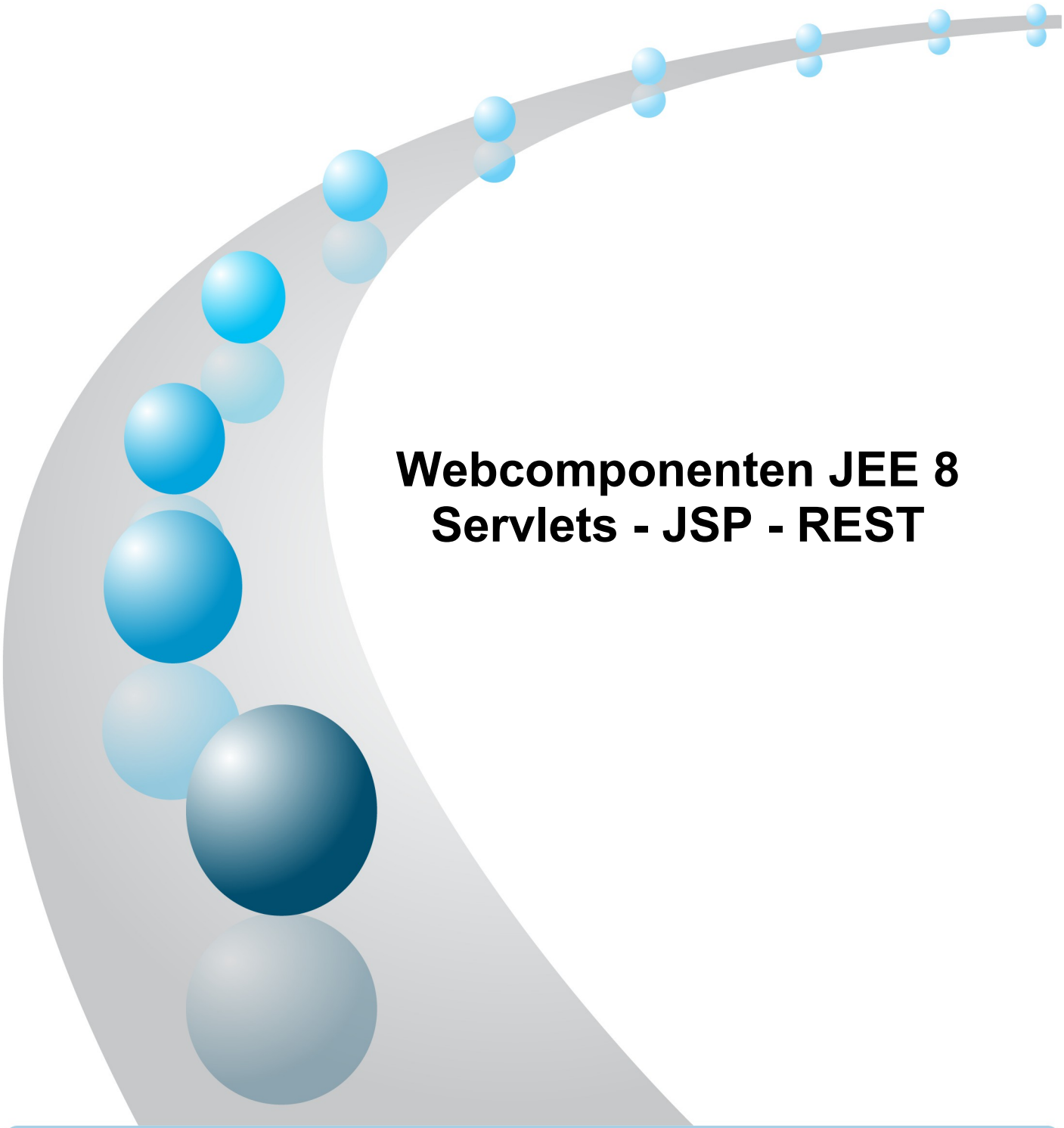
Noël Vaes

Java Trainer & Consultant



Webcomponenten JEE 8
Servlets - JSP - REST

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

11/08/2019

Inhoudsopgave

Hoofdstuk 1. Webcontainers

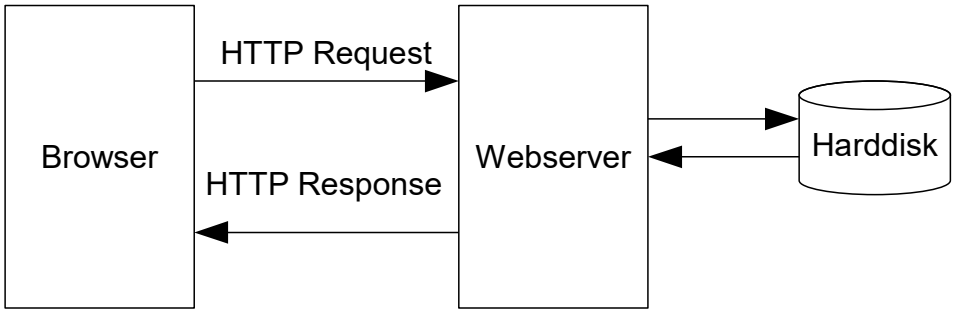
1.1. Inleiding

Het internet heeft de laatste jaren een enorme ontwikkeling gekend en ook de ontwikkeling van complexere websites neemt steeds maar toe. Waar de oorspronkelijke websites vooral uit statische pagina's bestonden, zijn we nu geëvolueerd naar meer dynamische sites. In dergelijke websites speelt de ontwikkeling van de specifieke software een belangrijke rol. Ook Java biedt voor dit soort toepassingen een waaier aan mogelijkheden. Webservers die beschikken over een Java-webcontainer kunnen volop gebruikmaken van de mogelijkheden die de Java-programmeertaal biedt voor het ontwerpen van complexe en dynamische websites.

1.2. Statische versus dynamische webpagina's

Het internet is een *client-server*-omgeving waarbij de *client* bestaat uit een browser die in staat is HTML-pagina's weer te geven en een *webserver* die de HTML-pagina's aan de browser levert.

De browser vraagt hierbij een bepaald HTML-document op door middel van een URL (*Uniform Resource Locator*). De webserver leest het betreffende bestand van het lokale bestandssysteem en stuurt de inhoud naar de browser. Beide communiceren met het HTTP-protocol.

