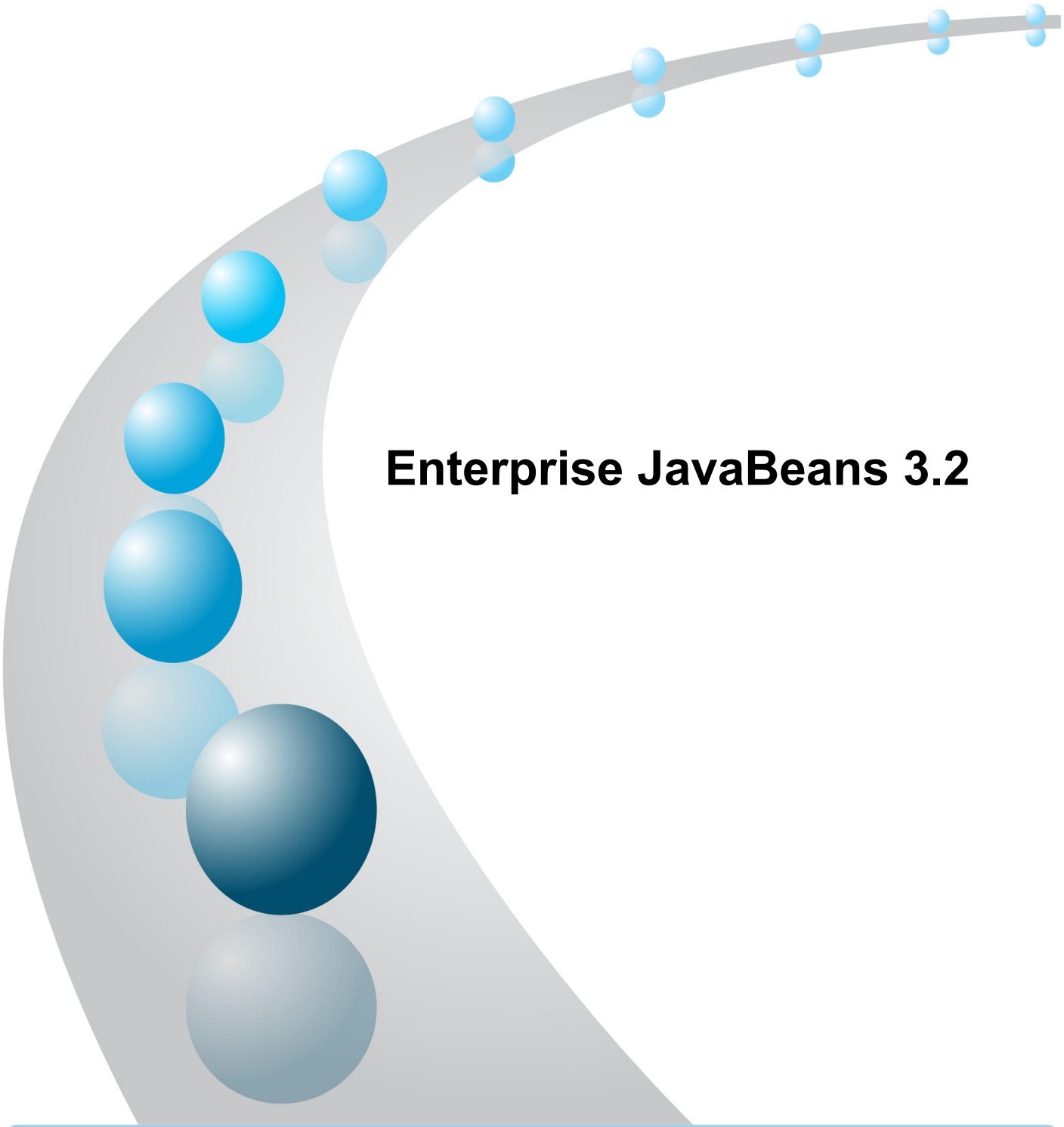
Noël Vaes

Java Trainer & Consultant



Enterprise JavaBeans 3.2

Roode Roosstraat 5
3500 Hasselt
België

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

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

21/08/2019

Inhoudsopgave

Hoofdstuk 1: Inleiding tot JEE

1.1 Multitier gedistribueerde applicaties

Java werd aanvankelijk vooral gebruikt voor het schrijven van platformonafhankelijke applicaties die makkelijk via het internet verspreid konden worden over heterogene systemen. Dit alles onder het motto: *Write Once Run Anywhere,* beter gekend als het *WORA*-principe. Het voorbeeld hiervan is de *applet* die makkelijk in webpagina's geïntegreerd kon worden en in nagenoeg alle browsers uitgevoerd kan worden, los van het onderliggende besturingssysteem.

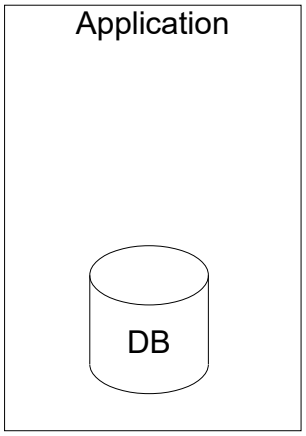
De grote voordelen van Java zorgden er voor dat deze taal ook meer en meer gebruikt ging worden voor *standalone*-toepassingen. Stilaan vond Java ook zijn weg naar de grotere *multitier* gedistribueerde applicaties. Om tegemoet te komen aan de vereisten voor dit soort applicaties werd het Java-platform uitgebreid met de *Java Enterprise Edition (JEE)*. Deze bestaat uit allerlei uitbreidingen en aanvullende technologieën voor het ontwikkelen van *enterprise*-applicaties op basis van Java.

Multitier gedistribueerde applicaties zijn toepassingen waarbij de functionaliteit verspreid ligt over meerdere systemen die door middel van een netwerk met elkaar verbonden zijn. De software wordt onderverdeeld in verschillende lagen en zuilen (*tiers*[1]) met elk hun eigen verantwoordelijkheid.

Om de noodzaak van dat soort applicaties aan te tonen, geven we even een overzicht van de verschillende soorten applicaties.

1.1.1 *One-tier*-applicaties

De meest eenvoudige applicaties zijn de *one-tier*-applicaties. Heel de functionaliteit is vervat in de applicatie en deze kan bijgevolg volledig zelfstandig uitgevoerd worden.

