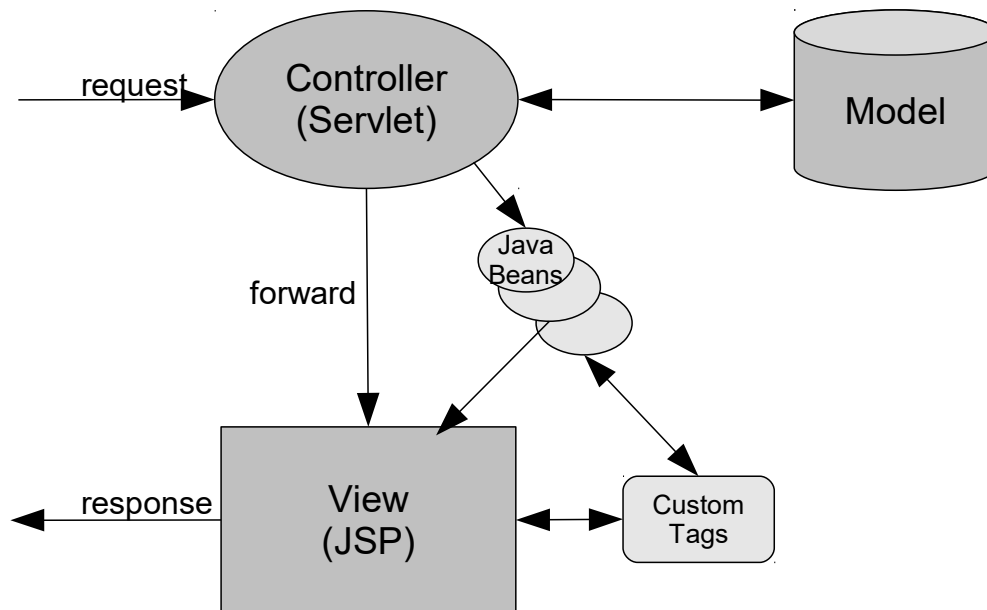
1. **Model:** Dit is doorgaans het datamodel van de applicatie. Bij eenvoudige applicaties is dit vaak een databank waarin de gegevens bewaard worden. In meer geavanceerde applicaties omvat dit ook de componenten die de *business*-logica omvatten (*multitier*-applicaties).
2. **View:** Dit is het zicht dat de eindgebruiker heeft op het *model*. Zodra het *model* om een of andere reden verandert, wordt de *view* hiervan op de hoogte gebracht.
3. **Controller:** De gebruiker kan via de *view* ook ingrijpen op de applicatie. De acties die



de gebruiker onderneemt worden afgehandeld door de controller. Deze zal de nodige wijzigingen aanbrengen in het *model* en er tevens voor zorgen dat de *view* wordt aangepast.

Door een dergelijke opsplitsing van verantwoordelijkheden, kan men op een gestructureerde wijze software ontwikkelen.

Dit *design pattern* kan concreet toegepast worden voor webapplicaties. Het plaatje ziet er dan als volgt uit:



Hier is een woordje uitleg op zijn plaats.

De pagina's die de gebruiker te zien krijgt, worden gegenereerd door middel van JSP. Dit is eerder het domein van de webdesigner die zorg moet dragen voor een mooie en consequente lay-out. De gebruiker kan echter acties ondernemen door te klikken op koppelingen of door formulieren te versturen. Al deze acties worden gestuurd naar een *servlet*. Dit is de controller die het verloop van de applicatie controleert. Op basis van de acties die de gebruiker doet, zal de controller aanpassingen doen aan het model. In eenvoudige applicaties is dit doorgaans het aanpassen van gegevens in een databank. Als het model wijzigt, moet ook de view aangepast worden. De nieuwe gegevens worden daarom door de controller bij het model opgevraagd en ondergebracht in *JavaBeans*. Deze zijn gewone Java-objecten voorzien van de nodige *getters* en *setters*. Deze *JavaBeans* worden nadien beschikbaar gesteld in een of andere *scope*, zodat ze nadien voor de JSP-pagina toegankelijk zijn.

In een volgende stap stuurt de controller het verzoek door naar een JSP-pagina. Deze is namelijk verantwoordelijk voor het genereren van de nieuwe *view* voor de gebruiker. In de JSP-pagina worden de gegevens uit de *JavaBeans* getoond. Om deze gegevens makkelijk te integreren, maakt men gebruik van *Expression Language* en *Custom Tags*.



De MVC-architectuur maakt het beheer en ontwikkelen van complexe applicaties makkelijker. Zo kan de beveiliging, het bijhouden van de applicatiestatus en de *flow* van de applicatie centraal beheerd worden.

