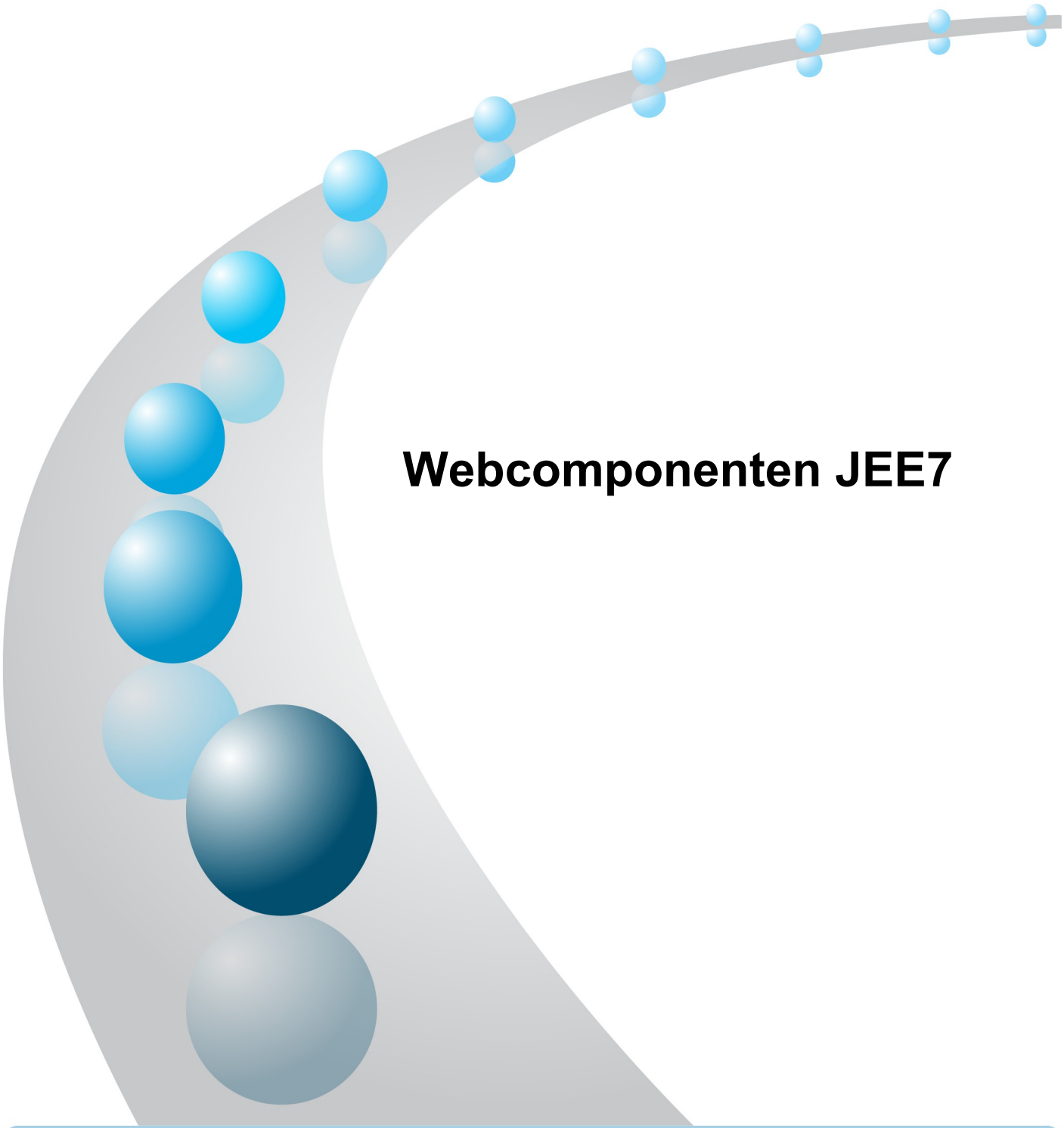
Noël Vaes

Java Trainer & Consultant



Webcomponenten JEE7

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

24/09/2016

Inhoudsopgave

Hoofdstuk 1. Webcontainers

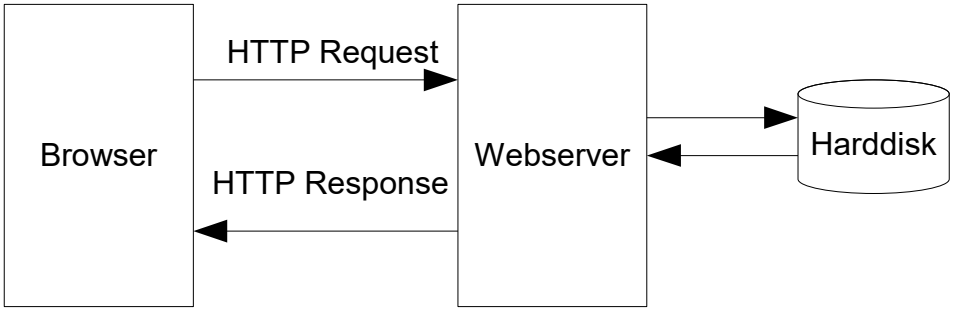
1.1. Inleiding

Het internet heeft de laatste jaren een enorme ontwikkeling gekend en ook de ontwikkeling van meer complexe 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 gebruik maken 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.