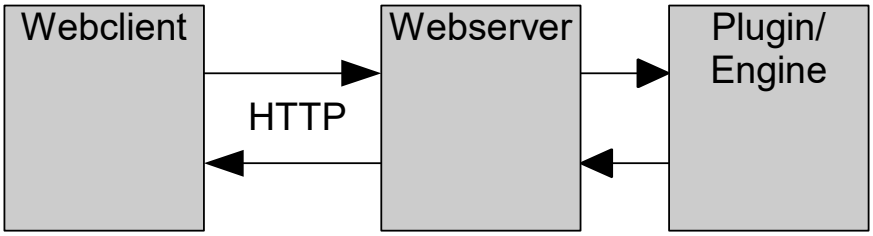


Bij dit mechanisme kan de webserver enkel statische HTML-pagina's afleveren aan de browser. Voor meer geavanceerde toepassingen is dit ontoereikend. Daarom werd dit uitgebreid met de mogelijkheid om HTML-pagina's dynamisch te genereren. De pagina's zijn hierbij niet als dusdanig opgeslagen op de harde schijf maar worden aangemaakt op het moment dat de vraag komt en kunnen op die manier ook op maat gemaakt worden.

In eerste instantie werd het aanmaken van dynamische pagina's uitbesteed aan externe applicaties. De communicatie tussen webserver en deze externe applicaties verloopt via CGI (*Common Gateway Interface*). CGI is een gestandaardiseerde interface die het mogelijk maakt een koppeling te maken tussen een webserver en een externe applicatie. Het gebruik van CGI had echter wel een aantal nadelen. Voor ieder verzoek van de browser werd namelijk een volledig nieuw proces (*heavyweight process*) opgestart dat dit verzoek moest afhandelen. Dit opstarten van zo'n nieuw proces resulteert in een zware belasting van de systeembronnen en is daardoor minder geschikt indien meerdere gebruikers tegelijkertijd een verzoek sturen naar de webserver.

Als alternatief voor CGI ontstonden allerlei technologieën waarbij een module (*lightweight process*) kan worden toegevoegd aan de bestaande webserver.

De webserver wordt hierbij voorzien van een extra module of *plugin (engine)* die in staat is HTML-documenten dynamisch te genereren. De webserver stuurt het verzoek van een *client* door naar deze module die op haar beurt de HTML-pagina onmiddellijk (*at runtime*) genereert. Hierbij kunnen dan bijvoorbeeld gegevens uit een databank opgenomen worden in het document.



Voorbeelden van dit soort technologieën zijn:

- **PERL**: scripttaal
- **PHP**: scripttaal
- **ASP** (*Active Server Pages*): Maakt gebruik van scripttalen als *Visual Basic Script* of *JavaScript*. Is specifiek voor *Internet Information Services (IIS)* van *Microsoft*.
- **ISAPI**: Een op maat gemaakte module (DLL) specifiek voor *Internet Information Services* (IIS).
- **NSAPI**: Een op maat gemaakte module specifiek voor *Netscape Server.*
- **Servlets/JSP** (*Java Server Pages*): Voor alle webservers die Java ondersteunen.

Er zijn dus momenteel een handvol technologieën waarmee men dynamische webpagina's kan maken. De meeste van die technologieën zijn heel specifiek voor een bepaalde programmeeromgeving. Indien men bijvoorbeeld gebruik maakt van *Internet Information Services (IIS)* van *Microsoft,* kan men zijn toevlucht nemen tot *Active Server Pages* of tot *ISAPI* DLL's. De zo ontwikkelde websites kunnen echter enkel gebruikmaken van IIS en zijn niet compatibel met andere webservers.

In de Java-wereld huldigt men echter het principe *Write Once Run Anywhere (WORA)*. Java is een programmeertaal die platformonafhankelijk is en mede daardoor heeft deze taal de laatste jaren een ongekende opmars doorgemaakt.

Ook voor het ontwikkelen van websites biedt Java platformonafhankelijke technologieën aan: *servlets, Java Server Pages* en aanverwante technologieën. Dit maakt het mogelijk webapplicaties te ontwikkelen die onafhankelijk zijn van het platform waarop de webserver draait en ook onafhankelijk van de webserver zelf.

Momenteel is de Java-technologie een van de meest gebruikte technologieën voor het ontwikkelen van dynamische websites.

1.3. Java Enterprise Edition

Het Java-platform kent drie edities:

1. ***Java Standard Edition* (JSE)**. Dit platform wordt vooral gebruikt voor het uitvoeren van *standalone*-applicaties.
2. ***Java Enterprise Edition* (JEE)**. Deze editie voegt een hele reeks technologieën toe aan de JSE die het mogelijk maken applicaties te ontwikkelen in een complexe *client-server*-omgeving. Deze editie vooronderstelt steeds de aanwezigheid van JSE.

3. ***Java Micro Edition* (JME)**: Dit is een afgeslankte vorm van het platform bedoeld voor software op kleine toestellen zoals handhelds, mobiele telefoons enzovoort.

De technologieën die nodig zijn voor het ontwikkelen van dynamische websites zijn ondergebracht in JEE. JEE bevat echter nog veel meer technologieën voor het ontwikkelen van *Enterprise-a*pplicaties met behulp van onder andere *Enterprise JavaBeans, CDI, JavaServer Faces* enzovoort. Dit valt echter buiten het bestek van deze cursus. We beperken ons tot het ontwikkelen van allerlei webcomponenten met het JEE-platform.

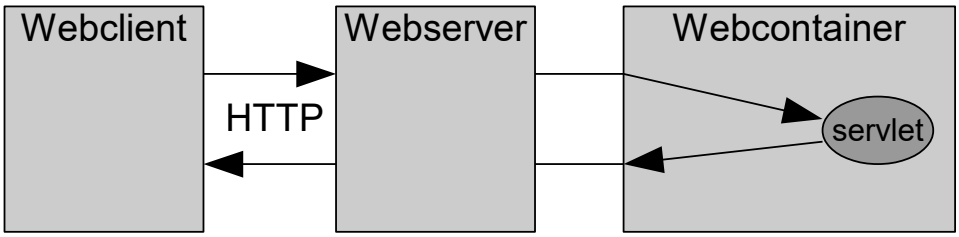
Het JEE-platform kan gedownload worden vanaf de Oracle-website: http://java.oracle.com*.* Hierin bevinden zich de nodige bibliotheken en ontwikkeltools. Voor het ontwikkelen van webcomponenten is de installatie van het volledige JEE-platform niet echt noodzakelijk. De nodige bibliotheken worden aangeleverd door de webcontainer die de JEE-specificaties implementeert.