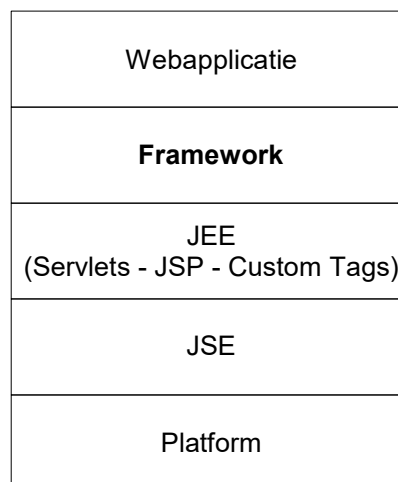
Strikt genomen is het in de MVC-architectuur mogelijk meerdere controllers te hebben die elk verantwoordelijk zijn voor een deel van de applicatie. Het *design pattern Front Controller* plaatst echter een centrale *servlet* aan de ingang van de webapplicatie. Van daaruit worden alle verzoeken doorgestuurd naar de verschillende componenten. De *Model View Controller* en *Front Controller* zijn twee *design patterns* die makkelijk samen gebruikt en geïntegreerd kunnen worden. Dit is onder andere het geval bij *JavaServer Faces*.

1.3. Frameworks

De MVC-architectuur is in de loop der jaren een veel toegepast *design pattern* geworden. Als men gebruik wil maken van dit *pattern* heeft men wel een goede manier om een webapplicatie op te bouwen, maar men is nog steeds verplicht de nodige componenten volgens dit *pattern* te ontwikkelen.

Gelukkig zijn er intussen allerlei *frameworks* gemaakt die ons helpen een webapplicatie te ontwikkelen. Een *framework* is niet meer dan een reeks klassen en interfaces die op een abstracte en generieke wijze bepaalde functionaliteit aanbieden waarmee we onze eigen (web-)applicatie gestalte kunnen geven. Applicaties worden als het ware boven een *framework* gebouwd waarbij het *framework* zorgt voor de onderliggende functionaliteit die gemeenschappelijk is voor alle applicaties.

Zo zijn er inmiddels ook *frameworks* die het MVC-pattern voor webapplicaties implementeren en waarboven een concrete webapplicatie gebouwd kan worden. De fundamentele code die zorgt voor het MVC-mechanisme wordt daarbij door het *framework* geleverd zodat de programmeur/webdesigner zich kan toeleunen op datgene wat specifiek is voor zijn applicatie.



Populaire *frameworks* zijn *Struts*, *Cocoon*, *Tapestry*, *Tiles*, *Spring MVC* enzovoort. Sommige zijn *open source*, andere daarentegen *closed source*.

1.4. JavaServer Faces

Al deze *frameworks* hebben hun eigen manier van werken en zijn niet compatibel. De keuze voor zo'n *framework* impliceert doorgaans ook dat men helemaal gebonden is aan dit



product. In het verleden zijn al meermaals allerlei *frameworks* ontstaan die na een tijdje een stille dood stierven en niet verder ontwikkeld werden. De keuze voor dergelijke *frameworks* houdt daarom ook altijd een risico in.

Vanuit de noodzaak aan een meer gestandaardiseerd *framework* is *JavaServer Faces* (JSF) ontstaan. JSF is evenwel geen concrete implementatie van een *framework* maar een specificatie die deel uitmaakt van de omvattende JEE-specificatie. Voor de concrete implementatie van deze specificatie moeten we beroep doen op de producten die door leveranciers aangeleverd worden. Producenten van JEE-webservers of applicatieservers voorzien deze concrete implementatie in hun product. Door het feit dat ze allemaal aan dezelfde specificatie moeten voldoen, bestaat de garantie dat de toepassingen die ontwikkeld worden met JSF, tevens compatibel zijn met verschillende concrete applicatieservers. Hierdoor is men niet gebonden aan een concrete leverancier, hetgeen het risico op lange termijn vermindert.

Van JSF zijn er intussen verschillende versies die gebruikt worden in de JEE-specificatie:

JEE 5: JSF 1.2

JEE 6: JSF 2.0

JEE 7: JSF 2.2

In deze cursus behandelen we JSF 2.2 en maken we gebruik van een JEE-applicatieserver (*WildFly 10*) die de JEE7-specificaties ondersteunt en in de nodige implementaties voorziet.



Hoofdstuk 2. Installatie & Configuratie

2.1. Wildfly

WildFly is beschikbaar op volgende website : <http://www.wildfly.org>.

Voor de installatie van *WildFly* moet de JDK geïnstalleerd zijn op het systeem. In deze cursus maken we gebruik van **JDK 1.8** in combinatie met **WildFly 10.1.0.Final**. Andere versies van *WildFly* kunnen een andere configuratie hebben.

2.1.1. Installatie

WildFly kan van de site afgehaald worden in de vorm van een ZIP-bestand dat we gewoon kunnen uitpakken in een of andere lokale map. Deze map noemen we de **WildFly home directory**. Deze *home directory* kan via een omgevingsvariabele **JBOSS_HOME** ingesteld worden zodat hij gebruikt kan worden in andere applicaties (bijvoorbeeld *Maven*). Aangezien *WildFly* vroeger *JBoss* heette, wordt doorgaans **JBOSS_HOME** gebruikt.