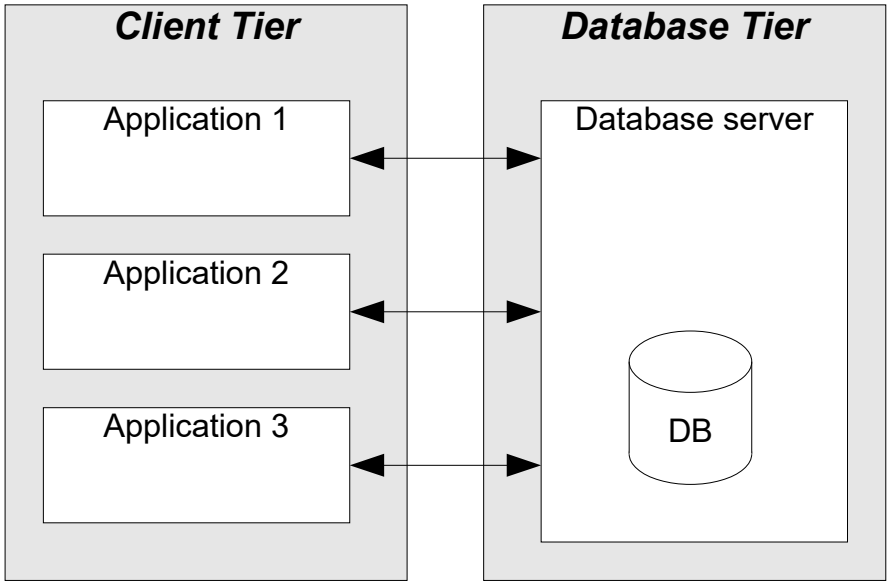


Vaak wordt er in dat soort applicaties gebruikgemaakt van een database. Deze is dan vervat in de applicatie zelf. Men spreekt dan van een *embedded database*. Dit soort applicaties is goed indien er geen informatie gedeeld moet worden met andere applicaties, eventueel andere instanties van dezelfde applicatie. Iedere applicatie staat volledig op zichzelf en is niet verbonden met andere applicaties. Alle functionaliteit wordt uitgevoerd op het lokale systeem.

1 Er is een onderscheid tussen *layers* en *tiers*. Bij *layers* gaat het om de logische lagen van de applicatie terwijl *tiers* eerder duiden op de fysische lagen. Vaak zijn de afzonderlijke lagen aanwezig afzonderlijke *tiers* maar dat hoeft niet altijd zo te zijn.

1.1.2 *Two-tier*-applicaties

Meestal is het echter nodig dat een applicatie informatie deelt met andere applicaties, eventueel met andere instanties van dezelfde applicatie. Hierbij wordt het bewaren van de gegevens afgezonderd uit de applicatie en toevertrouwd aan een tweede applicatie die voor meerdere toepassingen toegankelijk is. Indien de data bewaard wordt in een database, wordt gebruikgemaakt van een databaseserver.



De software wordt daarbij verspreid over twee *tiers*. Vooreerst is er de *client tier* die de applicatie bevat waarmee de gebruiker werkt. Voorts is er de *database tier* die de databaseserver bevat. Deze is doorgaans geplaatst op een andere machine in het netwerk. De communicatie tussen de applicatie en de databaseserver verloopt dan via het netwerk (meestal op basis van het TCP/IP protocol).
De gegevens die bewaard worden door de databaseserver zijn toegankelijk vanuit verschillende applicaties. Dit maakt het mogelijk dat deze applicaties hun gegevens delen.

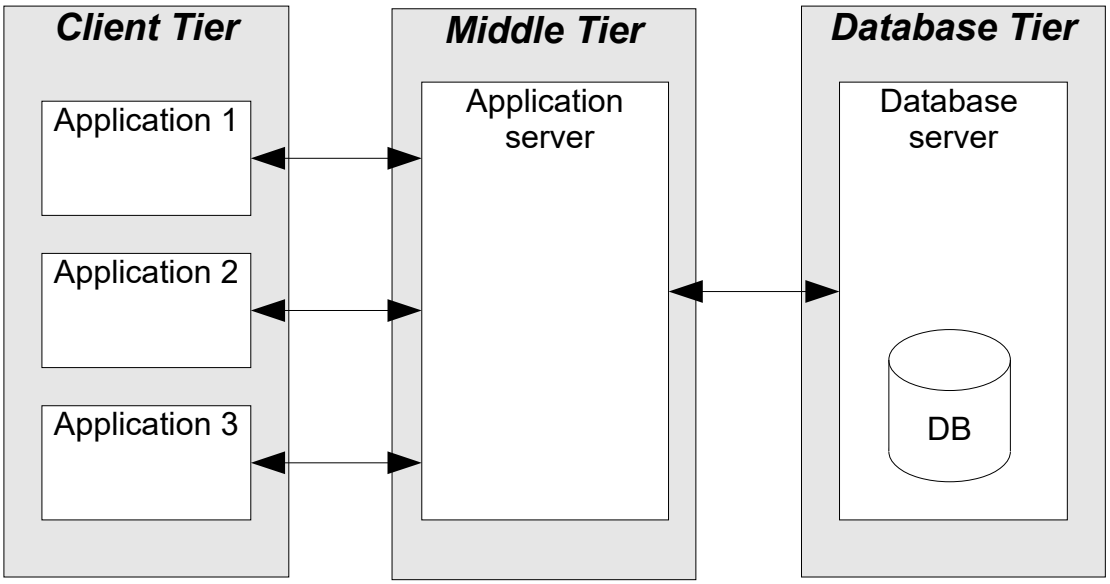
1.1.3 *Three-tier*-applicaties

Bij *two-tier*-applicaties bevindt zich heel de applicatielogica in de *client tier*. Tevens bevat deze *tier* ook alles om de gebruikersinterface te presenteren. We noemen dit ook wel de presentatielogica.

Het is echter mogelijk dat dezelfde applicatie verschillende soorten gebruikersinterfaces heeft. Denk maar aan een applicatie met zowel een WEB-interface als een SWING-interface.

In het *two-tier*-model moeten we voor iedere gebruikersinterface een afzonderlijke applicatie maken met de eigen presentatielogica. Vermits ook de applicatie-logica vervat is in de applicatie, moeten we deze daarin telkens opnieuw voorzien.