In deze cursus maken we gebruik van JSE 11 en JEE 8. De API-documentatie van JEE 8 is te vinden op de volgende website: https://javaee.github.io/javaee-spec/javadocs/ .

1.4. Java-webcontainers

De *plugin* of *engine* waar in vorige figuur sprake van is, wordt in de Java-technologie de webcontainer genoemd. Deze webcontainer is een extra module die deel uitmaakt van de webserver of die als extra module aan een bestaande webserver kan worden toegevoegd. Voor webservers die zelf in Java geschreven zijn, is deze container meestal een onderdeel van de webserver (bijvoorbeeld *Apache Tomcat*).

Zoals het woord web*container* zelf zegt, is deze module een container of een verzameling van andere componenten. Deze componenten zijn onder andere de ***servlets****:* kleine server-toepassingen die geschreven zijn in Java *(servlet* is in het Engels het verkleinwoord van *server).*



Indien de webserver een specifieke vraag krijgt van een webclient, wordt deze aanvraag doorgestuurd naar de webcontainer. De webcontainer beslist onder andere op basis van de URL naar welke *servlet* deze vraag gestuurd wordt. De *servlet* genereert vervolgens de HTML-pagina en levert die af aan de webcontainer die ze op zijn beurt doorgeeft aan de webserver.

Enterprise-servers bevatten zowel een webcontainer als een EJB-container en een CDI-container. Deze laatste twee worden gebruikt voor *Enterprise JavaBeans (EJB)* en *Contexts and Dependeny Injection (CDI).*

Voor het ontwikkelen van webcomponenten volstaat echter een webcontainer.

Er zijn uiteraard verschillende implementaties van webcontainers. Aangezien ze alle aan dezelfde standaard moeten voldoen die door de JEE-specificaties wordt bepaald, zijn webapplicaties in principe volledig overdraagbaar tussen verschillende webcontainers. Momenteel zijn er verschillende implementaties beschikbaar:

- ***WebLogic***: Commerciële *enterprise*-server van Oracle (www.oracle.com).
- ***WebSphere***: Commerciële *enterprise*-server van IBM (www.ibm.com).
- ***Resin****:* Een commerciële webcontainer van Caucho (*www.caucho.com*).
- ***Jetty:*** *Open-source*-webcontainer van *Eclipse* (www.eclipse.org).
- ***Tomcat***: Populaire *open-source*-webcontainer van Apache (*tomcat.apache.org*).
- ***TomEE***: Uitbreiding op *Tomcat* met andere JEE-bibliotheken (*tomee.apache.org*).
- ***WildFly****: Open-source-enterprise*-server met ingebouwde webcontainer (www.wildfly.org).
- ***GlassFish:*** *Open-source-enterprise*-server gepromoot door Oracle (***https://glassfish.java.net/***) die tevens gebruikt wordt als referentie.

1.5. *Apache Tomcat/TomEE*

We gaan in deze cursus gebruikmaken van de populaire *open-source*-webserver *TomEE* van *Apache*. Dit is een uitbreiding op de alom gekende *Tomcat.* Bij *Tomcat* ontbreken namelijk een aantal interessante functionaliteiten die deel uitmaken van de JEE-specificatie. *TomEE* voegt deze functionaliteiten toe.

Deze webserver kan vrij van het internet geplukt worden op de volgende website: ***http://tomee.apache.org***.

1.5.1. Installatie

TomEE is volledig in Java geschreven en kan bijgevolg werken op elk platform dat Java ondersteunt.

In deze cursus maken we gebruik van *TomEE Plus 8*. Deze versie ondersteunt de volgende JEE8-specificaties:

Servlets 4.0
JSP 2.3
Expression Language 3.0

Voor *Windows* bestaat er een *executable* die de installatie makkelijk maakt en die bovendien een *service* voor *TomEE* kan installeren.

Opdracht 1: TomEE installeren

In deze opdracht gaan we *TomEE 8.x,* op onze computer installeren. We gaan er hierbij vanuit dat de JDK (JSE 11) reeds geïnstalleerd is.
Tevens voorzien we een omgevingsvariabele ***TOMCAT_HOME*** die het pad naar de installatiemap bevat. Deze variabele zullen we later gebruiken in het *Maven* POM-bestand.

- Haal *TomEE Plus 8.x* van de *Apache*-website: http://tomee.apache.org.
- Pak het zip-bestand uit in een lokale map: bijvoorbeeld onder **C:**.
- Voeg in het besturingssysteem een omgevingsvariabele met de naam ***TOMCAT_HOME*** toe die het pad naar de installatiemap bevat (bijvoorbeeld ***C:\apache-tomee-plus-8.0.0).***
- Start *TomEE* op met het commando ***startup*** dat zich in de map ***bin*** van de installatie bevindt. Voor *Windows* is dat ***startup.bat***, voor *Unix/Linux* is dat ***startup.sh.***
- Open een browser en surf naar het volgende adres: ***http://localhost:8080.***
- Beëindig *TomEE* met het commando ***shutdown.bat*** of ***shutdown.sh***. Je mag ook gewoon CTRL-C gebruiken in het commandovenster waar je *TomEE* hebt opgestart.

1.5.2. Integratie in Eclipse

Voor het ontwikkelplatform *Eclipse* zijn er een aantal *plugins* voorhanden voor het beheer van *Tomcat/TomEE*. Deze maken deel uit van het *Eclipse Web Tools Platform (WTP)*. Dit is een reeks van *plugins* voor het ontwikkelen van webapplicaties. WTP is een standaard onderdeel van *Eclipse IDE for Java EE Developers*.

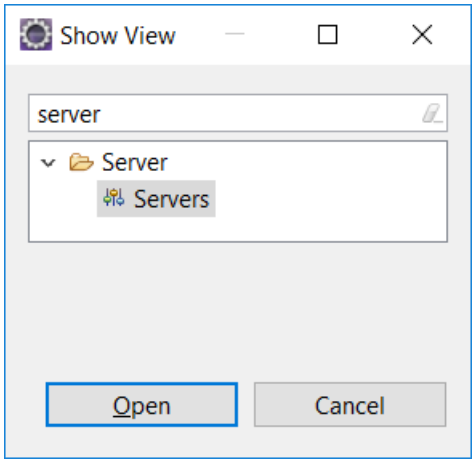
Met deze *plugins* is het mogelijk *Tomcat/TomEE* te starten en te stoppen vanuit *Eclipse*. Dit heeft onder andere als voordeel dat ook de *logging* van *Tomcat/TomEE* verschijnt in een venster van *Eclipse,* hetgeen erg handig is bij het *debuggen* van een webapplicatie.