Opdracht 1: WildFly installeren

In deze opdracht gaan we de *WildFly*-server installeren.

- Haal de laatste versie van **WildFly** af van de website <http://wildfly.org/downloads/> (**WildFly-10.1.0-Final.zip**).
- Pak het bestand uit in een lokale map, bijvoorbeeld **C:\WildFly**. Het is aangewezen geen map te gebruiken waarvan de naam een spatie bevat, zoals **C:\Program Files**
- Stel de omgevingsvariabele **JBOSS_HOME** in met deze folder:

```
JBOSS_HOME=C:\WildFly\WildFly-10.1.0.Final
```

2.1.2. Configuratie

Na de installatie vinden we de volgende mappenstructuur:

```
appclient
bin
docs
domain
modules
standalone
  +-- configuration
  +-- data
  +-- deployments
  +-- lib
```



Folder	Omschrijving
bin	Hierin bevinden zich de <i>batch</i> - en <i>shell</i> -bestanden voor het opstarten en afsluiten van <i>WildFly</i> .
docs	Hierin bevindt zich documentatie, waaronder de DTD's en schema's van bepaalde XML-bestanden.
domain	Bevat de configuratie voor <i>WildFly</i> in <i>domain</i> -mode.
modules	Bevat extra JAR-bestanden voor modules.
standalone	Bevat de configuratie voor <i>WildFly</i> in <i>standalone</i> -mode.
configuration	Hierin bevinden zich de configuratiebestanden van deze <i>WildFly</i> -configuratie. Dit zijn hoofdzakelijk XML-bestanden.
data	Bevat gegevensbestanden.
deployments	In deze folder kunnen de webapplicaties, EJB's en volledige <i>enterprise</i> -applicaties geplaatst worden. Deze worden door de server opgepikt en in werking gesteld.
lib	Hierin bevinden zich de extra JAR-bestanden die nodig zijn voor deze configuratie.

De server kan opgestart worden met het *batch*-bestand ***standalone.bat*** in de folder ***bin***.

Standaard is *WildFly* om veiligheidsredenen enkel toegankelijk via de *localhost* (127.0.0.1) netwerkinterface. Andere IP-adressen kunnen opgegeven worden via de optie `-b w.x.y.z`. Bij 0.0.0.0 worden alle IP-adressen gebruikt.

```
standalone.bat -b 0.0.0.0
```

Om *WildFly* terug af te sluiten, drukken we CTRL-C.

De openingspagina kan bereikt worden via deze URL: ***http://localhost:8080***



Voor het beheer van **WildFly** kan men gebruikmaken van volgende URL:
http://localhost:9990

Om toegang te krijgen, moet er eerst een gebruiker toegevoegd worden. Dit wordt gedaan met het programma ***add-user.bat***

De *batch*-bestanden zijn voor Windows-systemen. Voor UNIX/LINUX bevinden zich in dezelfde folder de gelijknamige *shell scripts*.

Opdracht 2: WildFly opstarten

In deze opdracht gaan we eerst enkele gebruikers met hun wachtwoord aanmaken. Daarna zullen we *WildFly* opstarten en terug afsluiten.

- Open een opdrachtvenster en navigeer naar de map ***bin***.
- Voeg een nieuwe gebruiker toe met het volgende commando:

add-user.bat

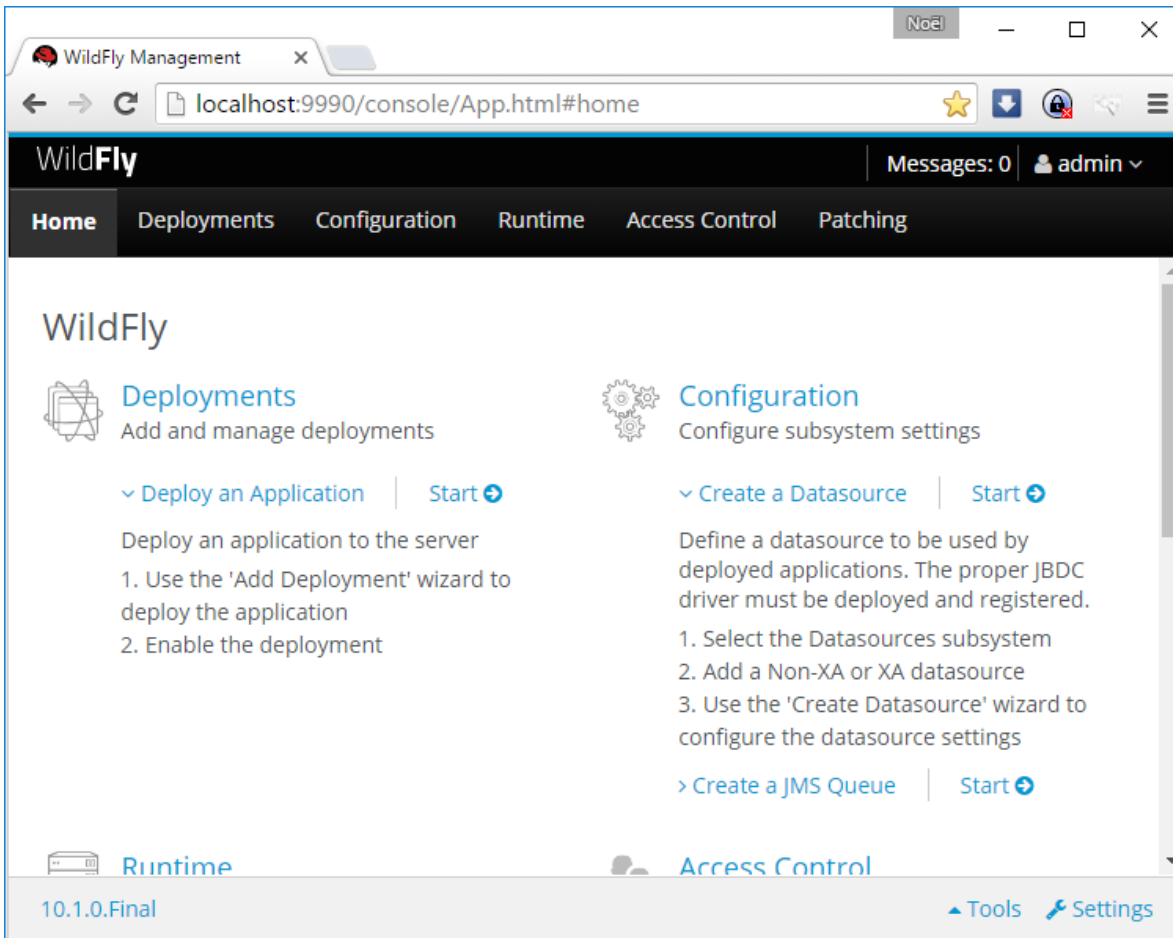
Selecteer *Management User* en gebruik als naam ***admin*** met wachtwoord ***jboss***

- Start de *WildFly*-server op met het commando:



standalone.bat

- Open de hoofdpagina: **http://localhost:8080**
- Open de pagina voor de administratie: **http://localhost:9990**



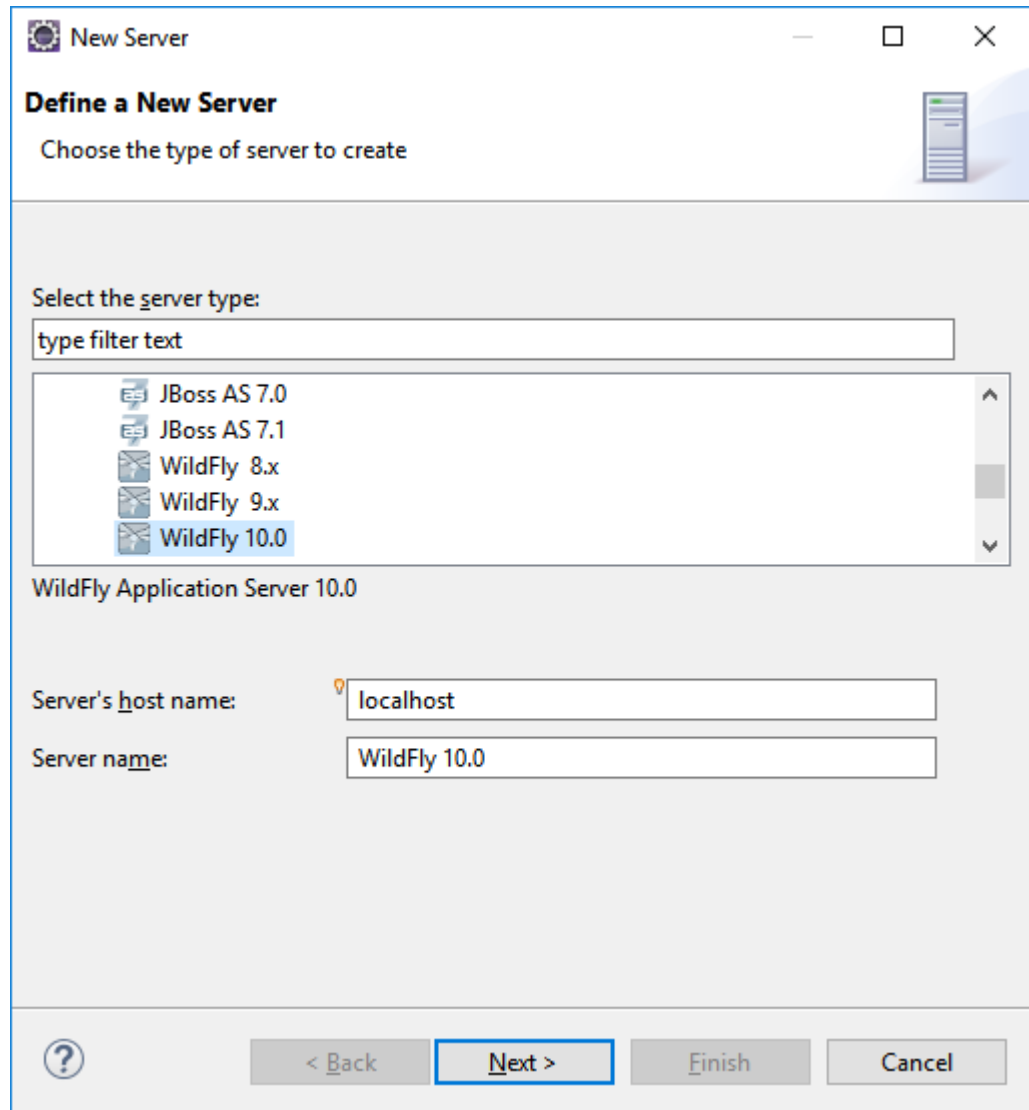
- Stop de *WildFly*-server door CTRL-C te drukken