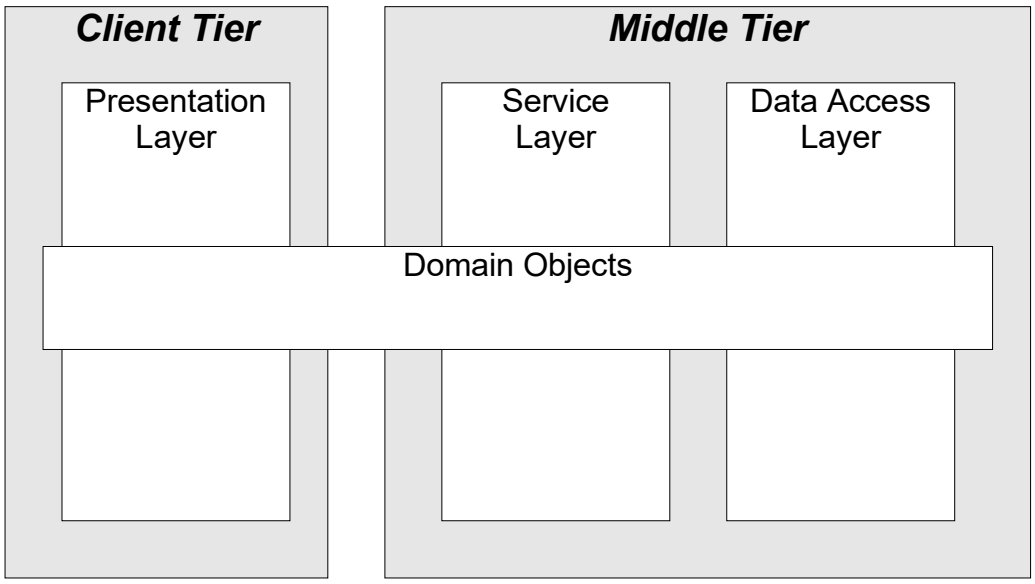
Het zou echter beter zijn de applicatielogica af te zonderen van de presentatielogica. Dit komt de herbruikbaarheid van de softwarecomponenten ten goede. Dit resulteert in een *three-tier*-applicatie.



De middelste *tier* bevat de *application server* die de applicatielogica bevat. Hiermee bedoelen we uiteindelijk alle functionaliteit die niet onmiddellijk gerelateerd is aan de presentatie van de applicatie aan de gebruiker. We spreken in het algemeen van *middleware*; dit is software die zich in het midden bevindt.

Deze *application server* bevindt zich doorgaans ook op een afzonderlijke machine in het netwerk. Verschillende applicaties kunnen simultaan gebruikmaken van de *application server.* De eindapplicaties moeten nu enkel nog zorgen voor de aangepaste presentatie van de applicatie naar de gebruiker toe. Zo kunnen verschillende applicaties met totaal verschillende gebruikersinterfaces toch samen gebruikmaken van dezelfde applicatie-logica. Vermits de applicaties enkel nog de presentatielogica bevatten, zijn ze dus vaak erg afgeslankt. Men spreekt in dat geval ook wel van *thin clients*.

We kunnen nog wat verder inzoomen op de *Client Tier* en *Middle Tier*. In de onderstaande afbeelding worden de verschillende lagen meer in detail afgebeeld:



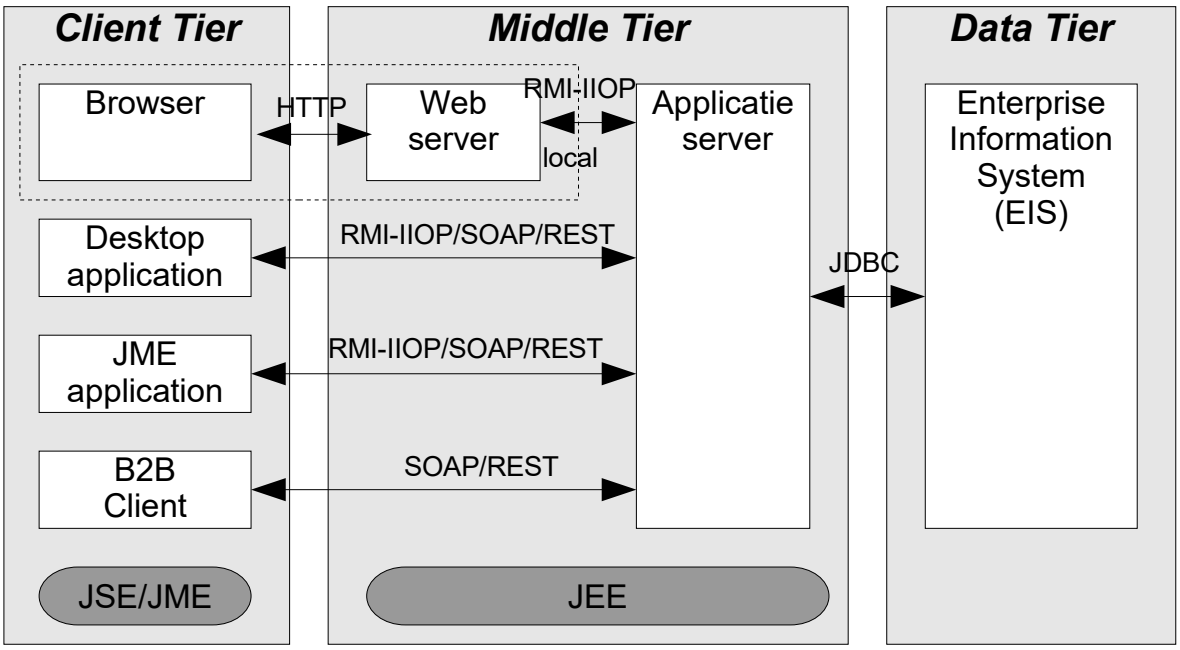
1. ***Data Access Layer***: Deze laag is verantwoordelijk voor de communicatie met de databank. De toegang tot de databank is hier gecentraliseerd en afgescheiden van

de rest.

2. ***Server Layer***: In deze laag wordt de *business logic* uitgevoerd. Deze bestaat uit allerlei diensten (*services*) ten behoeve van onder andere de *Presentation Layer*.
3. ***Presentation Layer***: Deze laag voorziet de presentatie van de applicatie naar de eindgebruiker toe. Tenminste in het geval er een grafische gebruikersinterface nodig is. Bij een B2B(*business to business)*-toepassing is dit niet noodzakelijk het geval.
4. ***Domain Objects:*** In de gehele applicatie zijn er meestal dataobjecten nodig. Deze worden in de *Data Access Layer* gesynchroniseerd met de databank. In de *Service Layer* worden deze objecten gemanipuleerd en in de *Presentation Layer* worden ze gebruikt om gegevens te tonen en nieuwe invoer van de gebruiker over te brengen naar de *Service Layer*. Deze objecten worden dus gebruikt in de drie andere lagen en om die reden hebben we ze in de afbeelding in een dwarsliggende balk getoond.