De *plugins* voorzien ook nog allerlei andere mogelijkheden om onder andere een webproject te maken dat automatisch in *Tomcat/TomEE* geconfigureerd wordt.

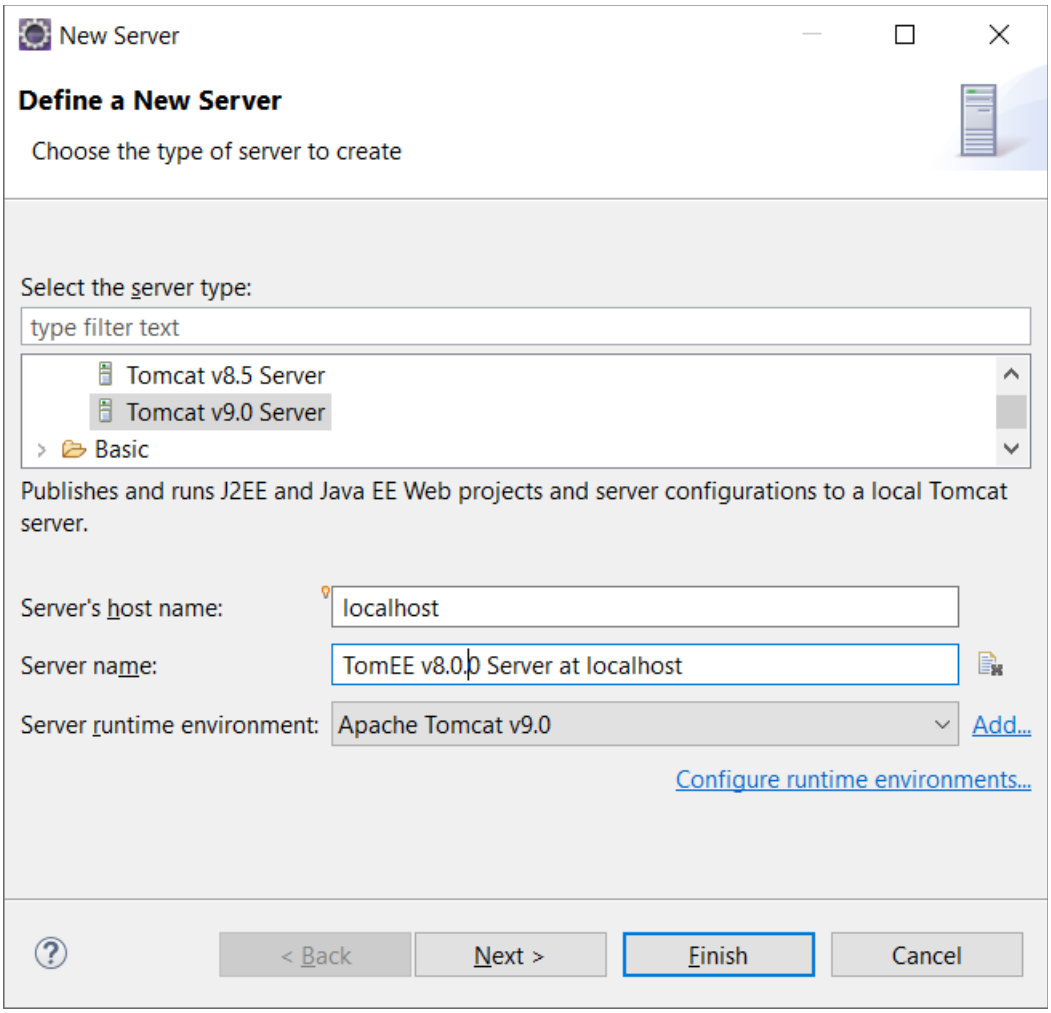
Opdracht 2: TomEE configureren in Eclipse

In deze opdracht configureren we *Eclipse* zodat we *TomEE* vanuit onze ontwikkelomgeving kunnen opstarten en configureren.

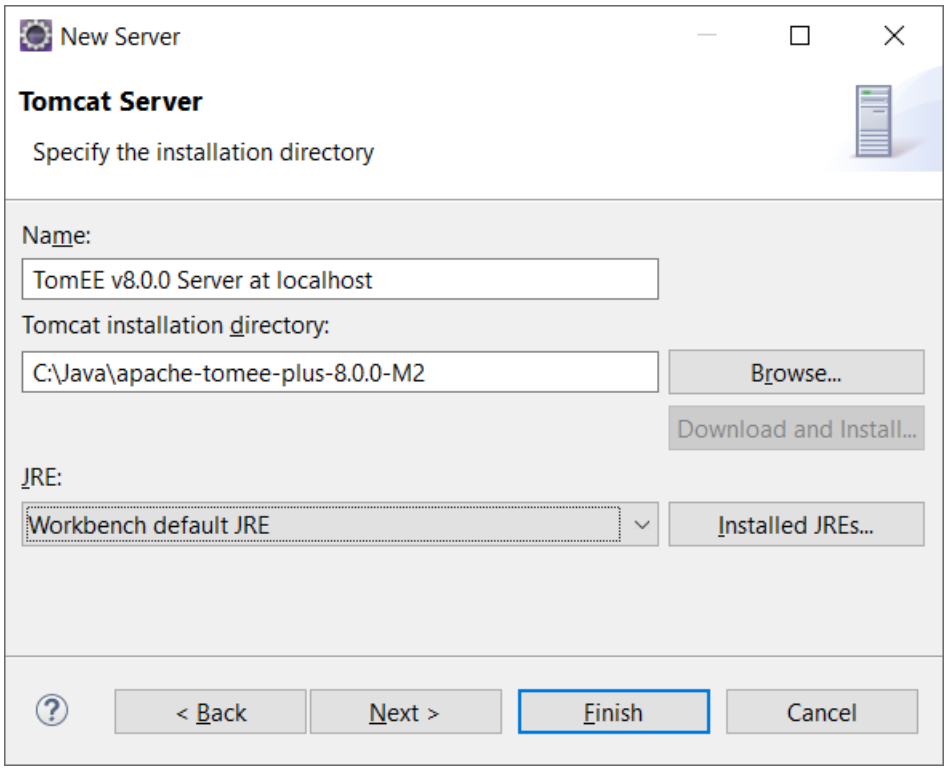
- Start *Eclipse* op.
- Voeg de volgende *view* aan de werkomgeving toe: ***Servers/Server.***