2.1.3. Integratie met Eclipse

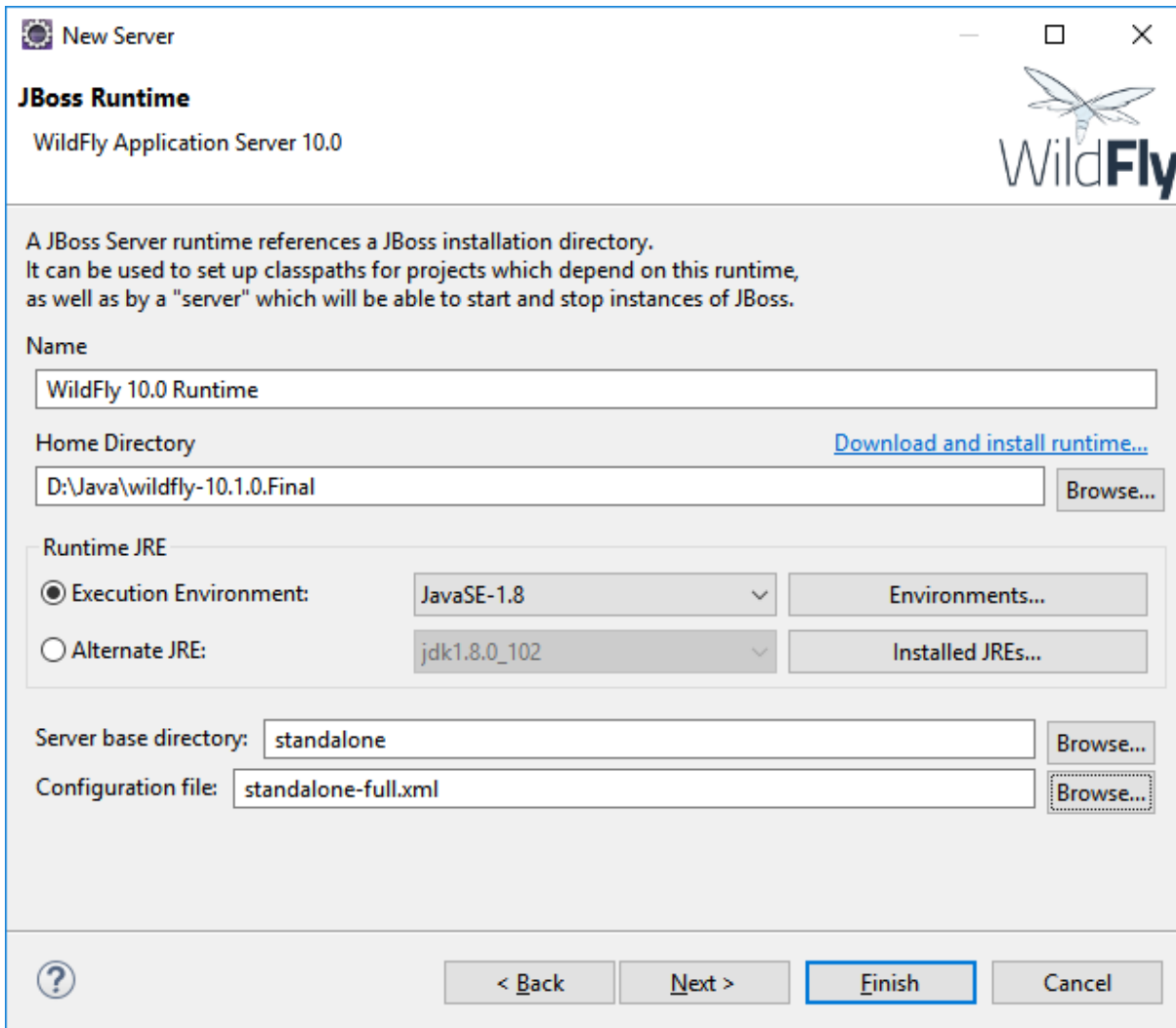
Indien men gebruikmaakt van *Eclipse* is het mogelijk *WildFly* vanuit deze omgeving op te starten.