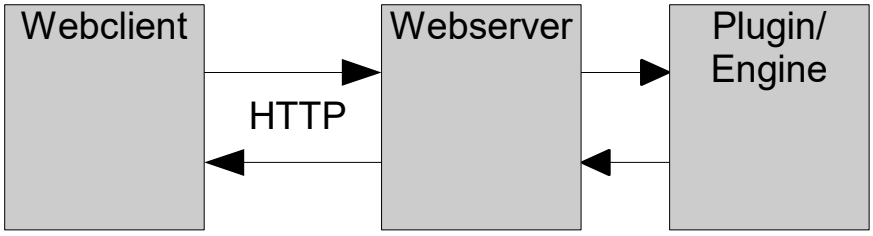


Bij dit mechanisme kan de webserver enkel statische HTML-pagina's afleveren aan de browser. Voor meer geavanceerde toepassingen is dit ontoereikend. Daarom werd dit uitgebreid met de mogelijkheid om HTML-pagina's dynamisch te genereren. De pagina's zijn hierbij niet als dusdanig opgeslagen op de harde 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 immers 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 gebruik maken 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 enz...

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 m.b.v. o.a. *Enterprise JavaBeans, CDI, JavaServer Faces* enz.. Dit valt echter buiten het bestek van deze cursus. We beperken ons tot het ontwikkelen van allerlei webcomponenten met het JEE-platform.

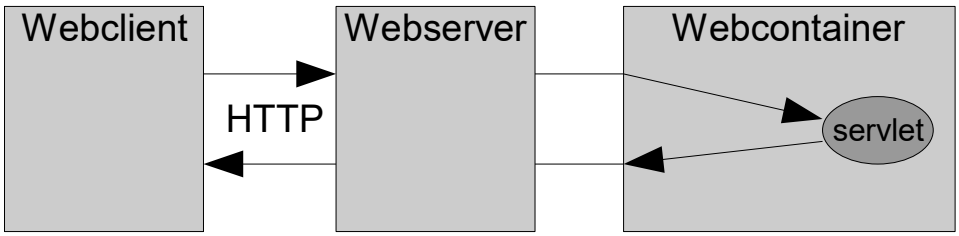
Het JEE-platform kan gedownload worden vanaf de Oracle-website: http://java.oracle.com***.*** Hierin bevinden zich de nodige bibliotheken en ontwikkeltools. Voor het ontwikkelen van webcomponenten is de installatie van het volledige JEE-platform niet echt noodzakelijk. De nodige bibliotheken worden aangeleverd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

In deze cursus maken we gebruik van JSE8 en JEE7. De API-documentatie van JEE7 is te vinden op volgende website: http://docs.oracle.com/javaee/7/api/

1.4. Java Webcontainers

De *plugin* of e*ngine* 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 verzameling van andere componenten. Deze componenten zijn o.a. 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 o.a. 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. Deze laatste wordt gebruikt voor *Enterprise JavaBeans (EJB).*

Voor het ontwikkelen van webcomponenten volstaat echter een webcontainer.

Er zijn uiteraard verschillende implementaties van webcontainers. Doordat ze allen 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).
- ***Tomcat***: populaire *open source*-webcontainer van Apache (tomcat.apache.org).
- ***JBOSS/WildFly****: Open source* enterprise server met ingebouwde webcontainer (www.jboss.org / www.wildfly.org).
- ***GlassFish:*** *Open source* enterprise server gepromoot door Oracle (https://glassfish.java.net/).

1.5. Apache *Tomcat*

We gaan in deze cursus gebruik maken van de populaire *open source*-webserver *Tomcat* van *Apache*. Dit is een webserver volledig in Java geschreven en die uiteraard beschikt over een webcontainer. Deze webserver kan vrij van het internet geplukt worden op volgende website: ***http://tomcat.apache.org***.

1.5.1. Installatie

Tomcat is volledig in Java geschreven en kan bijgevolg werken op elk platform dat Java ondersteunt. Op de site van *Apache* zijn verschillende varianten te vinden waarbij de JDK al dan niet wordt meegeleverd.

In deze cursus maken we gebruik van *Tomcat 8.x*. Deze versie ondersteunt volgende JEE7-specificaties:

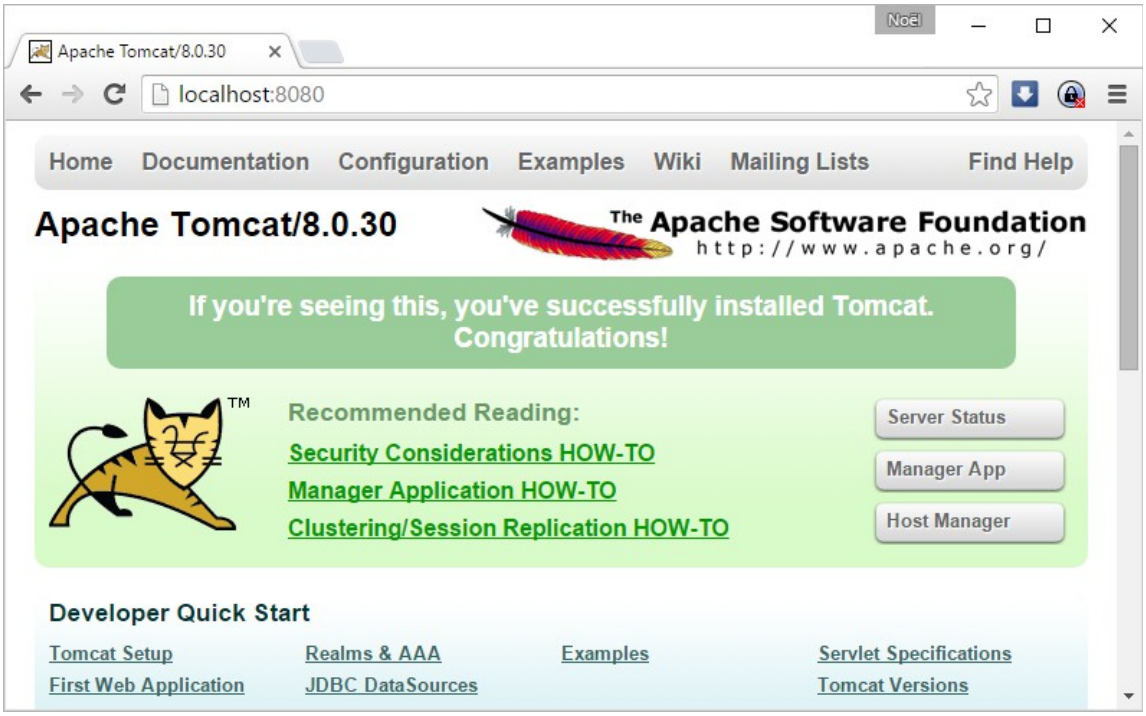
Servlets 3.1
JSP 2.3
Expression Language 3.0

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

Opdracht 1: Tomcat installeren

In deze opdracht gaan we *Tomcat* 8.x op onze computer installeren. We gaan er hierbij vanuit dat de JDK (JSE 8) 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 *Tomcat 8.x* van de *Apache*-website: ***http://tomcat.apache.org***. Kies het gewone zip-bestand dat onafhankelijk is van het besturingssysteem. Dat vind je bij ***Core: zip.***
- 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-tomcat-8.0.x***).
- Start *Tomcat* 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 volgend adres: ***http://localhost:8080.***



- Beëindig *Tomcat* met het commando ***shutdown.bat*** of ***shutdown.sh .*** Je mag ook gewoon CTRL-C gebruiken in het commandovenster waar je Tomcat hebt opgestart.

1.5.2. Integratie in Eclipse

Voor het ontwikkelplatform *Eclipse* zijn er een aantal *plugins* voorhanden voor het beheer van *Tomcat*. 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*. De schermafbeeldingen in deze cursus zijn afkomstig uit de *Luna*-versie van *Eclipse*.

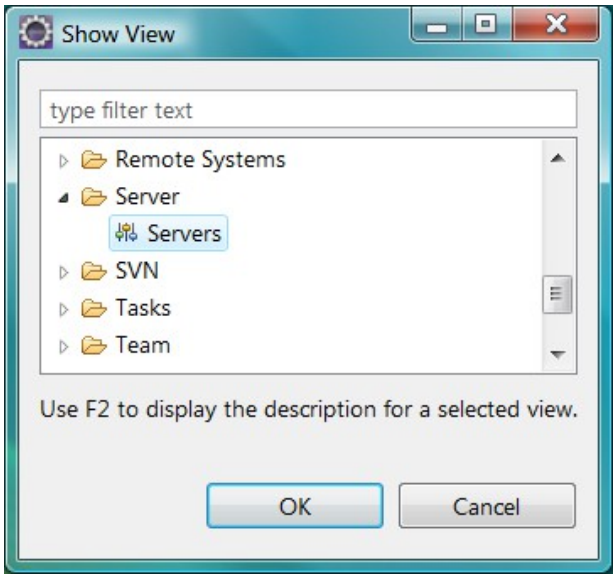
Met deze *plugins* is het mogelijk *Tomcat* te starten en te stoppen vanuit *Eclipse*. Dit heeft o.a. als voordeel dat ook de *logging* van *Tomcat* verschijnt in een venster van *Eclipse,* hetgeen erg handig is bij het *debuggen* van een webapplicatie.