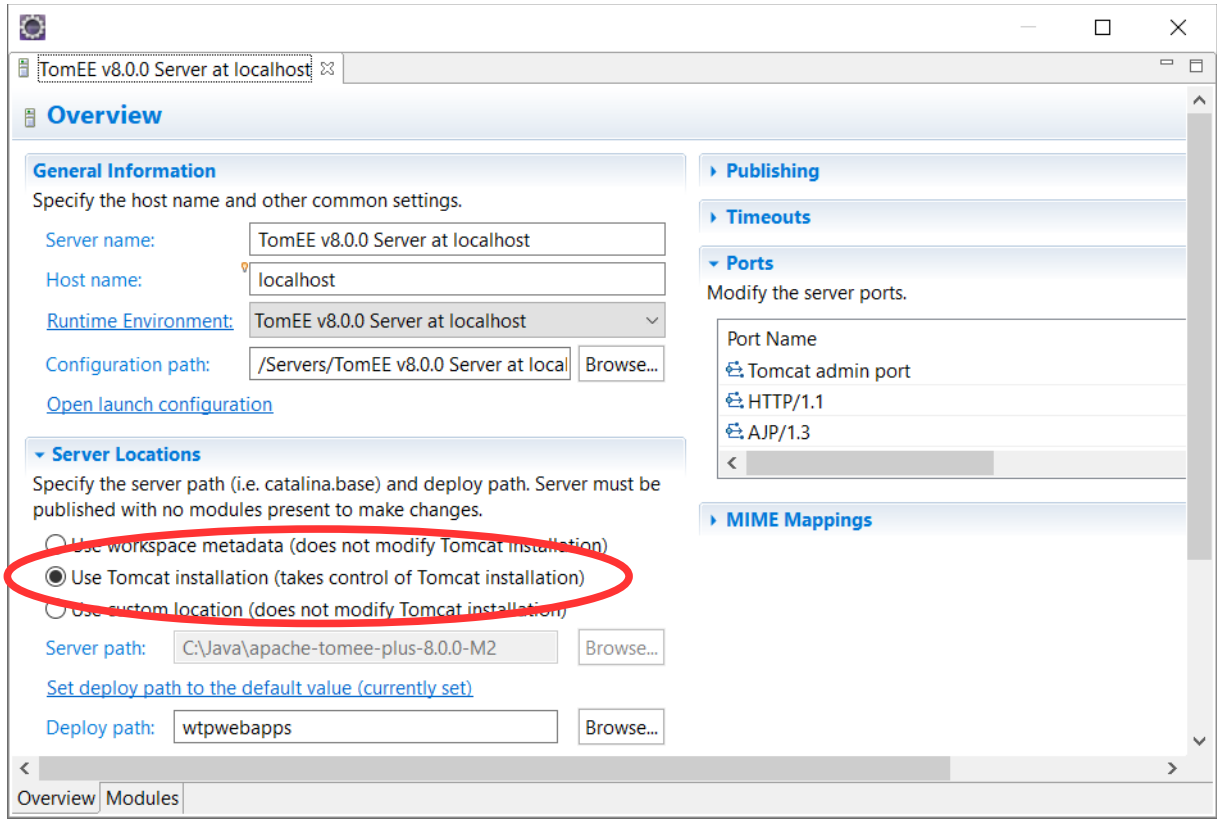
- Selecteer vervolgens in deze *view* ***New->Server***.
- Kies de ***Tomcat v9.0*** server. *TomEE 8* is namelijk gebaseerd op *Tomcat 9.0*.



- Eventueel kan je ***Server name*** aanpassen naar ***TomEE v8.0.0 Server at localhost.***
- Klik daarna op ***Next.***



- Geef hier de installatiemap van *TomEE* op en klik op ***Finish***.
- Dubbelklik op de configuratie om het venster met instellingen te openen.



- Selecteer hier de optie ***Use Tomcat installation (takes control of Tomcat installation)***.
- Start *TomEE* nu vanuit de *view* door de server te selecteren en vervolgens op de groene pijl te klikken (of via het context-menu).
- Open een browser en surf naar het volgende adres: ***http://localhost:8080***. *TomEE* zou nu correct opgestart moeten zijn.

1.5.3. Integratie in *IntelliJ*

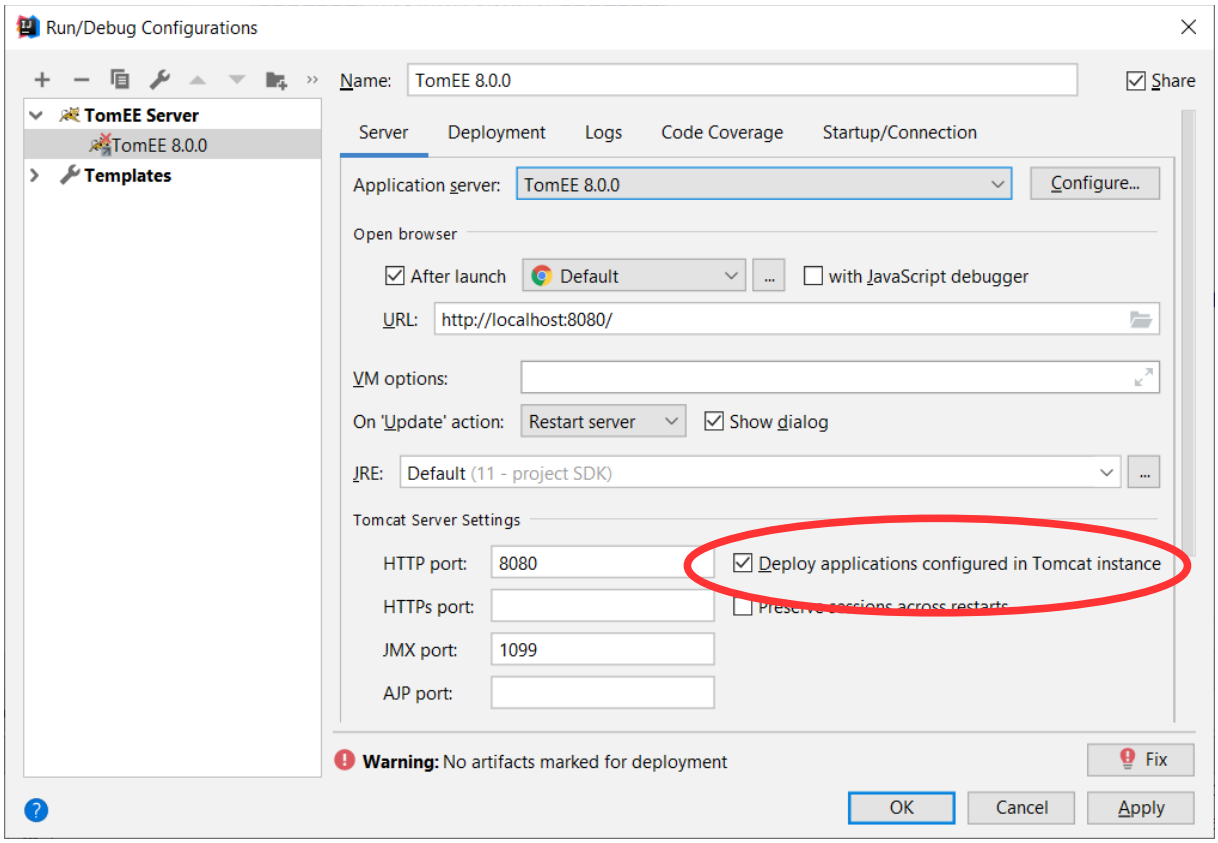
De *Ultimate Edition* van *IDEA IntelliJ* beschikt ook over een integratie van *Tomcat/TomEE.*

Indien je niet beschikt over deze (betalende) versie van *IntelliJ* kan je *TomEE* ook gewoon opstarten met het opdrachtbestand ***startup.bat*** of ***startup.sh***. In het vervolg van de cursus zullen we de integratie enkel gebruiken voor het opstarten van *TomEE* en het bekijken van de *logging*. Dit kan je ook doen vanuit het commandovenster.

Opdracht 3: TomEE configureren in IntelliJ Ultimate Edition

In deze opdracht configureren we *IntelliJ* zodat we *TomEE* vanuit onze ontwikkelomgeving kunnen opstarten en configureren.

- Start *IntelliJ*.
- Open het menu ***Run->Edit Configurations...***
- Selecteer ***TomEE Server*** en klik op het plusteken om een serverconfiguratie toe te voegen. Kies hierbij het profiel ***Local***.
- Vul de volgende gegevens in naargelang je lokale installatie van ***TomEE***:



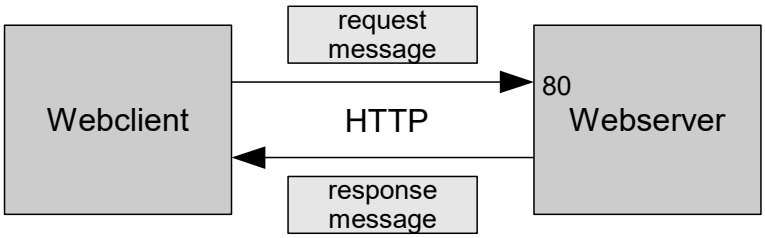
- Selecteer de optie ***Deploy applications configured in Tomcat instance***.
- Klik op ***Apply*** en vervolgens op ***Run***.
- Open een browser en surf naar het volgende adres: ***http://localhost:8080***. *TomEE* zou nu correct opgestart moeten zijn.

1.6. Het HTTP-protocol

In de communicatie tussen *webclient* en *webserver* wordt gebruikgemaakt van het ***HyperText Transfer Protocol*** afgekort ***HTTP***. Dit protocol maakt gebruik van het onderliggende TCP/IP-protocol voor het uitwisselen van de boodschappen.

Een *webserver* luistert standaard op TCP-poort 80. Een webclient (meestal een browser) maakt daarom een verbinding via poort 80 van de *webserver.* De *webclient* kiest voor zichzelf doorgaans een vrije poort boven 1024. Indien de webserver op een andere poort luistert, dient men deze te specificeren in de URL, bijvoorbeeld *http://localhost**:8080**/Webcomponenten/index.html.*

Het HTTP-protocol dient om allerlei gegevens van een webserver op te vragen. Hierbij stuurt de *webclient* een boodschap met een verzoek naar de *webserver* en deze antwoordt op zijn beurt met een boodschap waarin het antwoord vervat is. Het initiatief gaat daarbij steeds uit van de *webclient* die om informatie vraagt. Het HTTP-protocol houdt geen statusinformatie bij. Ieder verzoek wordt beschouwd als een afzonderlijk gebeuren.



We gaan nu even de inhoud van deze boodschappen verder onder de loep nemen.

1.6.1. Request message

Het verzoek dat de *webclient* naar de *webserver* stuurt, bestaat uit de volgende gegevens:

1. Initiële ***request line*** die de methode, de *request URI* en de versie van het protocol bevat.
2. Optioneel een of meerdere ***header lines*** die bestaan uit een *header*-naam en diens waarde.
3. Een **lege regel**.
4. Optioneel een ***message body*** die meer informatie bevat over het verzoek. Deze kan meerdere regels bevatten.

We geven een voorbeeld:

```
GET /path/to/file HTTP/1.1
Header1: value1
Header2: value2
```

Voor het HTTP 1.1-protocol zijn de volgende methoden vastgelegd:

Methode	***Omschrijving***
`GET`	Vraagt de inhoud van een bepaalde *resource* op. Eventuele parameters worden toegevoegd aan de URL en zijn daardoor ook beperkt in lengte.
`POST`	Stuurt gegevens naar de server met betrekking tot een bepaalde *resource*. Deze gegevens worden toegevoegd in de *body* van het verzoek en zijn in principe onbeperkt in lengte.
`HEAD`	Vraagt enkel de hoofding op van een bepaalde *resource* die door een GET verkregen zou worden; dit doorgaans om na te gaan of de gegevens uit de *cache* nog *up-to-date* zijn.
`PUT`	Stuurt gegevens naar de server met betrekking tot een bepaalde *resource*. Deze gegevens worden toegevoegd in de *body* van het verzoek en zijn in principe onbeperkt in lengte.
`DELETE`	Wist een bepaalde *resource* op de server.
`TRACE`	Stuurt gewoon het verzoek terug naar de *client* (echo). Dit wordt gebruikt om na te gaan of een bepaalde component correct functioneert.
`OPTIONS`	Geeft een lijst van beschikbare methoden voor een bepaalde *resource*.
`CONNECT`	Gereserveerd voor toekomstig gebruik.

Om te communiceren met een *webserver* wordt gebruikgemaakt van GET en POST. POST wordt onder andere gebruikt bij het verzenden van formulieren die veel informatie bevatten. De methode HEAD vraagt enkel de *header*-informatie op. Dit wordt onder andere gebruikt om na te gaan of de gegevens in de *cache* van de browser nog actueel zijn. De overige methoden worden bij gewone websites niet courant gebruikt en hun beschrijving valt buiten het bestek van deze cursus.

Na de methode volgt een spatie en het pad naar de informatie op de webserver. Men noemt dit de *request URI (Uniform Resource Identifier)* die de gegevens op de webserver uniek identificeert. Ten slotte wordt de eerste regel afgesloten met de versie van het HTTP-protocol. Dit is `HTTP/1.0`, `HTTP/1.1` of het nieuwe `HTTP/2.0`.

Indien het verzoek een *message body* heeft, dienen tevens de volgende *headers* aanwezig te zijn:

Content-Type: text/html
Content-Length: xxx

Deze *headers* geven meer informatie over het type en de lengte van de *message body*.

Afhankelijk van de browser worden nog allerlei andere *headers* meegegeven.

1.6.2. Response message

De *webserver* antwoordt op het verzoek van de *webclient* met een boodschap: de *response message*. Deze bestaat uit de volgende regels:

1. Een **initiële regel** met de protocol-versie, een statuscode en statusomschrijving.
2. Optioneel een of meerdere ***header lines*** die bestaan uit *header*-naam en diens waarde.
3. Een **lege regel**.
4. Optioneel een ***message body*** die het antwoord bevat van het verzoek. Deze kan meerdere regels omvatten.

We geven een voorbeeld:

```
HTTP/1.1 200 OK
ETag: W/"153-1077716422857"
Last-Modified: Wed, 25 Feb 2004 13:40:22 GMT
Content-Type: text/html
Content-Length: 153
Date: Wed, 27 Apr 2005 14:07:29 GMT
Server: Apache-Coyote/1.1
Connection: close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title></title>
</head>
<body>
Hello World!
</body>
</html>
```

De statuscode bestaat uit drie cijfers waarbij het eerste cijfer de categorie aangeeft:

Statuscode	*Omschrijving*
`1xx`	Enkel informatieve boodschappen.
`2xx`	Succesvol verzoek.
`3xx`	Doorverwijzing naar een andere URL.
`4xx`	Fout bij de *webclient.*
`5xx`	Fout bij de *webserver.*

Indien het antwoord een *message body* heeft, dienen ook de volgende *headers* aanwezig te zijn:

Content-Type: text/html *(of iets anders)*
Content-Length: xxx

Deze *headers* geven meer informatie over het type en de lengte van de *message body*.

Andere *headers* geven informatie over onder andere de server en de laatste keer dat het document gewijzigd werd.

1.6.3. HTTP/2

Sinds enige tijd wordt ook het nieuwe HTTP/2-protocol gebruikt. Dit biedt de volgende extra mogelijkheden:

1. Er kunnen meerdere verzoeken gebeuren via dezelfde connectie. Men gebruikt hier de techniek van multiplexing. De communicatie tussen *client* en *server* verloopt via binaire *streams* waarlangs meerdere gelijktijdige verzoeken en antwoorden verstuurd worden. De browser hoeft niet langer meerdere gelijktijdige connecties (*sockets)* te openen. Hierdoor kunnen webpagina's veel sneller geladen worden.
2. De *headers* kunnen gecomprimeerd worden zodat enkel nieuwe *headers* doorgestuurd moeten worden. Dit vermindert de hoeveelheid gegevens en verhoogt de snelheid.
3. De *webserver* kan reeds op voorhand gegevens naar de browser doorsturen die hij weldra zal nodig hebben. Deze gegevens worden dan in de *cache* van de browser geplaatst zodat ze beschikbaar zijn zodra ze nodig heeft. Men noemt dit *server push*. Ook dit verhoogt de laadsnelheid van een webpagina.

Momenteel wordt het HTTP/2-protocol door de meest gangbare browsers ondersteund, maar enkel indien de communicatie verloop via een beveiligde verbinding (HTTPS). We zullen later zien hoe we zo een beveiligde verbindingen kunnen opzetten.

Hoofdstuk 2. Java-webapplicaties

2.1. Inleiding

Een *Java*-webapplicatie bestaat uit JSP-pagina's, *servlets* en allerlei bestanden met statische gegevens zoals HTML-pagina's, afbeeldingen enzovoort. Deze bestanden worden ingepakt in een WAR-bestand en afgeleverd aan de webcontainer. In de volgende paragrafen zullen we zo'n webapplicatie stap voor stap opbouwen.

2.2. Webapplicatie-mappenstructuur

Een webapplicatie heeft een bepaalde gestandaardiseerde mappenstructuur die er als volgt uitziet:

Map	***Omschrijving***
/	De *root*. Hier bevinden zich de HTML-pagina's, JSP-pagina's en dergelijke. Eventueel kunnen hier submappen gemaakt worden om de bestanden te groeperen.
/WEB-INF/	De inhoud van deze map is niet rechtstreeks toegankelijk voor de buitenwereld. In deze map bevindt zich onder andere de ***deployment descriptor* (web.xml)** die configuratiegegevens voor de webapplicatie bevat.
/WEB-INF/classes	In deze map worden de klassenbestanden van de Java-klassen geplaatst. Dit kunnen klassen zijn van *servlets, beans* of allerlei hulpklassen. De submappenstructuur komt overeen met de pakketstructuur van de klassen. Deze map wordt toegevoegd aan het *classpath* of *modulepath* van de container.
/WEB-INF/lib	In deze map worden JAR-bestanden geplaatst. JAR-bestanden zijn gecomprimeerde bestanden die onder andere klassenbestanden bevatten. Alle JAR-bestanden in deze map worden toegevoegd aan het *classpath* of *modulepath.*
/WEB-INF/tags	In deze map worden de *tag*-bestanden geplaatst.

Iedere webapplicatie beschikt tevens over een eigen *classloader* die klassen tracht te vinden in de volgende locaties in de opgegeven volgorde:

1. Afzonderlijke klassenbestanden in **WEB-INF/classes**.
2. JAR-bestanden in **WEB-INF/lib**.
3. Het *classpath* of *modulepath* van de webcontainer.

Meerdere webapplicaties kunnen tegelijkertijd draaien in dezelfde webcontainer maar hebben elk hun eigen *classloader*. Dit maakt dat ze elk hun eigen versie van een klassenbestand of JAR-bestand kunnen gebruiken. Gemeenschappelijke JAR-bestanden kunnen eventueel geplaatst worden in het *classpath* of *modulepath* van de webcontainer zodat het niet nodig is die telkens toe te voegen aan het *classpath/modulepath* van de webapplicatie. Bij *TomEE* is dit de map **./lib.**

Bij het gebruik van een IDE zoals *Eclipse* of *IntelliJ* is het mogelijk een webproject op te zetten door middel van een ingebouwde *wizard* of *plugin*. Het is evenwel aangewezen gebruik te maken van *Maven* voor de opzet en configuratie van het een webproject. Dit maakt het project IDE-onafhankelijk en biedt tevens de mogelijkheid het project te bouwen in een commandovenster. Ook het inwerking stellen van de webtoepassing zullen hier met

behulp van *Maven* doen en niet met de ingebouwde mogelijkheden van de IDE.

Opdracht 4: Een webproject maken met Maven

In deze opdrachten maken we een webproject met *Maven.*

- Maak in je IDE een nieuw *Maven*-project. Gebruik hiervoor eventueel een *wizard* van je IDE. Bij het aanmaken van het project kies je voor het *packaging type* ***war*** zodat een webproject wordt aangemaakt.
- Ga na of je de volgende mappen hebt in je project en zo niet maak je de ontbrekende mappen aan:

  src/main/java
  src/main/resources
  src/main/webapp
  src/main/webapp/WEB-INF

- Voeg de volgende configuratie toe aan de POM (of vraag het bestand aan de lesgever).

```
<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>Webcomponents</artifactId>
   <version>1.0</version>
   <packaging>war</packaging>

   <properties>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
      <project.build.sourceEncoding>
         UTF-8
      </project.build.sourceEncoding>

   </properties>

   <build>
      <finalName>Webcomponents</finalName>
      <plugins>
         <plugin>
            <groupId>org.codehaus.cargo</groupId>
            <artifactId>cargo-maven2-plugin</artifactId>
            <version>1.7.4</version>
            <configuration>
               <container>
                  <containerId>tomee8x</containerId>
                  <type>installed</type>
                  <home>${env.TOMCAT_HOME}</home>
               </container>
               <configuration>
                  <type>existing</type>
                  <home>${env.TOMCAT_HOME}</home>
               </configuration>
            </configuration>
```

```
            </plugin>
         </plugins>
      </build>

      <dependencies>
         <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>8.0.1</version>
            <scope>provided</scope>
         </dependency>
      </dependencies>
</project>
```

- Voeg een pagina ***index.html*** toe in de map ***src/main/webapp.***
- Voeg in de map ***WEB-INF*** het bestand ***web.xml*** toe (gebruik hiervoor eventueel een *wizard* van je IDE) met de volgende inhoud:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
   version="4.0">

</web-app>
```

- Voer het volgende *Maven*-commando uit:

  mvn package cargo:deploy

  Hierbij wordt de webapplicatie in *TomEE* in werking gesteld.

- Open de webpagina ***http://localhost:8080/Webcomponents*.**

Het commando ***mvn package cargo:deploy*** maakt eerst een WAR-bestand via de *goal* ***package*** en vervolgens wordt dit WAR-bestand aan *TomEE* afgeleverd via de *goal* ***cargo:deploy***. Om de webapplicatie opnieuw uit werking te halen, kan men de *goal* ***cargo:undeploy*** gebruiken.

Opgelet: aangezien we de webtoepassing in werking stellen via een *Maven plugin* mag het webproject in de IDE niet geconfigureerd worden als module die automatisch in werking gesteld wordt door de IDE. Dit geeft namelijk conflicten.

2.3. Webapplicatie-configuratie

De configuratie van de webapplicatie kan op twee manieren gebeuren:

1. Via **annotaties** die we toevoegen aan de code. In de loop van deze cursus zullen we de verschillende annotaties verkennen.
2. Via een ***deployment descriptor***. Dit is een XML-bestand met de naam *web.xml* dat zich in de map WEB-INF bevindt. Dit bestand heeft een vastgelegde structuur die gedefinieerd is in een XML-schema. De instellingen van dit configuratiebestand kunnen de configuratie via de annotaties vervangen en aanvullen. Een combinatie van beide is dus mogelijk.

De aanwezigheid van de *deployment descriptor* is optioneel. Omdat we tijdens de cursus toch wat extra configuratie gaan toevoegen, hebben we het bestand alvast voorzien.

Een eenvoudig configuratiebestand ziet er als volgt uit:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
   version="4.0">
   <display-name>Webcomponents</display-name>
   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>
</web-app>
```

Met de bovenstaande configuratie geven we aan dat het bestand ***index.html*** het openingsbestand is dat getoond moet worden.

Verder bestaat de mogelijkheid fragmenten van dit configuratiebestand onder te brengen in afzonderlijke JAR-bestanden die zich in de map ***WEB-INF/lib*** bevinden. Dit maakt het mogelijk allerlei *frameworks* makkelijk te integreren door gewoon het JAR-bestand aan de webapplicatie toe te voegen. De configuratie bevindt zich dan in zowel annotaties als het configuratiebestand. Dit valt echter buiten het bestek van deze cursus.

2.4. WAR-bestanden

Een webapplicatie bestaat dus uit een mappenstructuur met componenten en eventueel configuratiebestanden. Deze mappenstructuur moet op de webserver gecreëerd worden. Om de installatie van dergelijke webapplicaties makkelijker te maken, kan men de gehele structuur inpakken in een **WAR**-bestand. WAR is de afkorting van ***W**eb **AR**chive*. In feite is een WAR-bestand hetzelfde als een JAR-bestand; enkel de extensie van het bestand is verschillend: **.war** in plaats van **.jar.** Zo'n WAR-bestand kan dus gewoon gemaakt worden met de JAR-tool of met om het even welk compressieprogramma dat het ZIP-formaat hanteert.

Het WAR-bestand dient de volgende mappen en bestanden uit ons project te bevatten:

```
..\Webcomponents
   +-- \src
        +-- \main
            +-- \webapp
                 +-- \WEB-INF
                      +-- \classes
                      +-- \lib
                      +-- \tags
                      +-- web.xml
                 +-- index.html
```

Merk op dat in ons project de gecompileerde klassen zich in de map ***target/classes*** bevinden. Bij het aanmaken van het WAR-bestand zullen deze door *Maven* op de juiste plaats gezet worden.

Dergelijke WAR-bestanden kunnen eenvoudigweg aan een webserver afgeleverd worden. De webserver zal vervolgens de bestanden uitpakken en uit de annotaties en *deployment*