Opdracht 3: Integratie met Eclipse

- Voeg de *view Servers* toe aan de werkomgeving.
- Selecteer **New->Server** (via rechtermuisklik in het venster).



- Selecteer **WildFly 10.0**. en klik vervolgens tweemaal op **Next**.



- Selecteer de juiste *Home Directory*.
- Klik op **Finish**
- Selecteer de server in de *Servers view* en start hem.

2.2. Installatie van *JavaServer Faces*

JSF maakt deel uit van de JEE-specificatie. Producenten van JEE-gecertificeerde applicatieservers dienen een concrete implementatie van JSF in hun product te voorzien. Dit is bijvoorbeeld het geval bij *Glassfish*, *JBoss/WildFly*, *Websphere*, *Weblogic e.a.* In dit geval is een extra installatie van een JSF-implementatie niet nodig.

Indien men gebruik wil maken van een server die niet de volledige JEE-specificatie implementeert (zoals bijvoorbeeld *Tomcat*), dient men wel de nodige JSF-bibliotheken toe te voegen. Deze procedure is soms omslachtig en valt buiten het bestek van deze cursus.

2.3. Een JSF-project maken

JSF is gewoon gebouwd op de standaardtechnieken van een Java-webapplicatie. Een JSF-project is daarom hetzelfde als elk ander webproject. Een gewone Java-webapplicatie dient volgende structuur te hebben:

```
.\WEB-INF
  +-- classes
```



```
+-- lib
+-- web.xml
+-- faces-config.xml
```

Het bestand **web.xml** is de *deployment descriptor* waarin de webapplicatie geconfigureerd wordt. Dit bestand is niet altijd nodig omdat bepaalde configuraties automatisch gebeuren.

Het bestand **faces-config.xml** is het specifieke configuratiebestand voor JSF. Dit bestand moet wel aanwezig zijn om de automatische configuratie van JSP te activeren.

Dit bestand ziet er als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.2"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
</faces-config>
```

De map **classes** bevat gecompileerde klassen die opgenomen moeten worden in het *classpath* van de webapplicatie.

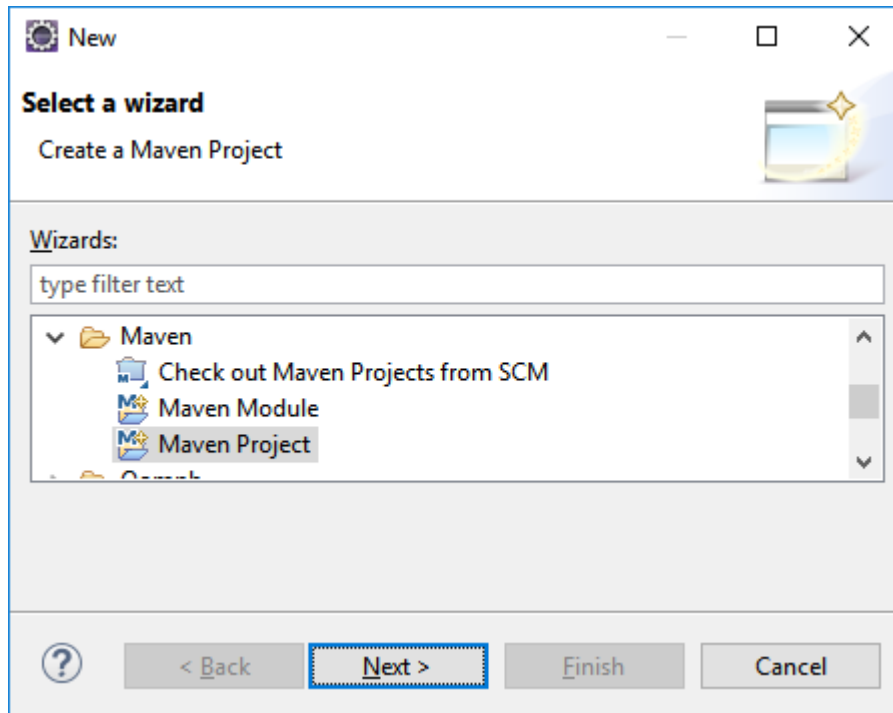
De map **lib** bevat JAR-bestanden met gecompileerde klassen die ook toegevoegd worden aan het *classpath*.

We kunnen een dergelijk webproject manueel opzetten maar de meeste IDE's hebben hiervoor de nodige *wizards* aan boord. Het gebruik van *Maven* is tevens aangewezen.

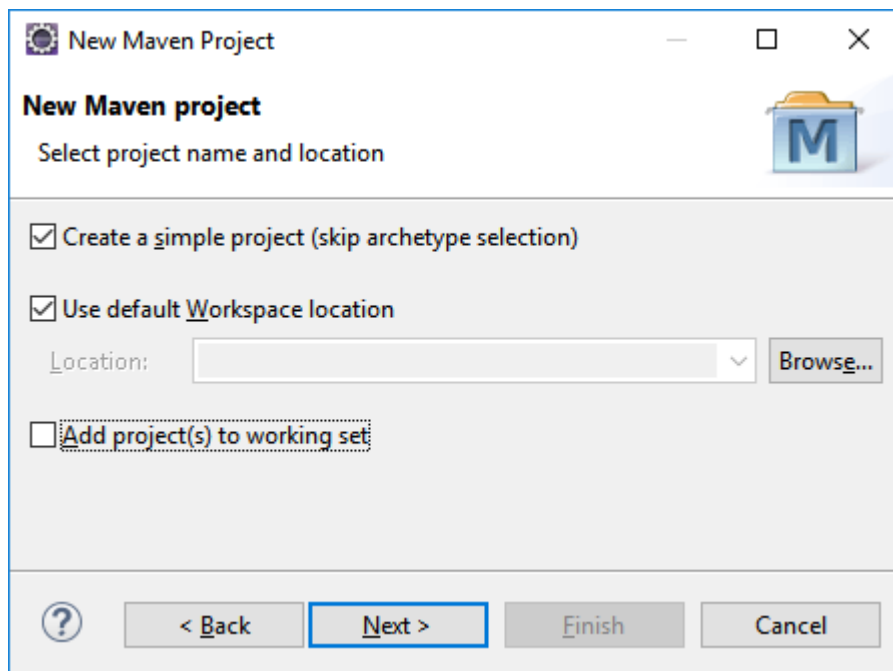
Opdracht 4: Een JSF-webproject opzetten met Maven in Eclipse

In deze opdracht gaan we een *Maven*-JSF-webproject maken in *Eclipse*.

- Selecteer **File->New->Other**



- Kies **Maven Project** en klik op **Next**



- Selecteer **Create a simple project (skip archetype selection)** en klik op **Next**.



- Vul *Group id* en *Artifact Id* van dit project (jsf) in en selecteer **war** als *Packaging*. Klik op **Finish**.
- Voeg het bestand **faces-config.xml** toe in de map **src/main/webapp/WEB-INF**:

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.2"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
</faces-config>
```