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

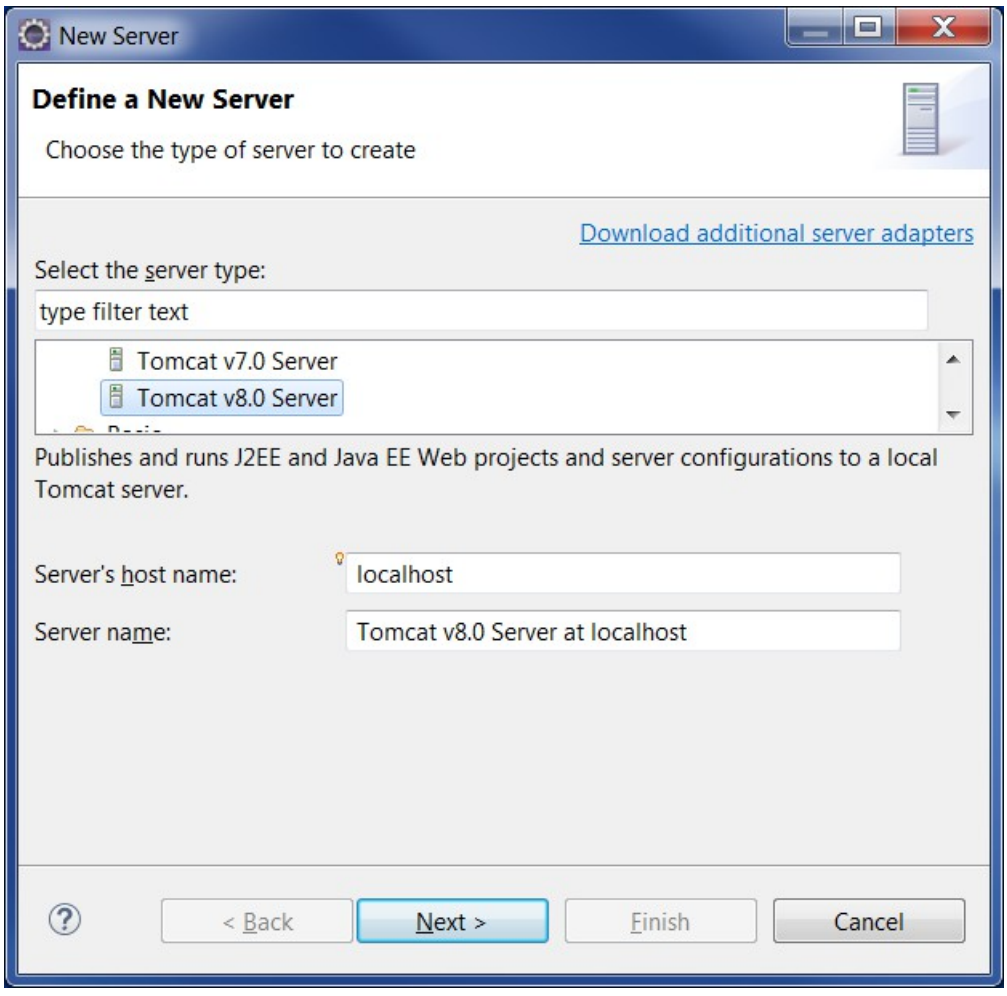
Opdracht 2: Tomcat configureren in Eclipse

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

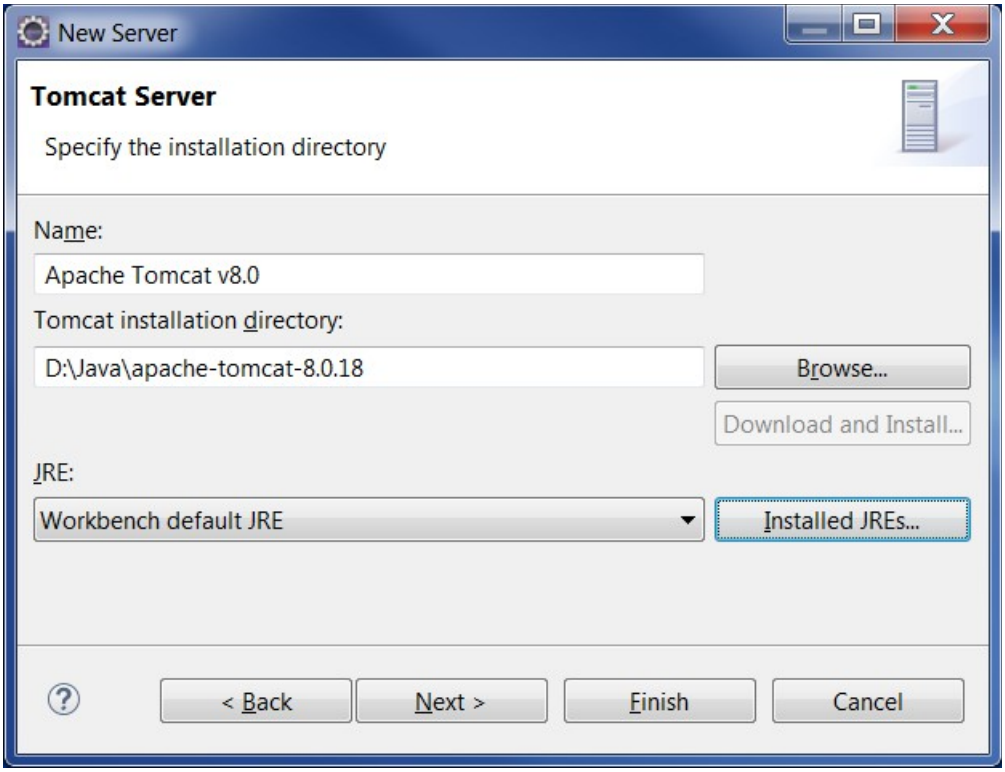
- Start *Eclipse* op.
- Voeg volgende *view* aan de werkomgeving toe: ***Servers/Server.***



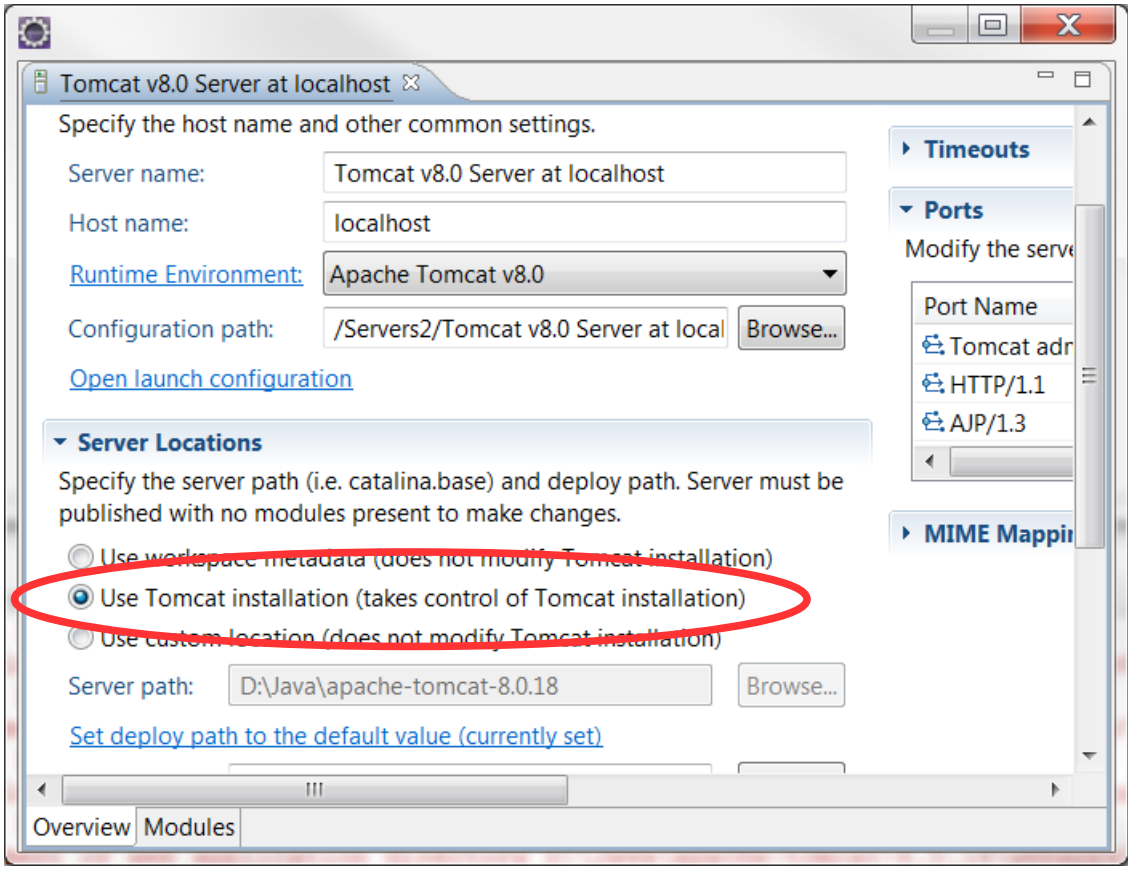
- Selecteer vervolgens in deze *view* ***New->Server***.
- Kies de ***Tomcat v8.0*** server.



- Klik daarna op ***Next.***



- Geef de installatiemap van *Tomcat* 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) .***

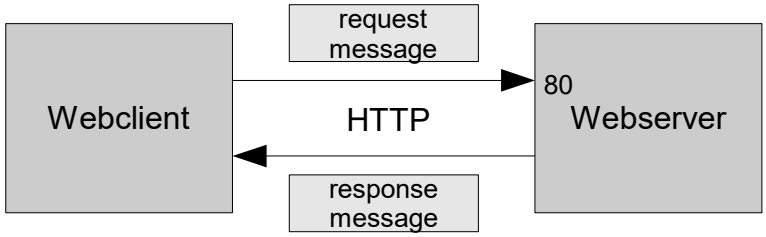
- Start *Tomcat* nu vanuit de *view* door de server te selecteren en vervolgens op de groene pijl te klikken (of via het context-menu).
- Open een browser en surf naar volgend adres: ***http://localhost:8080***. *Tomcat* zou nu correct opgestart moeten zijn.

1.6. Het HTTP-protocol

In de communicatie tussen *webclient* en *webserver* wordt gebruik gemaakt 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 status-informatie 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 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 *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

Dit is extra informate over het verzoek
```

Voor het HTTP 1.1-protocol zijn 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`	Vraagt de inhoud van een bepaalde *resource* op. Eventuele parameters 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`	Wijzigt een bepaalde *resource* op de server.
`DELETE`	Wist een bepaalde *resource* op de server.
`TRACE`	Stuurt gewoon het verzoek terug naar de *client* (echo). Dit wordt gebruikt op 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 gegevens van een *webserver* op te vragen wordt gebruik gemaakt van GET en POST. POST wordt o.a. gebruikt bij het verzenden van formulieren die veel informatie bevatten. De methode HEAD vraagt enkel de *header*-informatie op. Dit wordt o.a. 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. Tenslotte wordt de eerste regel afgesloten met de versie van het HTTP-protocol. Dit is ofwel `HTTP/1.0` of `HTTP/1.1`.

Indien het verzoek een *message body* heeft, dienen tevens 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 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:

Foutcode	***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 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 o.a. de server en de laatste keer dat het document gewijzigd werd.

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 enz... Deze bestanden worden ingepakt in een WAR-bestand en afgeleverd aan de webcontainer. In volgende paragrafen zullen we zo'n webapplicatie stap voor stap opbouwen.

2.2. Webapplicatie-mapstructuur

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