1.2 *Multitier*-applicaties in Java

Multitier-applicaties kunnen ontwikkeld worden in allerlei programmeertalen. Om dit soort applicaties ook te kunnen ontwikkelen in Java werd de *Java Enterprise Edition (JEE)* uitgebracht. Deze editie bevat de nodige functionaliteiten om vooral de *middle tier* te bouwen en om de communicatie hiermee te voorzien. We overlopen even de verschillende *tiers* zoals die in het Java platform gebruikt worden.



1.2.1 Java Client Tier

De *client tier* bevat de presentatielogica. In Java zijn er verschillende mogelijkheden om deze te bouwen.

1. ***Desktop***: Met *SWING* of *JavaFX* kunnen *standalone* applicaties ontwikkeld worden die de gebruiker een grafische interface biedt. De volledige presentatielogica wordt daarbij uitgevoerd op de machine van de gebruiker zelf. Om die reden noemt men dit ook wel een *medium client*. De *client*-toepassing bevat namelijk nog redelijk veel functionaliteit. De communicatie met de *middle tier* verloopt via het RMI-IIOP- of SOAP/REST-protocol.

2. ***Browser***: Een gebruikersinterface kan ook gepresenteerd worden door een browser die HTML-pagina's interpreteert en toont aan de gebruiker. Deze pagina's worden echter

gegenereerd door een webserver. De presentatielogica zit deels aan de kant van de *client* en deels aan de kant van de webserver. Vermits de *client* enkel in staat moet zijn HTML-pagina's te tonen, kunnen we hier echt spreken van een *thin client*. Eventueel kunnen de HTML-pagina's uitgebreid worden met *applets*, die wel lokaal uitgevoerd kunnen worden.
De communicatie met de webserver verloopt via het HTTP/HTTPS-protocol.

3. **J2ME:** De gebruikersinterface kan ook getoond worden op kleine toestellen zoals *handhelds, PDA's* en mobiele telefoons. In dit geval wordt gebruikgemaakt van de *Java Micro Edition (JME*). De communicatie met de applicatieserver verloopt ook hier via het RMI-IIOP- of SOAP/REST-protocol.

4. **B2B:** Ook *Business to business clients* kunnen gebruikmaken van de *middle tier*. Hier gaat het niet om grafische applicaties voor eindgebruikers maar eerder om andere *middleware*-systemen. Dergelijke systemen kunnen in andere programmeertalen geschreven zijn. De communicatie loopt dan bij voorkeur ook via SOAP of REST.

De technologieën voor het ontwikkelen van de *client tier* zijn beschikbaar in de *Java Standard Edition (JSE)* en de *Java Micro Edition (JME)*.

1.2.2 Java Middle Tier

De *middle tier* kan onderverdeeld worden in twee delen.

Vooreerst de **webserver** die de nodige HTML-pagina's genereert voor de browser-*client*. Bij eenvoudige webapplicaties, kan hierin ook de applicatielogica geïntegreerd worden. Hierdoor is deze logica echter niet herbruikbaar voor andere GUI-*clients*.
Bij een grotere *multitier*-applicatie is de webserver enkel verantwoordelijk voor het genereren van de webinterface. Alle andere taken worden overgelaten aan de applicatieserver. De communicatie met de applicatieserver verloopt doorgaans via RMI-IIOP. Rechtstreekse communicatie is evenwel ook mogelijk indien de webserver en applicatieserver geïntegreerd zijn en gebruikmaken van dezelfde virtuele machine.
De webserver wordt in JEE geïmplementeerd door de webcontainer die de Java webcomponenten bevat: *servlets,* JSP-pagina's en *Custom Tags*.

De **applicatieserver** bevat alle applicatielogica, ook wel *business logic* of *domain logic* genoemd. Hierin zit alle logica die te maken heeft met de functionaliteit en processen die niet gebonden zijn aan een specifieke gebruikersinterface. De componenten van de applicatieserver kunnen door middel van het RMI-IIOP- of SOAP-protocol aangesproken worden door allerlei *client*-applicaties.
De applicatieserver wordt geïmplementeerd door de EJB-container die *Enterprise JavaBeans* bevat. We komen hier later nog uitvoerig op terug.

Meestal zijn de webserver en applicatieserver verenigd in één server. We noemen dit een ***Enterprise Server***.

1.2.3 Enterprise Information System

De derde *tier* bevat de gegevens die in de applicatie gebruikt worden. De algemene term die hiervoor gebruikt wordt is *Enterprise Information System*, afgekort EIS. Doorgaans wordt deze *tier* geïmplementeerd door een databaseserver, maar het is ook mogelijk een koppeling te maken met andere soorten systemen voor gegevensbeheer. Het ontwikkelen van de EIS valt buiten het bestek van deze cursus. Het is uiteraard mogelijk deze systemen op basis van Java-technologie te ontwikkelen. De koppeling met EIS gebeurt in JEE applicaties door middel van de *Java Connector Architecture (JCA)*. JDBC is het meest gebruikte voorbeeld.

1.2.4 Interoperability met andere systemen

De middelste *tier* is bedoeld om de *business logic* te kunnen bevatten en deze functionaliteit

moet beschikbaar zijn voor de andere componenten, doorgaans de GUI-*clients*. Dit kunnen evenwel ook andere *middle-tier*-applicaties zijn die eventueel geschreven zijn in andere programmeertalen.

Om de functionaliteit universeel toegankelijk te maken is er in de JEE-specificatie gekozen voor het **RMI-IIOP**-protocol. Dit gebruikt namelijk op het laagste niveau IIOP voor de communicatie. IIOP is de afkorting van *Internet Inter-ORB Protocol* en wordt gebruikt door **CORBA** (*Common Object Request Broker Architecture*). Door gebruik te maken van dit protocol kunnen ook andere applicaties, geschreven in andere objectgeoriënteerde talen, gebruikmaken van de functionaliteit.

RMI-IIOP is momenteel in onbruik geraakt en wordt tegenwoordig vervangen door ***SOAP Web Services*** en ***RestFull Web Services***.

In deze cursus zullen we de drie technieken behandelen.

1.3 *JavaBeans* versus *Enterprise JavaBeans*

Enterprise JavaBeans zijn niet zomaar een uitbreiding van de klassieke *JavaBeans*. Ze hebben meer verschillen dan gelijkenissen. De gelijkenis zit in het feit dat beide bedoeld zijn als herbruikbare componenten die gebruikt kunnen worden binnen *Rapid Application Development (RAD).*

Klassieke *JavaBeans* moeten beantwoorden aan de *JavaBeans*-specificatie en worden vooral gebruikt bij de ontwikkeling van grafische applicaties. Een *JavaBean* kan hier namelijk de vorm van een grafische component aannemen die door IDE's herkend en gehanteerd wordt. Deze component kan dan met *drag and drop* op het ontwerpblad geplaatst worden en vervolgens ingesteld worden via zijn *properties*. Op die manier kan men snel nieuwe grafische toepassingen bouwen op basis van bestaande componenten. De componenten kunnen door de IDE makkelijk aan elkaar gekoppeld worden.

Ook binnen de webontwikkeling wordt vaak gebruikgemaakt van *JavaBeans*, maar dan in hun meer afgeslankte vorm. Men gebruikt hier vooral het patroon van *getters* en *setters*, terwijl de grafisch kant, de *bound* en *constraint properties* achterwege gelaten worden.

Enterprise JavaBeans zijn ook bedoeld als herbruikbare Java-componenten. In dit geval componenten voor de *middleware*. En daar houdt de vergelijking op. *Enterprise JavaBeans* moeten beantwoorden aan de EJB-specificaties, die geheel verschillend zijn van de *JavaBeans*-specificaties. Wat die specificaties zijn, zal in het vervolg van de cursus uitvoerig aan bod komen.

Hoofdstuk 2: EJB-Containers

2.1 Enterprise-servers en containers

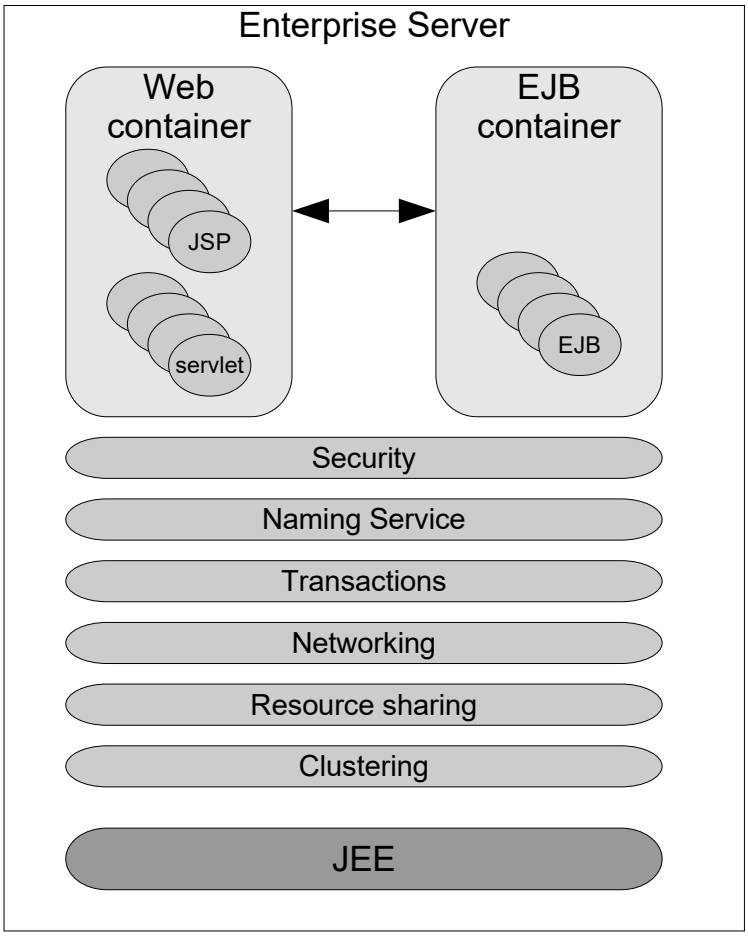
Voor het ontwikkelen van de *middleware* in een *multitier* gedistribueerde applicatie zijn er dus heel wat technologieën in het spel. Tevens is het zo dat veel functionaliteiten bij elke applicatie terugkomen en nagenoeg identiek zijn. We noemen er even enkele op:

- De **netwerkcommunicatie**: De componenten in een gedistribueerde applicatie zijn toegankelijk via het netwerk. Dit impliceert dat netwerkverbindingen opgezet moeten worden en dat *method calls* over deze verbinding verstuurd en ontvangen moeten worden.
- De **simultane toegang**: De *middleware* kan gebruikt worden door verschillende *client*-applicaties die allemaal gelijktijdig beroep kunnen doen op deze functionaliteit. Hiervoor moeten in de applicatieserver de nodige voorzieningen zijn om dit probleemloos te laten verlopen.
- De **beveiliging:** Applicaties dienen vaak beveiligd te worden tegen onrechtmatig gebruik. Dit impliceert authenticatie, autorisatie en eventueel encryptie.
- Het **transactiebeheer**: Sommige activiteiten moeten uitgevoerd worden binnen dezelfde transactie die eventueel veilig teruggedraaid kan worden. Het beheer van transacties wordt complexer naarmate er meer verschillende componenten deel van uit moeten maken.
- ***Load balancing***: Bij een zware belasting van de applicatieserver kan men eventueel gebruikmaken van *clusters* die de werklast verdelen.
- **Delen van resources**: Sommige netwerk-*resources* zoals databankconnecties, mailconnecties en dergelijke kunnen best gedeeld worden door de verschillende componenten. Dit resulteert in een lagere systeembelasting en hogere toegangssnelheid.
- ***Naming services***: Bij gedistribueerde applicaties moeten de componenten geregistreerd worden in een of andere *Naming Service* zodat ze daar door andere componenten opgezocht kunnen worden.

Voor de ontwikkelaar van een gedistribueerde applicatie zou het telkens een hele klus zijn om voor iedere applicatie deze onderdelen te voorzien. Het zou beter zijn als hij zich enkel zou kunnen concentreren op de specifieke *business logic* van zijn applicatie en voor de rest beroep doen op de aangeboden gemeenschappelijke functionaliteit. Dit is net de taak van de applicatieservers: zij bieden de bovenstaande diensten aan. De ontwikkelaar hoeft enkel de componenten met de pure specifieke logica te schrijven en deze vervolgens af te leveren aan de server. Deze server voorziet in een container waarin deze componenten geplaatst kunnen worden. Via de container gebruiken de componenten de mogelijkheden van de server op een uniforme wijze.

De webserver voorziet in een webcontainer. Hierin kunnen de webcomponenten (*servlets*, *JSP*, *Custom Tags*) geplaatst worden die uiteindelijk de webapplicatie vormen.

Op dezelfde manier voorziet de applicatieserver in een EJB-container. Deze bevat *Enterprise JavaBeans* die de echte business logica van de applicatie bevatten. Deze EJB's maken ook hier gebruik van alle functionaliteit die de container aanbiedt. We zullen later in detail zien waaruit dat allemaal bestaat.



Sommige servers bevatten enkel een webcontainer. Dit wordt vooral gebruikt bij eenvoudige webapplicaties. Voorbeelden van op Java gebaseerde webservers met een JEE-webcontainer zijn:

- *Tomcat* (open source)
- *Resin (www.caucho.com)*
- *ServletExec* (uitbreiding op IIS)

Daarnaast hebben we servers die zowel een webcontainer als een EJB-container hebben. Die zijn in staat volledige *enterprise*-applicaties te draaien. Servers met enkel EJB-containers zijn in principe mogelijk maar komen in de praktijk minder voor. De grote spelers op de markt zijn momenteel:

- *BEA Weblogic*
- *IBM WebSphere*
- *SUN Application Server*
- *GlassFish* (open source)
- *WildFly* /WildFly(open source)
- *JOnAS* (open source)

De specificaties van de containers zijn vastgelegd in de JEE-standaard (*Java Enterprise Edition)*. Dit maakt dat verschillende leveranciers een eigen *enterprise server* kunnen ontwikkelen en dat applicaties makkelijk gemigreerd kunnen worden naar een andere omgeving.

Zo kan een webapplicatie in de vorm van een WAR-bestand, in om het even welke

webcontainer geplaatst worden. Via de bijhorende *deployment descriptor (web.xml)* of annotaties wordt heel de webapplicatie geïntegreerd en geconfigureerd in de webserver.

Op gelijkaardige manier kunnen EJB's in de vorm van een JAR-bestand in de EJB-container geplaatst worden. Ook hier zorgen de *deployment descriptors* of annotaties ervoor dat deze *beans* geïntegreerd en geconfigureerd worden in de applicatieserver.

Door het gebruik van Java in combinatie met gestandaardiseerde specificaties voor de container bekomt men een hoge graad van onafhankelijkheid voor JEE-applicaties:

1. Onafhankelijkheid van de **hardware**: Java-applicaties kunnen uitgevoerd worden op alle soorten hardware, onder de voorwaarde dat er een virtuele machine beschikbaar is.
2. Onafhankelijkheid van het **besturingssysteem**: Voor nagenoeg alle belangrijke besturingssystemen zijn er virtuele machines beschikbaar.
3. Onafhankelijkheid van de ***enterprise server***: Alle JEE-*enterprise*-servers moeten aan dezelfde specificaties voldoen. Applicaties die dus gemaakt zijn volgens deze specificaties kunnen in principe makkelijk overgedragen worden naar een andere *enterprise*-server.

Deze onafhankelijkheid maakt dat men vrij is in de keuze van hardware, besturingssysteem en *enterprise*-server. Dit wakkert tevens de concurrentie aan op de markt van *enterprise*-servers hetgeen resulteert in competitiviteit op vlak van functionaliteit, performantie en stabiliteit.

De JEE-standaard is inmiddels aanbeland bij versie 8. EJB 3.2 maakt onderdeel uit van JEE 8 en we zullen in deze cursus deze versie behandelen.

2.2 Wildfly

In vorige paragraaf hebben we reeds de grote spelers op de markt van de *enterprise*-servers aangehaald. De meeste ervan zijn commerciële producten die vaak veel geld kosten. *WildFly* (voorheen *WildFly*) daarentegen is een *open-source*-project dat aanvankelijk begonnen is als alleenstaande EJB-container, maar dat intussen uitgegroeid is tot een volledige *enterprise*-server met alle noodzakelijke nevendiensten. *WildFly* bevat ook een webcontainer.

WildFly is beschikbaar op de 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 11*** in combinatie met ***WildFly 16.0.0.Final.***
Andere versies van *WildFly* kunnen een andere configuratie hebben.

2.2.1 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* installeren.

- Haal versie 16.0.0 van ***WildFly*** af van de website http://wildfly.org/downloads/ (***WildFly-16.0.0-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-16.0.0.Final

2.2.2 Configuratie

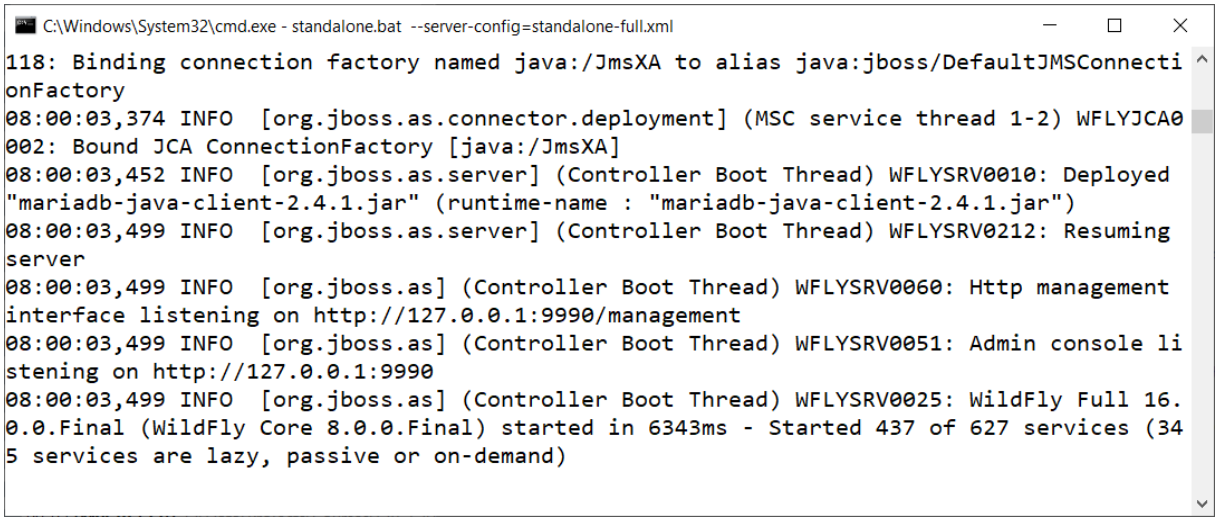
Na de installatie vinden we de volgende mappenstructuur:

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

Folder	***Omschrijving***
`bin`	Hierin bevinden zich de *batch*- en *shell*-bestanden voor het opstarten en afsluiten van *WildFly*.
`docs`	Hierin bevindt zich documentatie, waaronder de DTD's en schema's van bepaalde XML-bestanden.
`domain`	Bevat de configuratie voor *WildFly* in *domain*-mode.
`modules`	Bevat extra JAR-bestanden voor modules.
`standalone`	Bevat de configuratie voor *WildFly* in *standalone*-mode.
`configuration`	Hierin bevinden zich de configuratiebestanden van deze *WildFly*-configuratie. Dit zijn hoofdzakelijk XML-bestanden.
`data`	Bevat gegevensbestanden.
`deployments`	In deze folder kunnen de webapplicaties, EJB's en volledige *enterprise*-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 wordt hierbij de configuratie genomen uit het bestand ***standalone.xml.*** Indien men gebruik wil maken van alle functionaliteiten uit de JEE-standaard dient men evenwel het configuratiebestand ***standalone-full.xml*** te gebruiken:

```
standalone.bat --server-config=standalone-full.xml
```

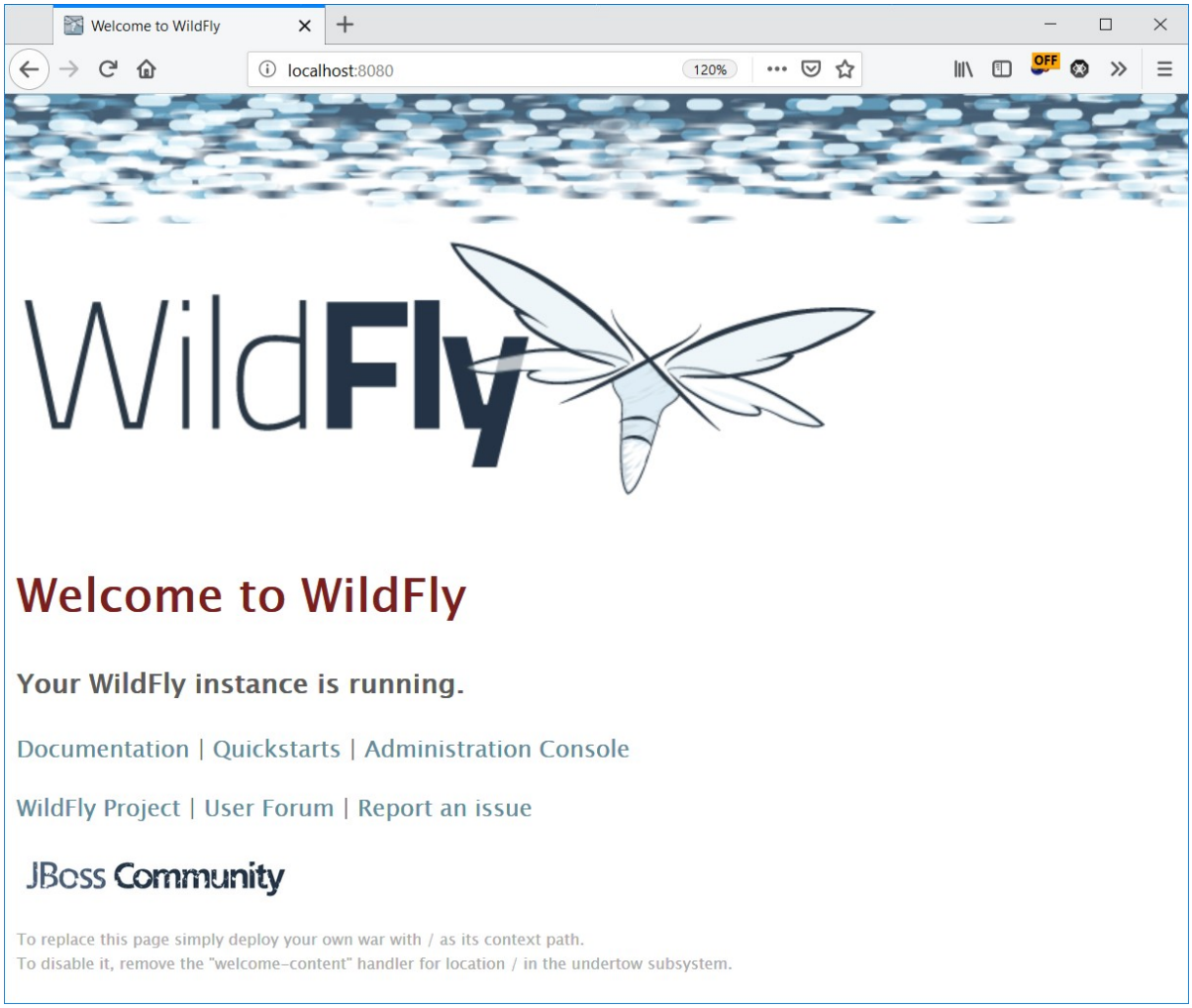


Standaard is *WildFly* om veiligheidsredenen enkel toegankelijk via de *localhost* (127.0.0.1) netwerkinterface. Andere IP-adressen kunnen opgegeven worden via de optie `-b w.x.y.z` Bij 0.0.0.0 worden alle IP-adressen gebruikt.

```
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 de 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

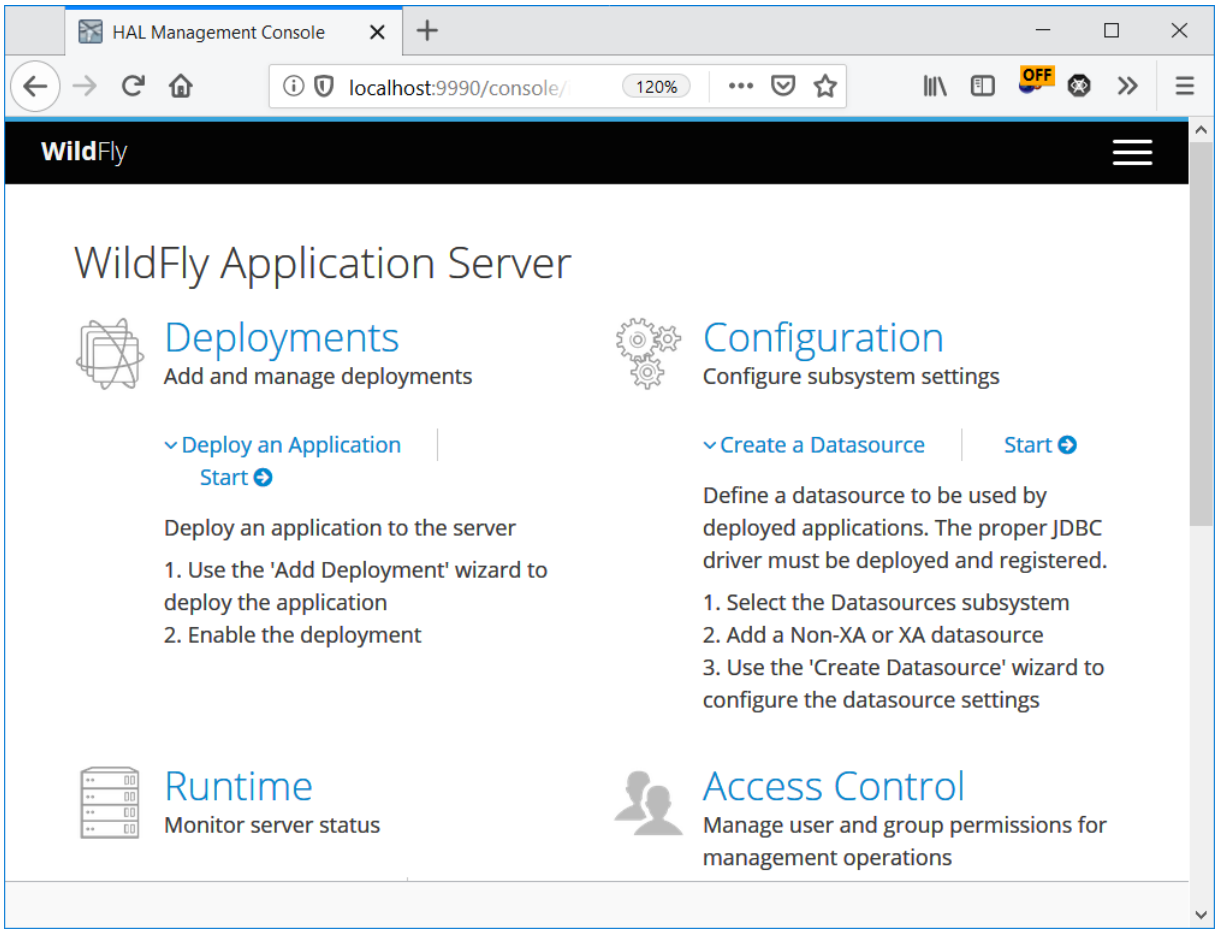
  Selecteer *Management User* en gebruik als naam ***admin*** met wachtwoord ***wildfly***

- Voeg een tweede gebruiker toe: selecteert hierbij *Application User* en gebruik als naam ***user*** en wachtwoord ***password***. Voeg deze gebruiker tevens toe aan de groep ***guest.***

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

```
standalone.bat --server-config=standalone-full.xml
```

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



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

2.2.3 Integratie in een IDE

De meeste IDE's bieden de mogelijkheid *WildFly* op te starten vanuit de omgeving van de IDE. Dit heeft als voordeel dat alles vanuit één ontwikkelomgeving kan gebeuren. We verwijzen naar de documentatie van de IDE om dit te configureren.

Tevens bieden de IDE's de mogelijkheid om automatisch een toepassing in werking te stellen op de applicatieserver. In deze cursus zullen we dit evenwel telkens doen met een *Maven*-plugin.

Voor het vervolg van de cursus dient *WildFly* opgestart worden vanuit het consolevenster of de IDE maar het in werking stellen (*deployen*) van een toepassing gebeurt telkens via *Maven.*

Hoofdstuk 3: EJB-architectuur

3.1 Onderdelen van de business-logica

Enterprise JavaBeans zijn de componenten die de *business logic* bevatten van de gedistribueerde applicatie. Deze componenten worden in de container van de applicatieserver geplaatst.
Deze *business logic* kunnen we doorgaans onderverdelen in twee onderdelen:

1. **Activiteiten**: Er zijn een aantal activiteiten of processen nodig die de taken uitvoeren die aangevraagd worden door de *client*-applicatie.
2. **Entiteiten**: Er zijn een aantal gegevens die gebruikt en gewijzigd worden tijdens die activiteiten.

Tijdens de UML-analyse kan men beide onderscheiden door het woordgebruik.
Werkwoorden duiden op activiteiten terwijl zelfstandige naamwoorden duiden op entiteiten.

3.2 Soorten Enterprise JavaBeans

De *Enterprise JavaBeans (EJB)* kunnen we ook indelen volgens deze twee grote categorieën: sommige EJB's vertegenwoordigen activiteiten en andere entiteiten.

We onderscheiden drie soorten EJB's:

1. **Session Beans**: Dit zijn EJB's die bedoeld zijn om allerlei activiteiten en processen uit te voeren binnen de applicatie. Zulke *session beans* zijn gebonden aan een bepaalde *client*-sessie en op die manier zijn ze eigenlijk een verlengstuk van de *client* die ze gebruikt. We onderscheiden hier verder nog ***stateful** session beans, **stateless** session beans* en ***singleton** session beans.* De *stateful session beans* houden tussen twee oproepen statusinformatie bij, terwijl de *stateless session beans* dat niet doen. *Singleton session beans* voorzien in één unieke instantie per virtuele machine.
   Session beans kunnen zowel op synchrone als asynchrone wijze aangesproken worden door de *client*-applicatie.

2. **Message Driven Beans**: Deze EJB's zijn net als de *session beans* bedoeld om activiteiten en processen uit te voeren. Zij zijn evenwel niet rechtstreeks toegankelijk, maar kunnen enkel aangesproken worden door er een asynchrone boodschap naartoe te sturen via een boodschappensysteem. Doorgaans gebeurt dit via JMS (*Java Messaging Service*).

3. **Entity Beans**: Dit zijn EJB's die de gegevens (*entities*) voorstellen in een applicatie. Doorgaans stellen *entity beans* databaserecords voor waarbij gegevens makkelijk opgevraagd en ingesteld kunnen worden via *getters* en *setters*. De synchronisatie met de database gebeurt door middel van een *entity manager.* Er wordt hier gebruikgemaakt van de *Java Persistence API (JPA)* die deel uitmaakt van de EJB-specificatie.

In het verdere verloop van deze cursus zullen we de verschillende soorten EJB's uitvoerig behandelen.