- Open de POM en voeg de volgende configuratie toe:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```



```

http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>eu.noelvaes</groupId>
  <artifactId>jsf</artifactId>
  <packaging>war</packaging>
  <version>2.2</version>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>
      UTF-8
    </project.build.sourceEncoding>
  </properties>

  <build>
    <finalName>jsf</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.0.0</version>
      </plugin>
      <plugin>
        <groupId>org.wildfly.plugins</groupId>
        <artifactId>wildfly-maven-plugin</artifactId>
        <version>1.1.0.Final</version>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <!-- Only to help Eclipse solve JSF-namespaces -->
      <groupId>org.wildfly</groupId>
      <artifactId>wildfly-jsf</artifactId>
      <version>10.1.0.Final</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>

```

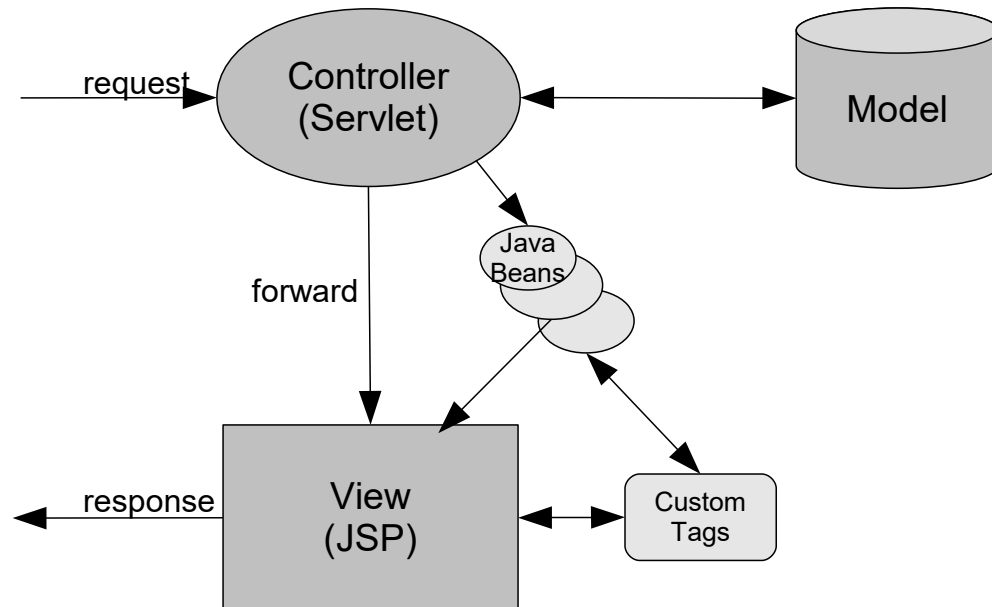
- Start *WildFly* op.
- Maak een nieuwe pagina *index.html* in de map *webapp* en voeg hier naar eigen keuze wat inhoud aan toe.
- Voer het volgende *Maven*-commando uit:
`mvn clean install wildfly:deploy`
- Open de volgende URL: **<http://localhost:8080/jsf>**.