Map	***Omschrijving***
/	De *root*. Hier bevinden zich de HTML-pagina's, JSP-pagina's e.d. 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 o.a. 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 hulp-klassen. De submap-structuur komt overeen met de pakketstructuur van de klassen. Deze map wordt toegevoegd aan het *classpath* van de container.
/WEB-INF/lib	In deze map worden JAR-bestanden geplaatst. JAR-bestanden zijn gecomprimeerde bestanden die o.a. klassenbestanden bevatten. Alle JAR-bestanden in deze map worden toegevoegd aan het *classpath*.
/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* van de webcontainer.

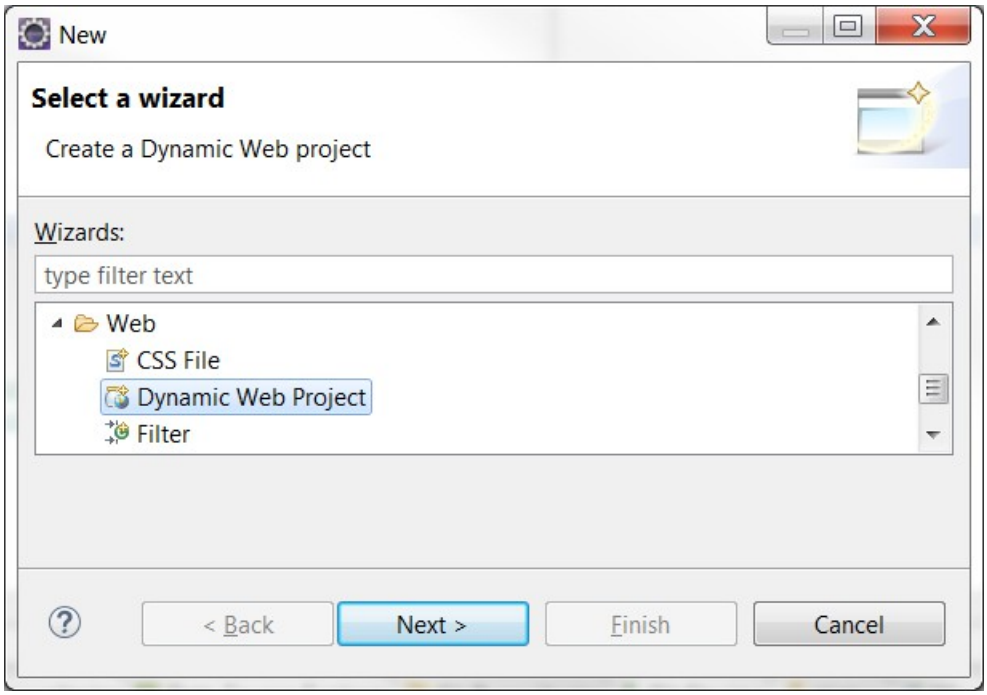
Meerdere webapplicaties kunnen tegelijkertijd draaien in dezelfde webcontainer maar hebben elk hun eigen *classloader*. Dit maakt dat ze elk hun eigen versie van een klassenbestand of JAR-bestand kunnen gebruiken. Gemeenschappelijke JAR-bestanden kunnen eventueel geplaatst worden in het *classpath* van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

We kunnen een webproject in *Eclipse* op twee manieren opzetten: ofwel via het *Web Tools Platform* ofwel m.b.v. *Maven*. We zullen beide illustreren in de volgende twee oefeningen.

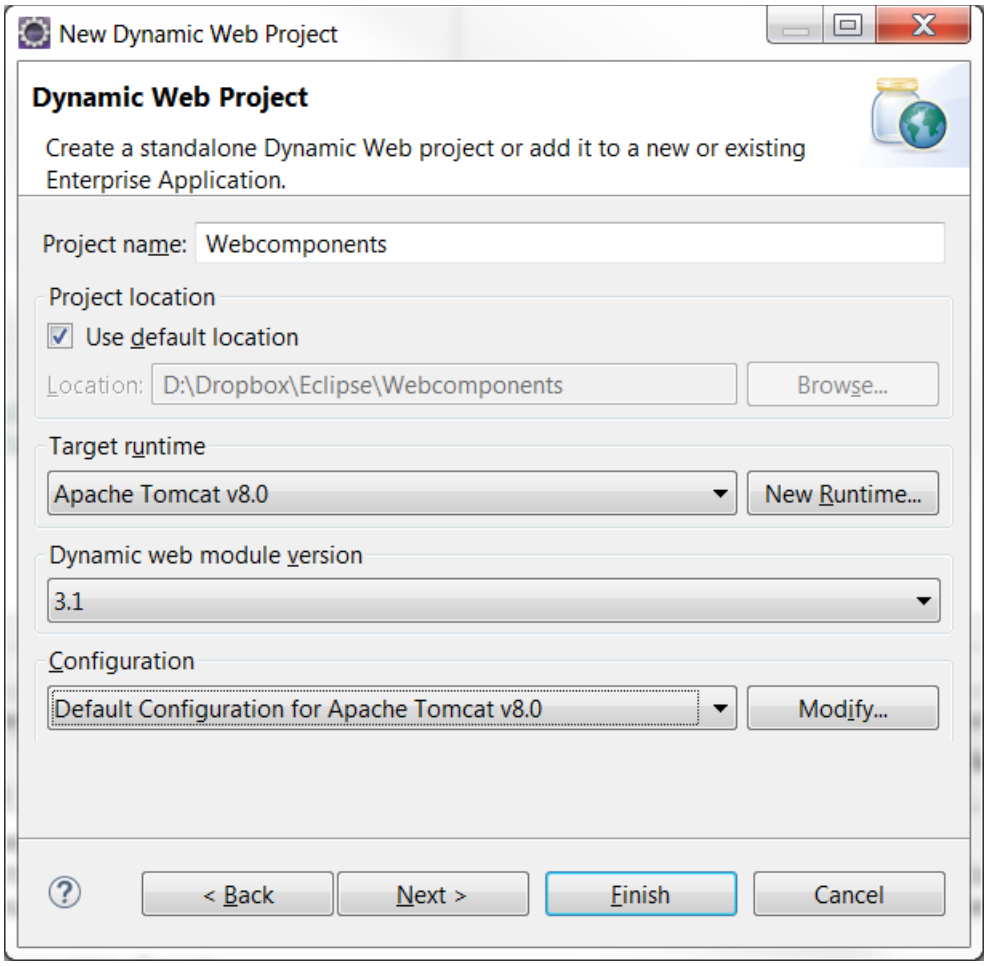
Opdracht 1: Een web-project maken met WTP in Eclipse

In deze opdracht gaan we een webproject maken met behulp het *Web Tools Platform* van *Eclipse.*

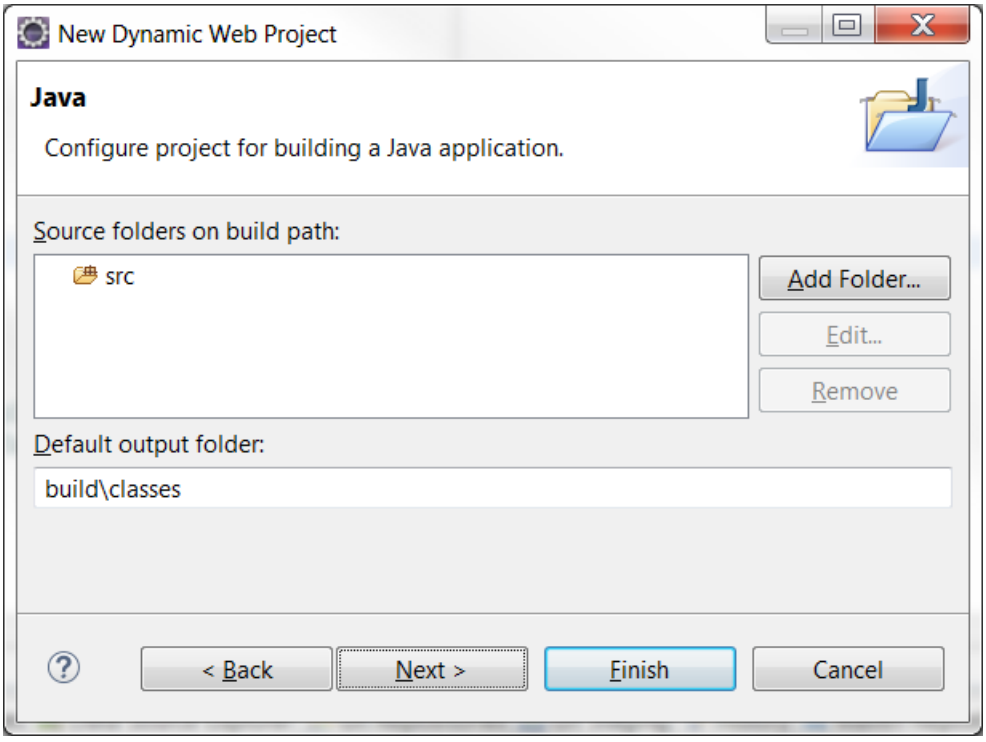
- Maak een nieuw project in *Eclipse* via ***File->New->Other*** en kies hierbij ***Dynamic Web Project***.



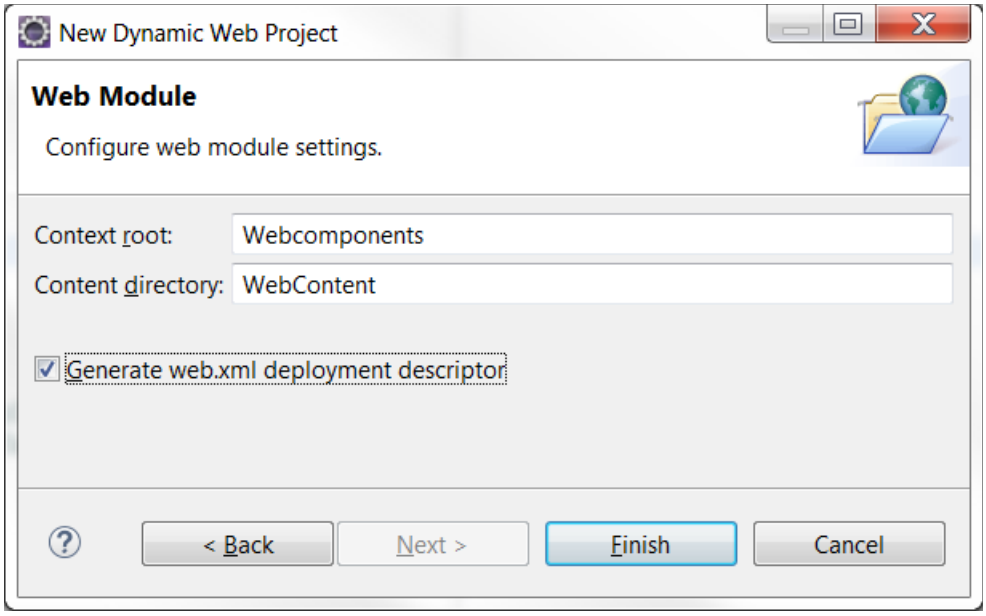
- Klik op ***Next.***



- Geef het project de naam ***Webcomponents*** en selecteer ***Apache Tomcat v7.0*** als *Target runtime*.
- Klik op ***Next.***



- Klik op ***Next.***



- Vink de optie ***Generate web.xml deployment descriptor*** aan.
- Klik vervolgens op ***Finish*** om het nieuwe project te laten genereren.
- Bekijk vervolgens de mapstructuur en bestanden die gegenereerd werden via de *Navigator view.*

  De map *WebContent* bevat de nodige bestanden voor de webapplicatie. De inhoud van deze map zal door *Eclipse* later afgeleverd worden aan *Tomcat*.

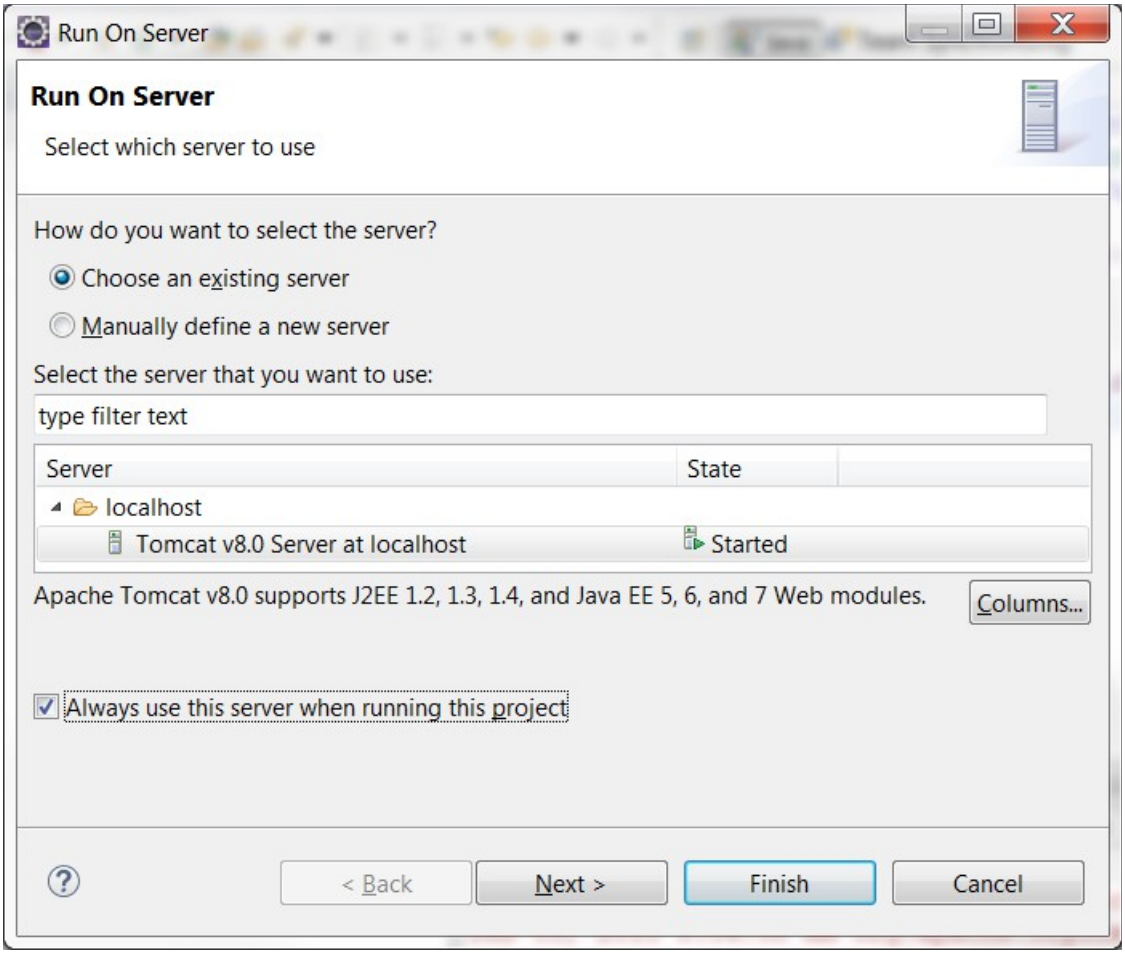
- Open de eigenschappen van het gegenereerde project en bekijk het onderdeel ***Java Build Path.*** Merk de toegevoegde JAR-bestanden op in het tabblad ***Libraries***.

  Deze JAR-bestanden maken deel uit van *Tomcat* maar moeten opgenomen worden in het project om het geheel te kunnen compileren.

- Voeg een nieuwe HTML-pagina toe via ***File->New*** en geef deze de naam ***index.html***. Voeg wat inhoud toe aan deze pagina.

- Voer het project uit (via het menu, de knoppenbalk of gewoon CTRL-F11).

- Selecteer *Tomcat v8.0* als server waar het project op uitgevoerd moet worden. Klik vervolgens op ***Finish***.



Opdracht 2: Een web-project maken met Maven in Eclipse

In deze opdrachten maken we een web-project met de *Maven plugin* in *Eclipse.* We maken hierbij een eenvoudig *Maven*-project dat we verder manueel uitbreiden met een aantal *Maven plugins*.

- Selecteer ***File->New->Other*** en navigeer naar het onderdeel ***Maven.***