Hoofdstuk 3. JSF Overzicht

3.1. MVC algemeen

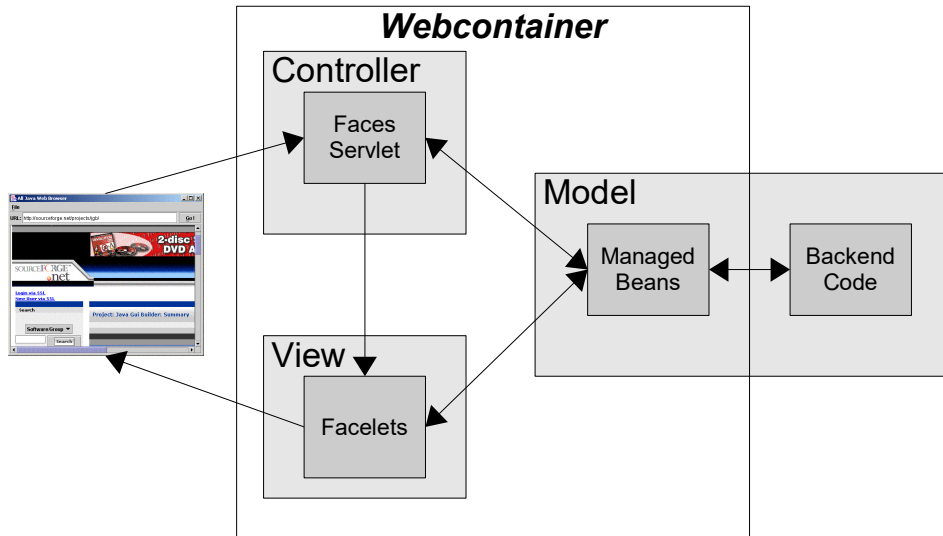
JSF is een specificatie die de *Model-View-Controller*-architectuur gebruikt (MVC). We hernemen nog even de schematische voorstelling hiervan:



1. **Controller:** Deze component is de centrale plaats waar alle verzoeken van de *client* (browser) binnenkomen. De controller onderneemt de nodige acties naar het *model* toe en zendt vervolgens het verzoek door naar de *View* (JSP-pagina).
2. **Model:** Deze component bevat de *business logic* van de applicatie. Vaak heeft deze component de vorm van *JavaBeans*, *Enterprise JavaBeans* of *CDI-beans*.
3. **View:** Deze component genereert de lay-out die de gebruiker te zien krijgt. De nodige informatie verkrijgt hij via onder andere *JavaBeans* die door het *model* gegenereerd zijn en hem door de *controller* zijn doorgespeeld.

3.2. De JSF-architectuur

Bij JSF ziet de concrete MVC-architectuur er als volgt uit:



De controller wordt geïmplementeerd door de *FacesServlet*. De view wordt hier bepaald door *Facelets* en het model bestaat uit *managed beans* met eventueel de *backend code* die erachter ligt. We zullen nu de drie componenten meer in detail bespreken.

3.2.1. FacesServlet

De controller is een *servlet* met de naam ***FacesServlet***. Alle verzoeken komen bij deze *servlet* binnen en van daaruit gebeuren de nodige acties. Men noemt dit ook de *front controller*. Deze *servlet* zouden we op de klassieke wijze moeten configureren in ***web.xml***.

Sinds JEE6 bestaat er evenwel de mogelijkheid tot het gebruik van *web fragments* en *WebServletContextListeners*. Deze mechanismen maken het mogelijk dat *frameworks* zichzelf configureren. De JAR-bestanden van het *framework* dienen hiertoe de nodige XML-bestanden (***web-fragment.xml***) of klassen met de annotatie `@WebServletContextListener` te bevatten. De configuratie wordt bijgevoegd automatisch toegevoegd aan de configuratie van de webapplicatie.

JSF 2.2 ondersteunt dit mechanisme en daarom is het niet meer nodig een expliciete configuratie toe te voegen in ***web.xml***. De *FacesServlet* wordt automatisch geconfigureerd zodra het bestand ***faces-config.xml*** in ***WEB-INF*** aanwezig is. Er wordt dan een mapping met de volgende URL-patronen toegevoegd:

```

*.faces
*.jsf
/faces/*
  
```

Dat wil dus zeggen dat alle verzoeken met dit patroon steeds bij de *FacesServlet* terechtkomen.

Bij de opzet van het *Maven*-project hebben we de configuratie van ***web.xml*** achterwege gelaten omdat we dit bestand voorlopig niet nodig hebben. De afwezigheid van ***web.xml*** kan bij oudere versies van de *Maven-WAR-plugin* tot fouten leiden. Om die reden maken we gebruik van een recente versie van de *plugin* die deze aanwezigheid niet meer vereist: