

Laboratorul 1

Dezvoltarea unei aplicații simple pentru întreprindere (JEE)

Maniera de evaluare la această disciplină

- Participarea la orele de curs și de laborator: este recomandată la curs și obligatorie la laborator:
- Neparticiparea la mai mult de 50% din laboratoare conduce la refacerea disciplinei.
- Neparticiparea la curs conduce la probleme la examenul teoretic
- Nefacerea temelor la timp (adică săptămânal) conduce în 95% din cazuri la imposibilitatea de a realiza în timp util a subiectelor practice la examen (procent obținut pe analiza notelor din ultimii trei ani)

Evaluarea de laborator: pentru a putea lua 10 la laborator trebuie ca studentii să fie capabili la întrebările asistenților (minim 2-3 de student) cu privire la conținutul cursului curent (și pentru care a fost creat laboratorul) - 30% -

Examen final 70% (este o singura notă) defalcată astfel:

Proba de laborator – 40 % cu bilete (trase din teanc) și două ore maxim la dispoziție. Un subiect din două trebuie să fie îndeplinit integral pentru a se putea nota (min 5).

Proba teoretică – 40% - test docimologic - conține și întrebări cu caracter practic specifice laboratorului (min 5)

Teme acasă: 20% (atenție ca efect al acestui procent se poate întâmpla ca 5 la subiectul practic și 5 la teorie ceea ce face $5 \cdot 0.8 = 4!!!$ deci să picați din motive de teme nepredate

În caz de variantă on line (SARS-CoV-II)

la testul practic - se alege automat două probleme din pool și se trimit

la testul teoretic - oral cu întrebări selectate automat din pool de către aplicație

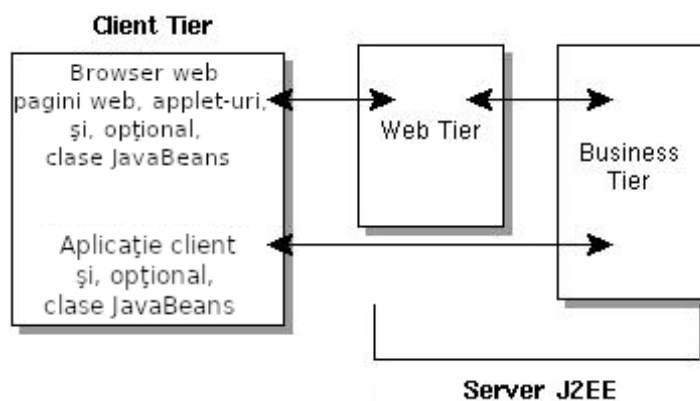
Arhitectura JEE

După cum am discutat la curs platforma **JEE (Java Enterprise Edition)** este proiectată pentru a-i ajuta pe dezvoltatori să creeze aplicații specifice întreprinderilor sau corporațiilor, multinivel - multistrat, scalabile, fiabile și sigure.

1. Exemplu simplu de implementare în stilul JEE

Nivelul client constă într-o aplicație client care efectuează cereri către nivelul de afaceri care, în acest caz este alcătuit din două straturi cel pentru web și cel care conține logica de afaceri. Acest nivel tratează cererile venite de la clienți și procesează datele aplicației, pe care apoi le trimite către nivelul de persistență (detaliile au fost discutate la curs).

Datorită complexității mari a soluției Oracle pentru acest laborator s-a ales, utilizând GlassFish ca server de aplicații *enterprise* deoarece acesta este o variantă *open-source* de implementare a standardului JEE.

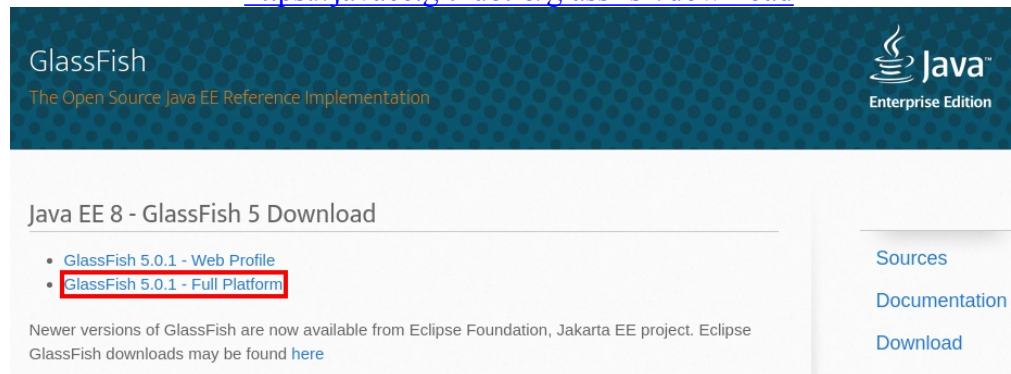


Atenție! Cei care lucrează de pe stațiile din laborator, săriți direct la Secțiunea 1.2.

1.1. Instalare server GlassFish

Server-ul GlassFish este disponibil pentru descărcare aici:

<https://javaee.github.io/glassfish/download>



Descărcați ultima versiune de **GlassFish Full Platform** (versiunea **5.0.1** la momentul scrierii acestui laborator) de la URL-ul furnizat și dezarhivați conținutul. Pentru exemplul detaliat în continuare, vom considera server-ul dezarhivat în folder-ul `/home/student/opt/glassfish5`.

Atenție! GlassFish nu funcționează cu versiunile de Java ≥ 9 , așadar trebuie să folosiți o versiune de **Java SDK cel mult egală cu 8**. Verificați versiunea de Java cu următoarea comandă:

```
java -version
```

În cazul în care sunt disponibile mai multe versiuni de Java pe stația de lucru (cu sistemul de operare Debian), iar acestea sunt înregistrate ca alternative, se poate schimba versiunea de Java la cerere, folosind comanda:

```
sudo update-alternatives --config java
```

Din lista de alternative, se alege indicele versiunii 8 (dacă este disponibilă! Dacă nu, trebuie instalată și înregistrată ca alternativă sau ca `JAVA_HOME` implicit în sistem).

Notă: Se recomandă instalarea ultimelor versiuni de JDK disponibile

1.1.1. Instalare JDK 11

Accesați următorul link:

<https://www.oracle.com/java/technologies/downloads/#java11>

Și descărcați `jdk-11.0.12_linux-x64_bin.tar.gz` (ultima versiune la scrierea laboratorului)

Evident că, pentru o versiune mai veche de JDK, vă va cere un cont la ei (fie aveți, fie vi-l faceți). Apoi:

```
cd /home/$USER/Downloads
tar xfvz jdk-11.0.12_linux-x64_bin.tar.gz
```

Se va despacheta în: `/home/nume_user/Downloads/jdk-11.0.12/`

Această cale va fi utilizată în cadrul proiectelor IntelliJ din laborator pentru a indica JDK 11 atunci când acesta trebuie selecționat și nu există pe sistemul dumneavoastră deja instalat în

calea implicită, adică `/usr/lib/jvm/.....`

Sau, puteți să o despachetați acolo și eventual, dacă doriți să o mai folosiți și în alte medii:

```
sudo cp -r ~/Downloads/jdk-11.0.12 /usr/lib/jvm/.  
  
sudo update-alternatives --install /usr/bin/java java  
/usr/lib/jvm/jdk-11.0.12 1102
```

1.1.2. Instalare JDK 1.8

Se urmează aceiași pași ca și în cazul instalării JDK 11.

Accesați următorul link:

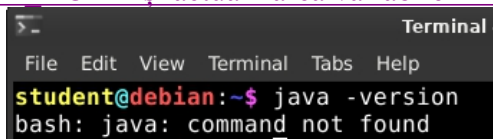
<https://www.oracle.com/java/technologies/downloads/#java8>

Setare JAVA_HOME

Se rulează comanda:

```
java -version
```

În cazul în care comanda nu este disponibilă, este necesară adăugarea acesteia prin setarea variabilei JAVA_HOME și actualizarea variabilei PATH.

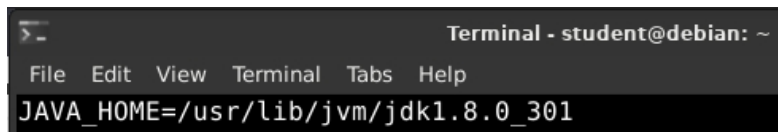


```
Terminal -  
File Edit View Terminal Tabs Help  
student@debian:~$ java -version  
bash: java: command not found
```

Pentru aceasta, se editează fișierul de environment pentru setarea variabilei globale JAVA_HOME, și fișierul de bashrc, pentru actualizarea variabilei PATH folosindu-ne de variabila JAVA_HOME (pentru JAVA_HOME s-a ales bineînțeles jdk1.8 instalat anterior).

1. Deschidem și edităm etc/environment

```
sudo vim /etc/environment
```

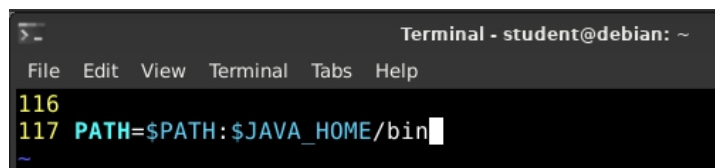


```
Terminal - student@debian: ~  
File Edit View Terminal Tabs Help  
JAVA_HOME=/usr/lib/jvm/jdk1.8.0_301
```

```
# INSIDE /etc/environment #  
JAVA_HOME=/usr/lib/jvm/jdk1.8.0_301
```

2. Deschidem și edităm ~/.bashrc

```
sudo vim ~/.bashrc
```



```
Terminal - student@debian: ~  
File Edit View Terminal Tabs Help  
116  
117 PATH=$PATH:$JAVA_HOME/bin
```

```
# INSIDE ~/.bashrc #  
PATH=$PATH:$JAVA_HOME/bin
```

Pentru actualizarea variabilelor respective în sesiunea curentă de terminal:

```

./etc/environment
~/bashrc

```

```

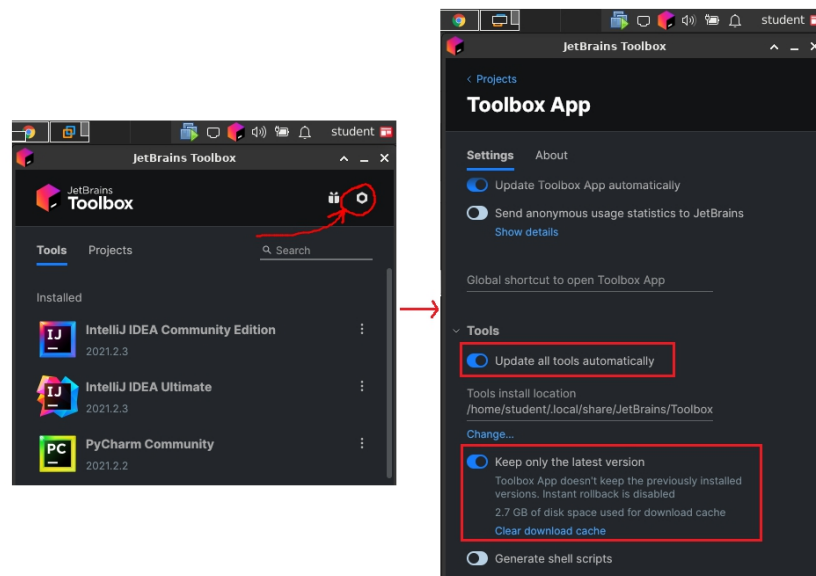
Terminal - student@debian: ~
File Edit View Terminal Tabs Help
student@debian:~$ ./etc/environment
student@debian:~$ java -version
java version "1.8.0_301"
Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)
student@debian:~$

```

1.2. Creare și configurare proiect JEE minimal (Web Application) utilizând IntelliJ IDEA Community

În cadrul orelor de laborator se vor folosi IDE-urile de la JetBrains și anume IntelliJ și PyCharm. Se recomandă folosirea utilitarului JetBrains Toolbox pentru gestiunea produselor JetBrains, care poate fi utilizat pentru actualizarea automată a acestora.

De asemenea, dacă se dorește utilizarea IDE-urilor respective în versiunea Professional, prin intermediul licenței de student de care beneficiați, utilitarul ToolBox facilitează acest lucru, nefiind nevoie introducerea acesteia în fiecare din IDE-urile utilizate.



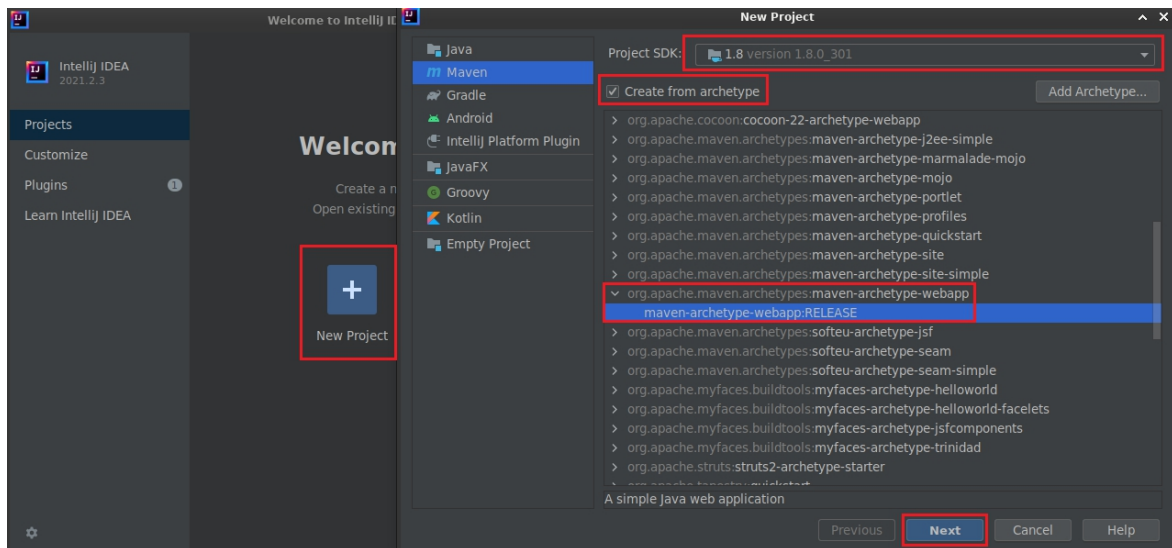
Mediul de dezvoltare IntelliJ IDEA Community nu dispune de posibilitatea de a crea în mod facil tipurile de proiecte Java Enterprise, așa încât se vor urma pașii prezentați în continuare pentru a obține un proiect JEE **Web Application**.

1.2.1. Creare proiect IntelliJ

Deschideți IntelliJ IDEA Community, iar în meniul din partea dreaptă alegeți „**Create new project**”.

În fereastra de selecție a tipului de proiect, alegeți „**Maven**” în partea stângă, apoi selectați versiunea de **Java SDK 1.8**. Bifați „**Create from archetype**”, iar din lista de arhetipuri disponibile, expandați **org.apache.maven.archetypes:maven-archetype-webapp** și selectați **maven-archetype-webapp:RELEASE**. Apăsați pe „**Next**”.

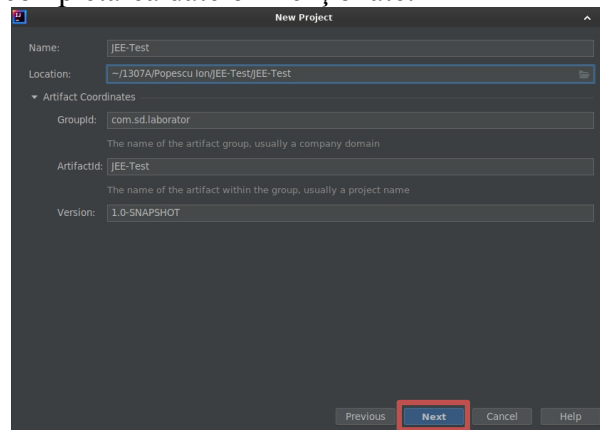
Dacă nu aveți Java 8 afișat în listă, apăsați pe butonul „**New**” din partea dreaptă a listei cu JDK-uri disponibile și selectați folder-ul unde este disponibilă versiunea 8 de Java SDK (de exemplu, **/usr/lib/jvm/oracle-java8-jdk-amd64**).



În continuare, se alege numele și locația proiectului pe disc, precum și detaliile artefactului rezultat. Momentan, secțiunea „**Artifact Coordinates**” poate fi lăsată cu valorile implicite.

În acest exemplu, proiectul se va numi „**JEE-Test**”, iar locația va fi `~/1307A/Popescu Ion/JEE-Test`.

Apăsați pe „**Next**” după completarea datelor menționate.



În următoarea fereastră, se lasă totul neschimbat și se apasă „**Finish**”.

1.2.2. Configurare proiect Maven

Deschideți fișierul `pom.xml` (**Project Object Model**) și adăugați o primă dependență a proiectului: **JavaEE API** (<https://mvnrepository.com/artifact/javax/javaee-api/8.0>). Ca subordonat al `tag`-ului `<dependencies>`, adăugați următorul element:

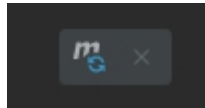
```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>8.0.1</version>
  <scope>provided</scope>
</dependency>
```

```

22 <dependencies>
23 <dependency>
24 <groupId>junit</groupId>
25 <artifactId>junit</artifactId>
26 <version>4.11</version>
27 <scope>test</scope>
28 </dependency>
29
30 <dependency>
31 <groupId>javax</groupId>
32 <artifactId>javaee-api</artifactId>
33 <version>8.0.1</version>
34 <scope>provided</scope>
35 </dependency>
36 </dependencies>
37
38 <build>
39 <finalName>JEE-Test</finalName>
project > dependencies > dependency

```

Pentru ca Maven să actualizeze proiectul conform cu schimbările făcute în fișierul `pom.xml`, atunci când IntelliJ afișează în partea din dreapta-sus un mesaj de avertizare în acest sens, alegeți iconița cu maven.



Pentru a facilita execuția server-ului de aplicații *enterprise* și a încărca (a face *deploy*) automat la artefactele rezultate în urma compilării surselor, trebuie să adăugați plugin-ul **Cargo** pentru Maven, și să îl marcați ca dependență a proiectului. Așadar, adăugați următorul element ca subordonat al *tag*-ului `<plugins>`:

```

<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <version>1.8.5</version>
  <configuration>
    <container>
      <containerId>glassfish5x</containerId>
      <type>installed</type>
      <!-- Path to directory where glassfish is installed -->
      <home>/home/student/opt/glassfish5</home>
    </container>
    <configuration>
      <type>existing</type>
      <!-- Path to domains directory -->
      <home>/home/student/opt/glassfish5/glassfish/domains</home>
      <properties>
        <!-- Domain name where application will be deployed. -->
<cargo.glassfish.domain.name>domain1</cargo.glassfish.domain.name>
        <!-- Glassfish user to authenticate -->
        <cargo.remote.username>admin</cargo.remote.username>
        <!-- Glassfish password to authenticate -->
        <cargo.remote.password></cargo.remote.password>
      </properties>
    </configuration>
  </configuration>
</plugin>

```

Mai sus s-a evidențiat cu text îngroșat calea către server-ul GlassFish descărcat în pașii anteriori. **Înlocuiți această cale în mod corespunzător locației folder-ului `glassfish5` pe discul**

dvs.

```

73 <plugin>
74 <artifactId>maven-deploy-plugin</artifactId>
75 <version>2.8.2</version>
76 </plugin>
77 <plugin>
78 <groupId>org.codehaus.cargo</groupId>
79 <artifactId>cargo-maven2-plugin</artifactId>
80 <version>1.8.5</version>
81 <configuration>
82 <container>
83 <containerId>glassfish5x</containerId>
84 <type>installed</type>
85 <!-- Path to directory where glassfish is installed -->
86 <home>/home/student/opt/glassfish5</home>
87 </container>
88 <configuration>
89 <type>existing</type>
90 <!-- Path to domains directory -->
91 <home>/home/student/opt/glassfish5/glassfish/domains</home>
92 <properties>
93 <!-- Domain name where application will be deployed. -->
94 <cargo.glassfish.domain.name>domain1</cargo.glassfish.domain.name>
95 <!-- Glassfish user to authenticate -->
96 <cargo.remote.username>admin</cargo.remote.username>
97 <!-- Glassfish password to authenticate -->
98 <cargo.remote.password></cargo.remote.password>
99 </properties>
100 </configuration>
101 </configuration>
102 </plugin>
103 </plugins>
104 </pluginManagement>

```

Credențialele de autentificare la consola de administrare GlassFish au fost păstrate la valorile implicite (nume de utilizator: **admin**, parola: **<go1>**), cu scop demonstrativ. **Evident că nu se acceptă utilizarea unei parole vide în cazul real dintr-o firmă.**

După adăugarea *plugin*-ului **Cargo**, marcați-l ca dependență a proiectului, adăugând următorul element ca subordonat al tag-ului **<dependencies>**:

```

<dependency>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <version>1.8.5</version>
</dependency>

```

```

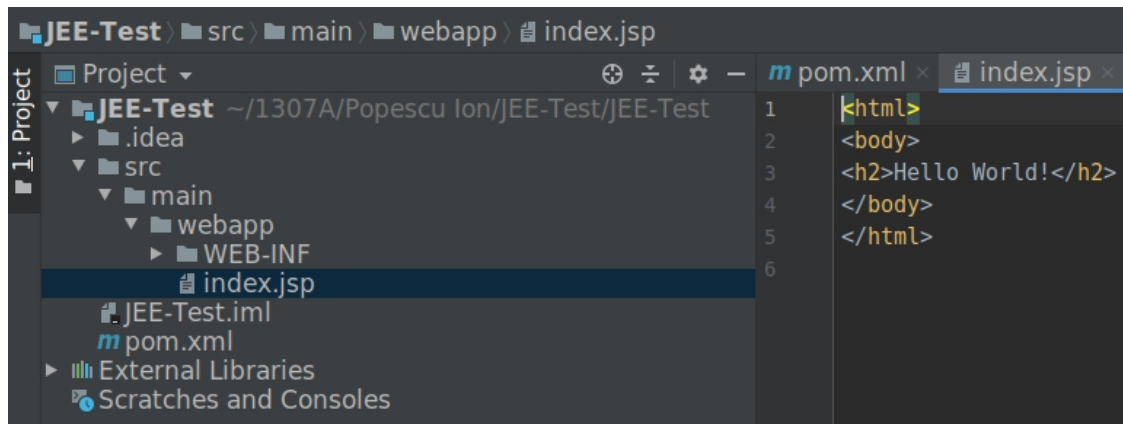
30 <dependency>
31 <groupId>javax</groupId>
32 <artifactId>javaee-api</artifactId>
33 <version>8.0.1</version>
34 <scope>provided</scope>
35 </dependency>
36
37 <dependency>
38 <groupId>org.codehaus.cargo</groupId>
39 <artifactId>cargo-maven2-plugin</artifactId>
40 <version>1.8.5</version>
41 </dependency>
42 </dependencies>

```

Nu uitați să apăsați iconița pentru import când IntelliJ detectează o modificare a fișierului `pom.xml` specific Maven. Faceți acest lucru de fiecare dată când îl modificați (adăugați / ștergeți o dependență / un *plugin* sau modificați alte configurări).

1.2.3. Definirea organizării fișierelor sursă

Proiectul creat are structura din figura următoare:



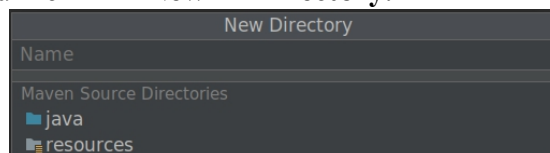
Folder-ul `src/main/webapp` este rădăcina componentei web a aplicației JEE de tip **Web Application**, ceea ce înseamnă că tot conținutul acesteia va fi împachetat, la compilare, într-un fișier **WAR (Web ARchive)**, ce va reprezenta artefactul încărcat (*deployed*) pe server-ul de aplicații *enterprise*.

Fișierul `index.jsp` este o pagină de tip **Java Server Page**, cu un conținut de test generat implicit.

Folder-ul **WEB-INF** conține fișiere / directoare inaccesibile public. Ele pot fi accesate doar de **servleți** sau alte pagini JSP, dar în niciun caz în mod direct. De obicei, aici se pun fișiere XML ce conțin descriptori care definesc comportamentul aplicației (de exemplu, în fișierul **WEB-INF/web.xml** se mapează servleții pe rutele de acces dorite - urmează în continuare).

Din structura proiectului lipsesc două directoare, pe care le veți crea astfel:

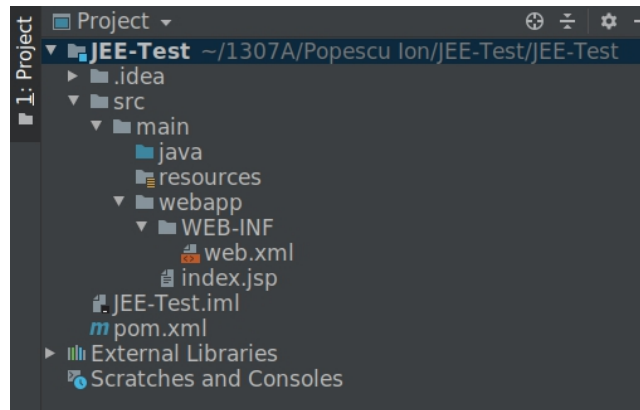
Apăsați dreapta pe folder-ul `main` → **New** → **Directory**.



Selectați, pe rând, cele două directoare sugerate de IntelliJ, conform alegerii proiectul de tip Maven (dublu apăsați pe „**java**”, apoi repetați operațiunea de creare a unui folder nou și selectați apoi „**resources**”).

Notă: În cazul în care nu aveți sugerate directoarele respective, va fi nevoie să le creați manual și să marcați **java** ca fiind **Sources Root**, iar **resources** ca fiind **Resources Root**. Pentru aceasta apăsați click dreapta pe fiecare din aceste foldere și apoi **Mark Directory As**.

Folderul **java** va conține fișiere sursă Java (de ex. servleți), iar folderul **resources** va conține fișiere de tip resursă (de ex. fișiere de configurare XML, imagini, font-uri etc.). Așadar, structura finală de proiect arată astfel:



1.3. Adăugare servlet simplu

Un *servlet* este, în mod simplist spus, o componentă web care primește cereri și generează răspunsuri bazate pe cererile primite. Ca și tehnologie JEE, este plasat în stratul de *web*.

Adăugați un servlet simplu care va răspunde la o cerere HTTP de tip GET cu un mesaj „Hello from servlet”:

Apăsați dreapta pe folder-ul cu surse Java (**java**) → **New** → **Java Class**. Denumiți clasa ca „**HelloServlet**”. Introduceți următorul conținut:

```
public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.getWriter().print("Hello from servlet");
    }
}
```

Servleții trebuie să se conformeze cu Java Servlet API, deci clasa creată extinde clasa de bază **HttpServlet**. Un servlet conține metode de tratare a fiecărui tip de cerere HTTP: pentru cerere GET → metoda **doGet()**, pentru cerere POST → metoda **doPost()**, ș.a.m.d. Fiecare din aceste metode au la dispoziție ca parametri o variabilă care încapsulează cererea efectuată de client (**HttpServletRequest**), respectiv una care încapsulează răspunsul ce va fi generat și trimis în metoda de tratare respectivă (**HttpServletResponse**).

În cazul în care nu suprascrieți o anumită metodă de tratare a unui tip de cerere, aceasta va conține o implementare implicită, disponibilă în clasa de bază **HttpServlet**, mai exact un mesaj de eroare de tipul „**HTTP method TIP_METODĂ is not supported by this URL**”. Nu înseamnă că metoda nu există / nu este implementată!

Servletul creat mai sus pur și simplu scrie mesajul „Hello from servlet” în răspunsul trimis către client.

1.4. Maparea servleților

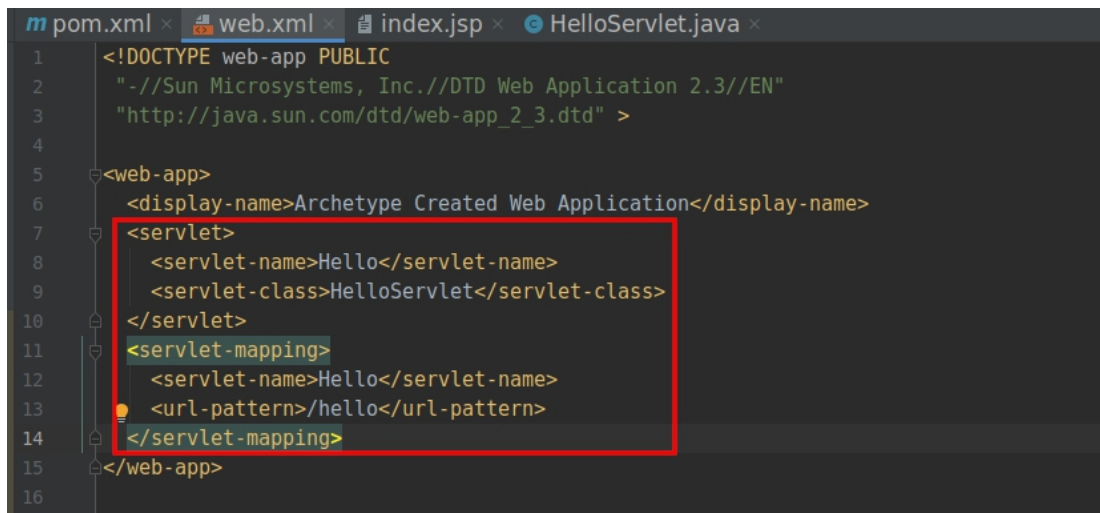
Servleții sunt accesați pe baza rutelor la care aceștia sunt mapați în server-ul enterprise, dar pentru **HelloServlet** nu ați specificat încăieri nici o cale de mapare, și deci nu l-ați putea accesa, dacă ar fi să testați aplicația așa cum este acum.

Deschideți fișierul **webapp/WEB-INF/web.xml** și adăugați următorul conținut, **ca un subordonat al elementului <web-app>**:

```

<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Hello</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>

```



```

1  <!DOCTYPE web-app PUBLIC
2  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3  "http://java.sun.com/dtd/web-app_2_3.dtd" >
4
5  <web-app>
6    <display-name>Archetype Created Web Application</display-name>
7
8    <servlet>
9      <servlet-name>Hello</servlet-name>
10     <servlet-class>HelloServlet</servlet-class>
11   </servlet>
12   <servlet-mapping>
13     <servlet-name>Hello</servlet-name>
14     <url-pattern>/hello</url-pattern>
15   </servlet-mapping>
16 </web-app>

```

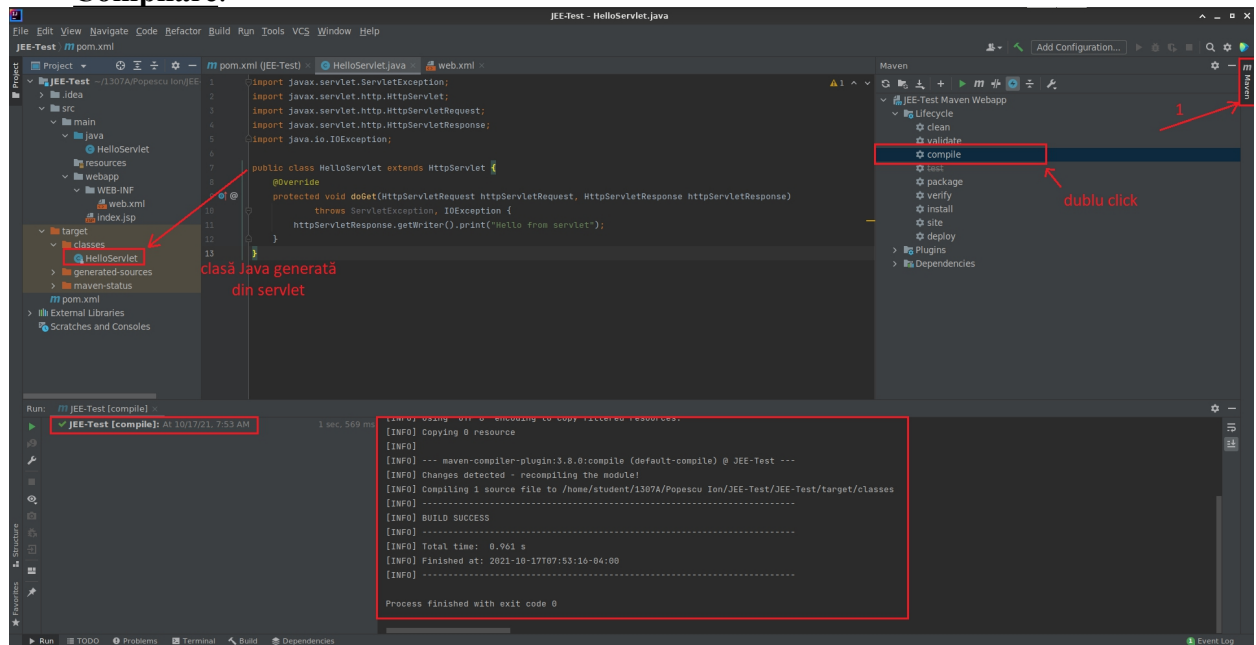
Cu ajutorul descriptorilor XML, pentru fiecare servlet în parte, descris prin numele clasei din care face parte (prefixat de numele pachetului, dacă este cazul), se poate specifica pe ce cale să fie mapat. Serverul *enterprise* folosește acest XML la încărcarea aplicației pentru a expune servleții corespunzător.

Calea prin care se poate accesa servlet-ul **HelloServlet** este, în acest caz, **/hello**. **Atenție, calea este relativă la rădăcina proiectului!** Adică, dacă pagina principală care face trimitere la rădăcina aplicației JEE este **/my-app**, atunci acest servlet se accesează folosind calea **/my-app/hello**, și nu simplu **/hello**!

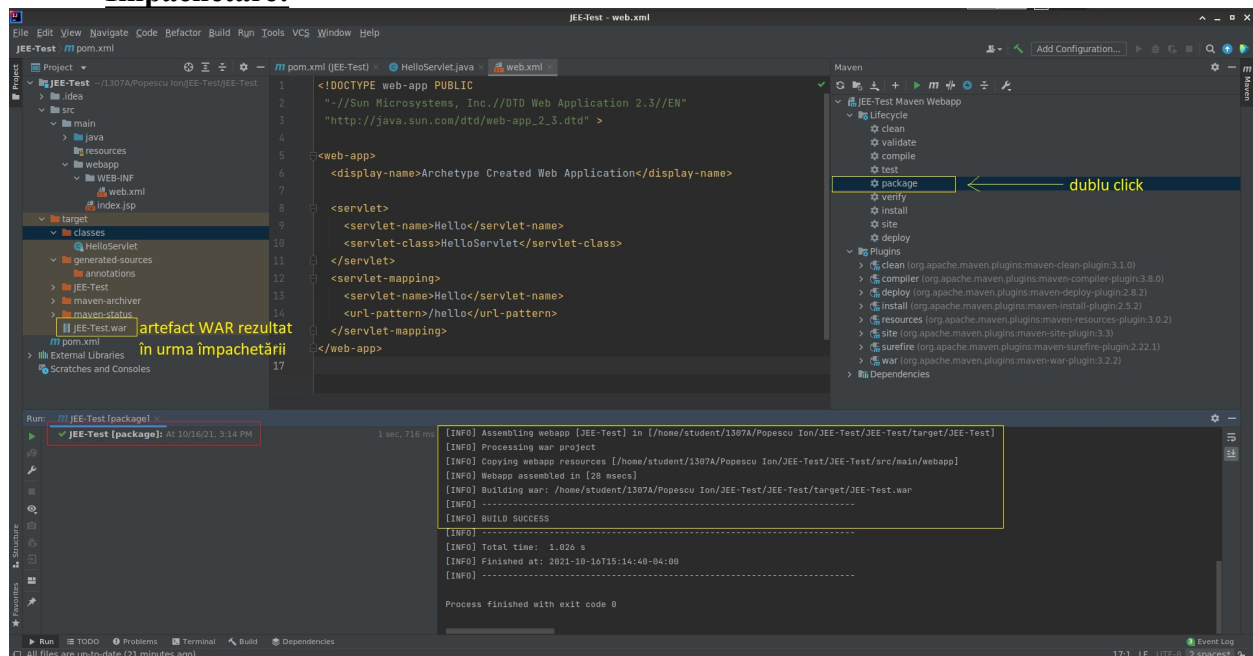
1.5. Compilare și împachetare aplicație JEE

Aplicația se compilează utilizând *lifecycle*-ul standard Maven numit „**compile**”, accesibil din panoul Maven din partea dreaptă a ferestrei IntelliJ. De asemenea, împachetarea aplicației într-un fișier **WAR**, pentru a o pregăti de încărcare pe server se face tot din același panou, folosind *lifecycle*-ul „**package**”.

Compilare:



Împachetare:

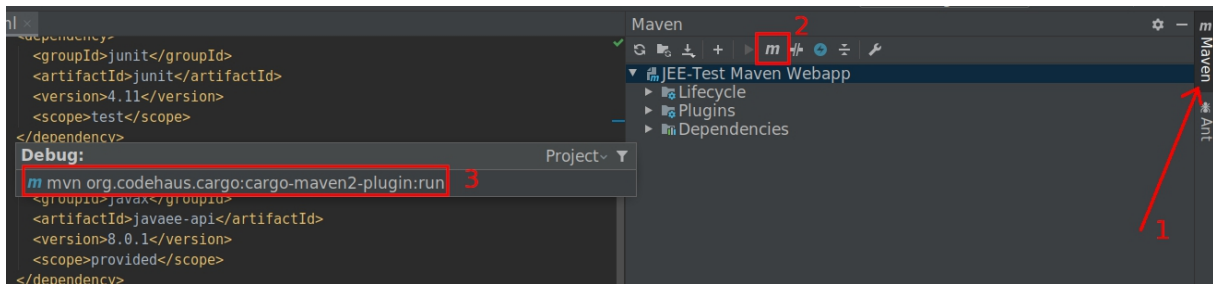


1.6. Încărcare artefact WAR și testare aplicație

După ce artefactul WAR este disponibil, trebuie încărcat (*deployed*) pe server-ul de aplicații enterprise GlassFish (în acest caz). Acest lucru este făcut automat de *plugin*-ul **Cargo**, utilizând țelurile (*goal*-urile) Maven puse la dispoziție de acesta. Lista completă poate fi consultată în documentația Cargo: <https://codehaus-cargo.github.io/cargo/Maven+3+Plugin.html>

Goal-urile acestui *plugin* nu apar în lista de *Maven goals* din meniul standard IntelliJ (accesibilă din tab-ul **Maven** → secțiunea **Plugins**), așa încât vor trebui executate manual, astfel:

- Expandati tab-ul Maven din partea dreaptă a ferestrei IntelliJ, apoi apăsați pe pictograma în formă de „m” înclinat din partea de sus a panoului (**Execute Maven Goal**).

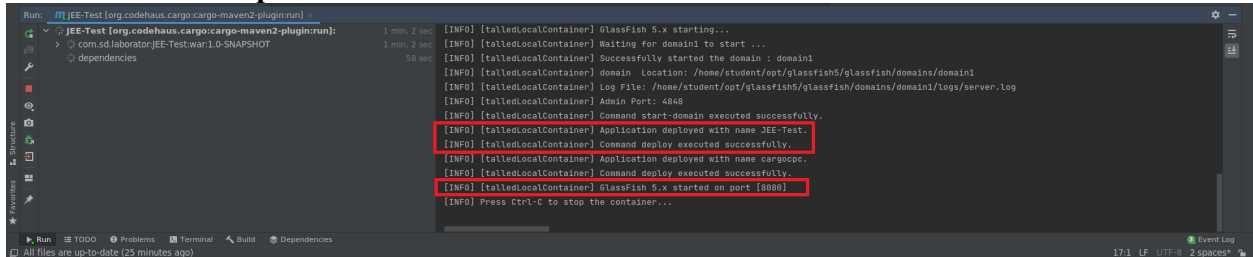


- Introduceți următoarea comandă și apăsați **ENTER**:

```
mvn org.codehaus.cargo:cargo-maven2-plugin:run
```

Atenție, prefixul `mvn` este adăugat automat de IntelliJ! Nu îl duplicați.

Acest goal Maven vă pornește server-ul GlassFish și, totodată, face și încărcarea artefactului WAR pe server.



Se poate observa că server-ul GlassFish a fost pornit cu succes, iar aplicația creată, împachetată sub formă de arhivă WAR a fost încărcată.

URL-ul de unde aplicația va fi disponibilă pentru testare este:

http://localhost:8080/<NUME_PROIECT>

În acest caz, dacă accesați <http://localhost:8080/JEE-Test> (**ATENȚIE** - este **case sensitive!**) dintr-un browser web, veți primi ca răspuns o pagină cu un mesaj „Hello World!”.



Hello World!

Ceea ce vedeți este rezultatul transformării într-un servlet a paginii de index din rădăcina folder-ului **webapp** (adică **index.jsp**). Paginile JSP sunt transformate automat în servleți de container-ul web!

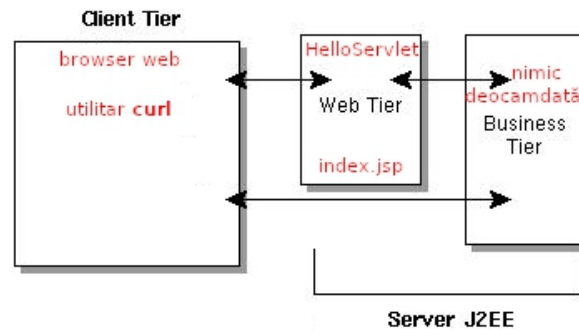
Prin accesarea rutei **/JEE-Test**, de fapt accesați rădăcina aplicației JEE create, deci server-ul vă va servi pagina implicită pe care o găsește disponibilă (dacă există). În acest caz există, și se numește **index.jsp**.

Pentru a accesa servlet-ul **Hello**, trebuie trimisă o cerere HTTP de tip GET către următorul URL: <http://localhost:8080/JEE-Test/hello>

Puteți face acest lucru pur și simplu navigând către acel URL dintr-un browser web, sau trimițând cererea manual folosind utilitarul **curl** din Linux:

```
$ curl -X GET http://localhost:8080/JEE-Test/hello
```

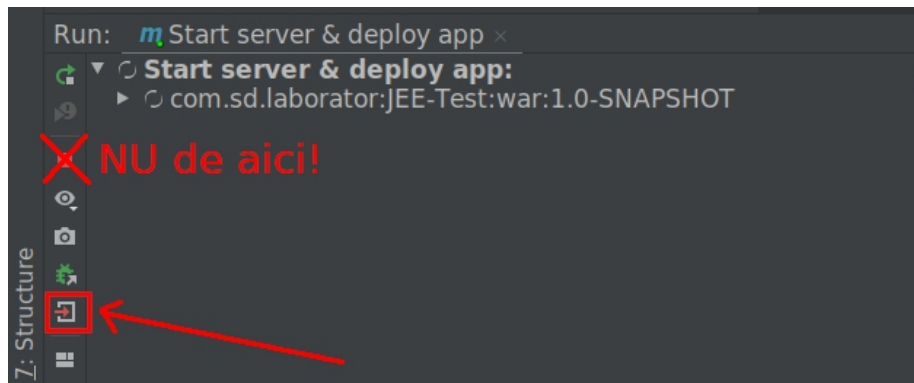
În acest moment, aplicația are următoarele componente, create în pașii efectuați până acum:



(structura aplicației create până acum)

1.7. Oprirea corectă a server-ului GlassFish

Server-ul GlassFish se oprește folosind butonul **Exit** din panoul de control al execuției din partea din stânga jos a ferestrei IntelliJ:

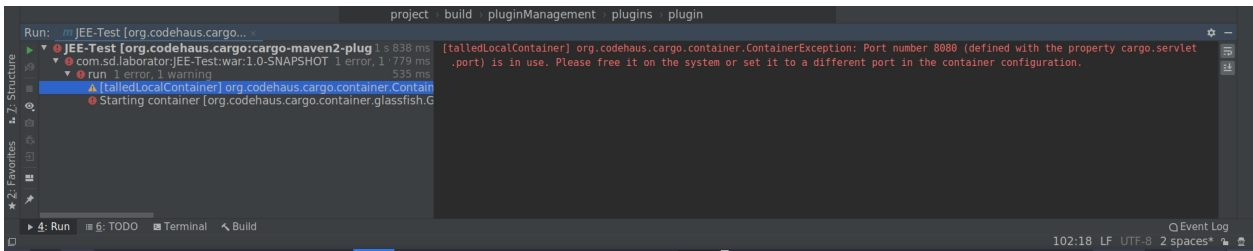


Veți ști că server-ul a fost oprit corect dacă vedeți următoarele mesaje în consola cu informații din partea de jos:

```
[INFO] [talledLocalContainer] Application deployed with name JEE-Test.
[INFO] [talledLocalContainer] Command deploy executed successfully.
[INFO] [talledLocalContainer] Application deployed with name cargocpc.
[INFO] [talledLocalContainer] Command deploy executed successfully.
[INFO] [talledLocalContainer] GlassFish 5.x started on port [8080]
[INFO] Press Ctrl-C to stop the container...
[INFO] [talledLocalContainer] GlassFish 5.x is stopping...
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 49.247 s
[INFO] Finished at: 2019-12-30T23:53:10+02:00
[INFO] -----
[INFO] [talledLocalContainer] Waiting for the domain to stop .
[INFO] [talledLocalContainer] Command stop-domain executed successfully.
[INFO] [talledLocalContainer] GlassFish 5.x is stopped
```

NU opriți server-ul folosind butonul Stop (pictograma pătrat roșu), deoarece procesul copil creat de *goal*-ul Maven care a pornit execuția GlassFish se va desprinde de procesul părinte controlat de IntelliJ și va rămâne în execuție în fundal, blocând portul 8080.

Dacă ați făcut din greșală acest lucru, veți primi o eroare dacă încercați să reporniți GlassFish pe același port implicit (8080):



Pentru a rezolva această problemă, executați următoarea comandă în terminal, ca să aflați PID-ul procesului care ocupă portul TCP 8080:

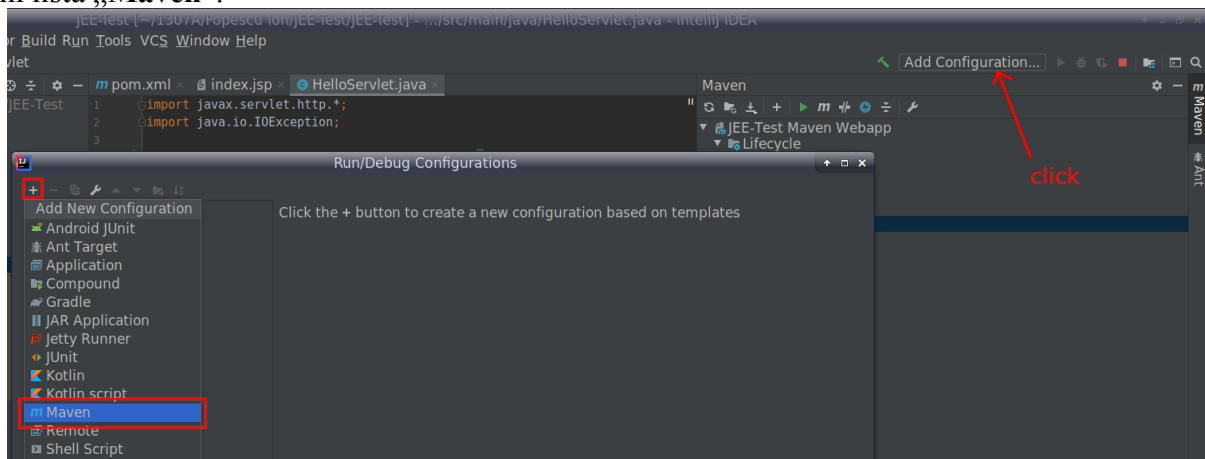
```
$ lsof -i tcp:8080
```

Folosiți PID-ul (de ex 3510) rezultat ca să închideți forțat acel proces:

```
$ kill -9 3510
```

1.8. Configurații de execuție

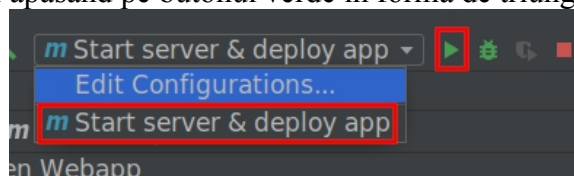
Pentru ușurință sporită, se pot crea configurații de execuție care execută Maven *goals* automat la apăsarea butonului „Run Configuration” din IntelliJ. În partea din dreapta-sus, apăsați pe „Add Configuration...”. În meniul afișat, apăsați pe semnul plus din stânga-sus, apoi alegeți din listă „Maven”.



Completați cu:

- Name: **Start server & deploy app**
- Command line: **org.codehaus.cargo:cargo-maven2-plugin:run**

Apoi apăsați pe OK, iar configurația de execuție va apărea în dreapta-sus a ferestrei IntelliJ. De acum, puteți porni server-ul GlassFish și încărca aplicația selectând configurația creată mai sus din listă, și apăsând pe butonul verde în formă de triunghi.



Apoi adăugați și o altă configurație des folosită, și anume reîncărcarea (*redeploy*) artefactelor:

- Name: **Redeploy**
- Command line: **org.codehaus.cargo:cargo-maven2-plugin:redploy**

1.9. Reîncărcare după modificarea surselor

Dacă modificați fișierele sursă / adăugați fișiere noi / ștergeți fișiere și vreți să testați aplicația în versiunea nouă pe server, trebuie să urmați obligatoriu pașii:

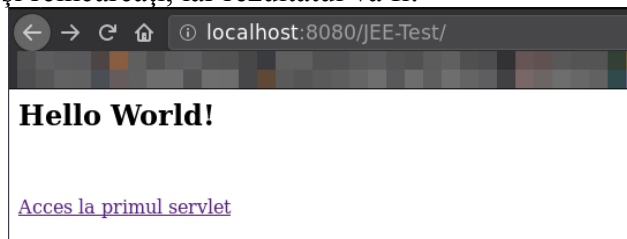
Compilare (compile) → Împachetare (package) → Reîncărcare (redeploy)

Cei trei pași au fost descriși în etapele anterioare.

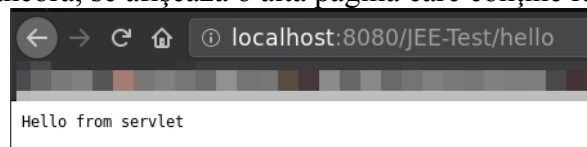
Spre exemplu, modificați pagina `index.jsp` pentru a include o ancoră HTML care redirecționează utilizatorul către calea prin care se accesează servletul `HelloServlet`.

```
<html>
  <body>
    <h2>Hello World!</h2>
    <br />
    <p>
      <a href="./hello">Acces la primul servlet</a>
    </p>
  </body>
</html>
```

Atenție la calea țintă, care este dată sub formă de cale relativă față de cea actuală: `./hello`. Compilați, împachetați și reîncărcați, iar rezultatul va fi:

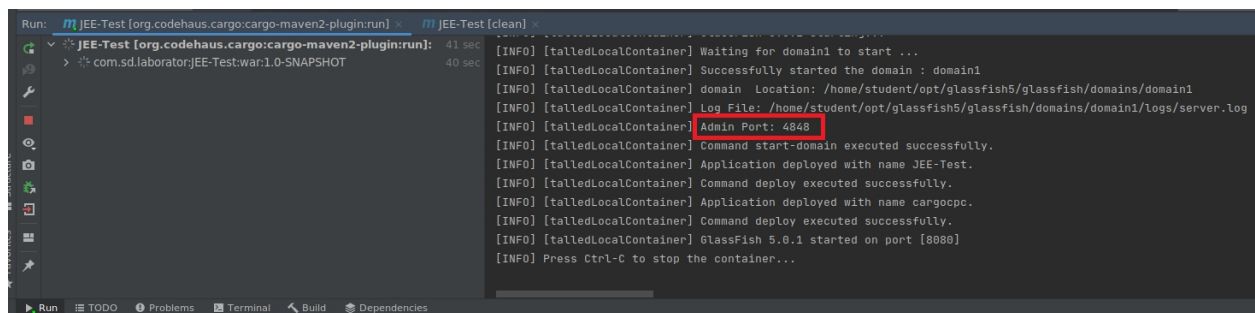


După apăsați pe ancoră, se afișează o altă pagină care conține răspunsul servlet-ului:

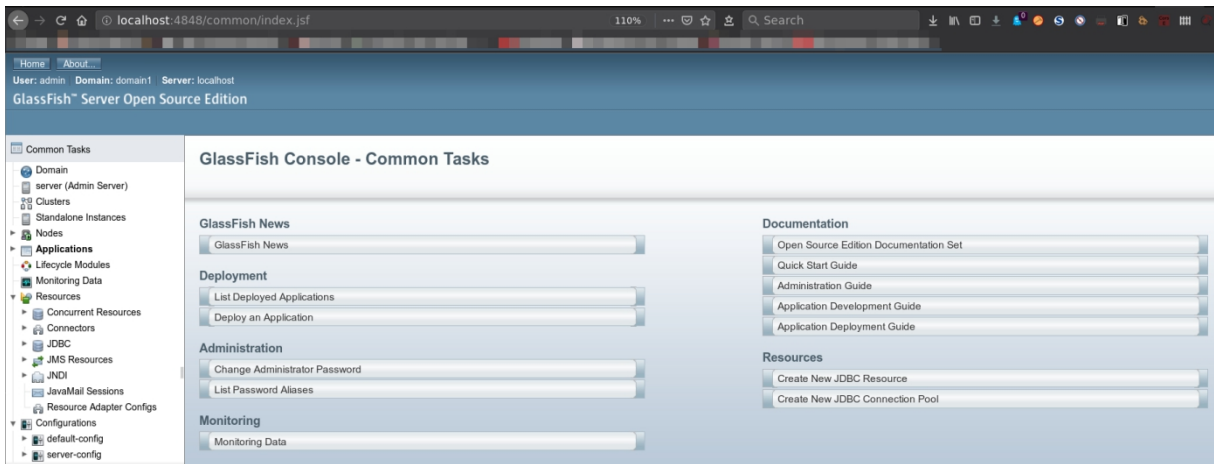


1.10. Accesul la consola de administrare GlassFish

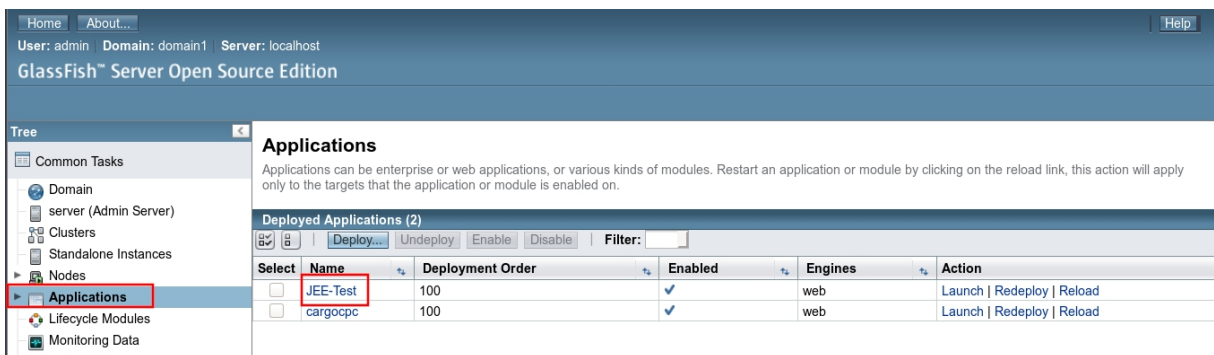
Având server-ul GlassFish pornit, se poate accesa consola de administrare ce este disponibilă în mod implicit pe portul 4848, după cum apare și în mesajele de la consola IntelliJ:



Accesați dintr-un browser web URL-ul <http://localhost:4848>, după ce v-ați asigurat că server-ul GlassFish a fost pornit cu succes.

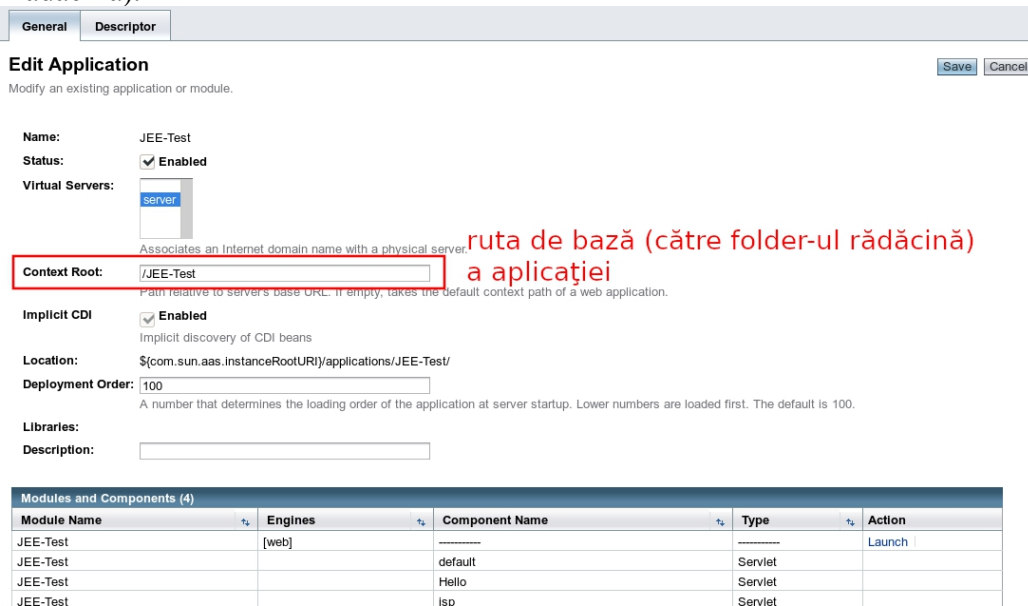


Puteți, de exemplu, vizualiza aplicațiile enterprise încărcate pe server, din meniul din stânga → **Applications**. Acolo ar trebui să apară aplicația creată în pașii anteriori, numită „JEE-Test”.



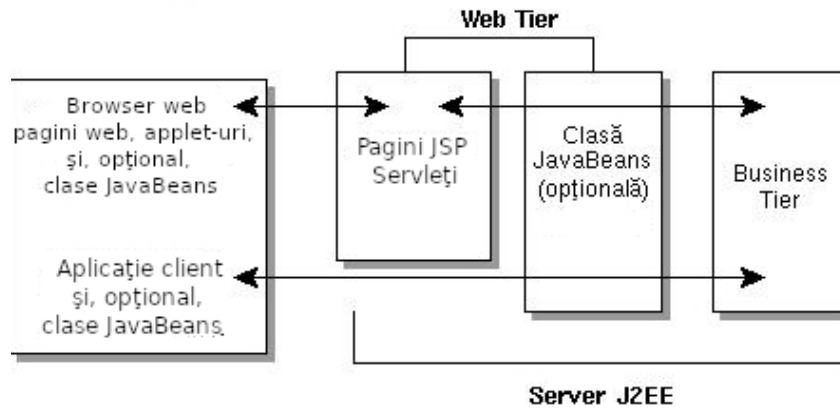
După ce apăsați dreapta pe numele aplicației, se pot vedea componentele și tipul acestora într-o listă afișată în partea de jos. Puteți observa că apar listați servleții generați la compilare: cel implicit (**default**), pagina JSP `index.jsp` (**jsp**), și servlet-ul **Hello** pe care l-ați creat și mapat sub calea `/hello`.

De asemenea, tot de aici se poate găsi calea care identifică aplicația încărcată (către folder-ul rădăcină).



2. Utilizarea JavaBeans

Implementarea anterioară poate fi îmbunătățită în scopul creșterii flexibilității, și astfel se obține abordarea de mai jos:



Aici stratul de web poate conține clase de tip *JavaBeans* (tehnologia precursoră *bean*-urilor de tip *enterprise*). Acestea sunt clase Java ce respectă anumite convenții de proiectare și pot fi reutilizate și transmise între componentele aplicației pentru întreprindere.

Clasele *JavaBeans* încapsulează mai multe obiecte într-unul singur (numit simplu, *bean*). Scopul lor este de ușura reutilizarea componentelor software. Dezvoltatorii pot folosi astfel de componente scrise de alții fără a fi nevoiți să le înțeleagă funcționalitatea internă. Aceste clase au următoarele proprietăți:

- sunt **serializabile**
- au un **constructor fără argument**
- proprietățile sunt private, iar accesul la acestea este permis doar prin intermediul **funcțiilor accesori** (*getters*) și **mutator** (*setters*)

În continuare, veți crea un formular web ce acceptă câteva date despre un student, le trimite unui servlet spre „procesare”, iar servlet-ul le încapsulează într-un *JavaBean* și le afișează într-o pagină JSP (stilul *Model-View-Controller*).

Creați o clasă Java denumită **StudentBean**, într-un nou pachet numit **beans**, în folderul cu surse **java**. Clasa are următoarea structură:

```
package beans;

public class StudentBean implements java.io.Serializable {
    private String nume = null;
    private String prenume = null;
    private int varsta = 0;

    public StudentBean() {
    }

    public String getNume() {
        return nume;
    }

    public void setNume(String nume) {
        this.nume = nume;
    }
}
```

```

public String getPrenume() {
    return prenume;
}

public void setPrenume(String prenume) {
    this.prenume = prenume;
}

public int getVarsta() {
    return varsta;
}

public void setVarsta(int varsta) {
    this.varsta = varsta;
}
}

```

Se observă că respectă toate caracteristicile unui *JavaBean*:

- Este serializabilă, deoarece implementează interfața **Serializable**
- are un constructor fără argumente
- conține proprietăți **private** accesibile prin metode *getter* și *setter*

Acum, creați o pagină JSP ce conține un formular prin care se cer datele unui student. Adăugați un fișier de tip „file” numit **formular.jsp** în folder-ul **webapp**:

```

<html xmlns:jsp="http://java.sun.com/JSP/Page">
  <head>
    <title>Formular student</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h3>Formular student</h3>
    Introduceti datele despre student:
    <form action="./process-student" method="post">
      Nume: <input type="text" name="nume" />
      <br />
      Prenume: <input type="text" name="prenume" />
      <br />
      Varsta: <input type="number" name="varsta" />
      <br />
      <br />
      <button type="submit" name="submit">Trimite</button>
    </form>
  </body>
</html>

```

Formularul conține trei câmpuri: nume, prenume și vârstă. Calea spre care sunt trimise datele introduse este **./process-student**. Metoda HTTP prin care datele sunt trimise este **POST**, așadar, logica de procesare a datelor va fi conținută în metoda **doPost()** a servlet-ului țintă.

În continuare, creați servlet-ul **ProcessStudentServlet** în folder-ul cu surse **java**. Introduceți codul Java:

```

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.time.Year;

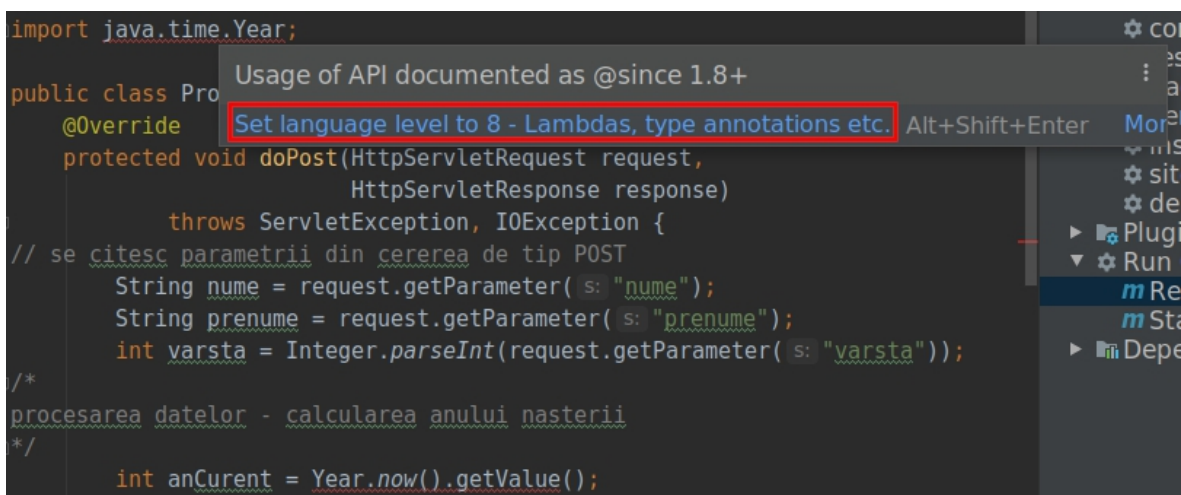
public class ProcessStudentServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        // se citesc parametrii din cererea de tip POST
        String nume = request.getParameter("nume");
        String prenume = request.getParameter("prenume");
        int varsta = Integer.parseInt(request.getParameter("varsta"));

        /*
        procesarea datelor - calcularea anului nasterii
        */
        int anCurent = Year.now().getValue();
        int anNastere = anCurent - varsta;

        // se trimit datele primite si anul nasterii catre o alta
        pagina JSP pentru afisare
        request.setAttribute("nume", nume);
        request.setAttribute("prenume", prenume);
        request.setAttribute("varsta", varsta);
        request.setAttribute("anNastere", anNastere);
        request.getRequestDispatcher("./info-
student.jsp").forward(request, response);
    }
}

```

Dacă mediul de dezvoltare nu permite utilizarea `java.time.Year` (permis începând cu Java 1.8), dați click pe clasa importată `java.time.Year`, așteptați câteva secunde sau apăsați ALT+ENTER și selectați „Set language level to 8 ...”, apoi „Import changes” în dreapta-jos.



Servlet-ul preia pur și simplu datele din parametrii cererii HTTP, calculează anul (aproximativ) al nașterii studentului și redirecționează datele către o altă pagină JSP numită `info-student.jsp`.

Mapați servlet-ul corespunzător în **web.xml**:

```
...
<servlet>
  <servlet-name>ProcessStudent</servlet-name>
  <servlet-class>ProcessStudentServlet</servlet-class>
</servlet>
...
<servlet-mapping>
  <servlet-name>ProcessStudent</servlet-name>
  <url-pattern>/process-student</url-pattern>
</servlet-mapping>
...
```

Așadar, urmează pagina finală, unde se afișează datele redirecționate de servlet-ul intermediar **ProcessStudentServlet**. În folder-ul **webapp**, creați o altă pagină JSP numită **info-student.jsp** și introduceți următorul conținut:

```
<html xmlns:jsp="http://java.sun.com/JSP/Page">
  <head>
    <title>Informatii student</title>
  </head>
  <body>
    <h3>Informatii student</h3>

    <!-- populare bean cu informatii din cererea HTTP -->
    <jsp:useBean id="studentBean" class="beans.StudentBean" />
    <jsp:setProperty name="studentBean" property="nume" value='<%=
request.getAttribute("nume") %>' />
    <jsp:setProperty name="studentBean" property="prenume" value='<%=
request.getAttribute("prenume") %>' />
    <jsp:setProperty name="studentBean" property="varsta" value='<%=
request.getAttribute("varsta") %>' />

    <!-- folosirea bean-ului pentru afisarea informatiilor -->
    <p>Urmatoarele informatii au fost introduse:</p>
    <ul type="bullet">
      <li>Nume: <jsp:getProperty name="studentBean"
property="nume" /></li>
      <li>Prenume: <jsp:getProperty name="studentBean"
property="prenume" /></li>
      <li>Varsta: <jsp:getProperty name="studentBean"
property="varsta" /></li>
      <li>Anul nasterii: <%=
request.getAttribute("anNastere") %></li>
    </ul>
  </body>
</html>
```

Pagina JSP utilizează sintaxa specifică pentru preluarea unui *JavaBean* în contextul de execuție (**jsp:useBean**), setarea unei proprietăți din *bean* (adică apelarea unui *setter* - **jsp:setProperty**) și preluarea unei proprietăți din *bean* (adică apelarea unui *getter* - **jsp:getProperty**). Câmpul calculat în servlet (**anNastere**) nu face parte din *bean*, deci este preluat pur și simplu din cererea HTTP, ca atribut suplimentar.

Alternativ, pentru a popula mai ușor *bean*-ul cu toate datele conținute în cererea HTTP, se poate proceda astfel:

```
<jsp:useBean id="studentBean" class="beans.StudentBean"
scope="request">
  <jsp:setProperty name="studentBean" property="*" />
</jsp:useBean>
```

În această variantă, proprietățile bean-ului vor fi populate automat **dacă numele proprietăților încapsulate coincid cu numele atributelor primite în cererea HTTP**. Alegeți varianta pe care o preferați.

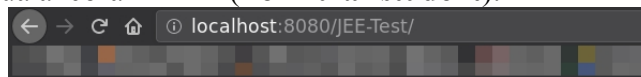
Adăugați o nouă ancoră HTML în pagina `index.jsp` pentru a avea legătură către formularul creat anterior:

```
...
<p>
  <a href="./formular.jsp">Formular student</a>
</p>
...
```

Compilați, împachetați și reîncărcați artefactul rezultat pe server-ul *enterprise*. Apoi, accesați următorul URL:

<http://localhost:8080/JEE-Test/>

Apăsați pe a doua ancoră HTML (**Formular student**):

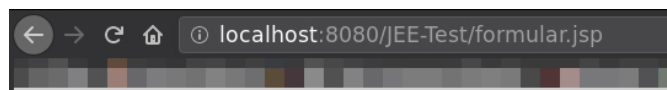


Hello World!

[Acces la primul servlet](#)

[Formular student](#)

Introduceți câteva date în câmpurile de completat și apăsați „Trimite”.

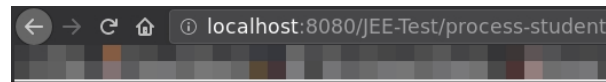


Formular student

Introduceți datele despre student:

Nume:
 Prenume:
 Varsta:

Observați că datele au fost stocate într-un *JavaBean* și preluate spre afișare de o pagină JSP. Servlet-ul intermediar a calculat un al patrulea câmp, anul nașterii (pentru a simula o „procesare” făcută în stratul web).

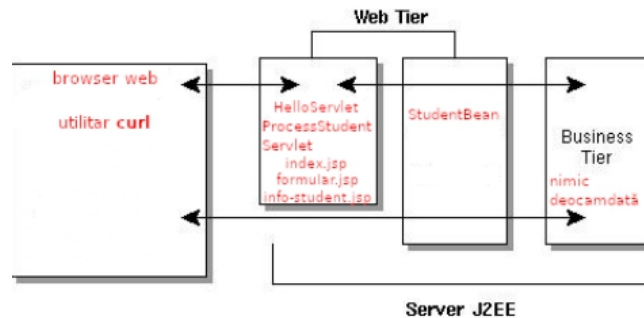


Informatii student

Urmatoarele informatii au fost introduse:

- Nume: Popescu
- Prenume: Ion
- Varsta: 23
- Anul nasterii: 1996

Adăugând funcționalitatea din pașii anteriori, aplicația dvs. are următoarea arhitectură în acest moment:



3. O variantă simplistă de persistență a datelor

Aplicația demonstrativă conține funcționalități de bază ce nu persistă datele care sunt implicate în fluxul de execuție: servlet-ul **ProcessStudentServlet**, spre exemplu, doar preia datele din formular și le trimite unei pagini JSP pentru afișare.

Pentru finalul laboratorului, veți implementa o variantă simplă de persistență a datelor încapsulate în *JavaBean*-ul **StudentBean**. Mai precis, se dorește ca atunci când utilizatorul accesează formularul pentru introdus datele despre student, completează și apasă pe butonul „Trimite”, servlet-ul țintă nu doar preia datele și le redirecționează, ci le și salvează (**serializează**) într-un fișier XML. Pentru aceasta, veți folosi, spre exemplu, serializatorul **Jackson 2.x**

(<https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml>).

Adăugați **Jackson Dataformat** ca dependență în proiectul Maven, în fișierul **pom.xml**, ca subordonat al tag-ului **<dependencies>**:

```

<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.11.4</version>
</dependency>

```

Apoi modificați codul servlet-ului **ProcessStudentServlet**, adăugând următorul bloc de cod înainte de trimiterea datelor către pagina de afișare JSP:

```

...
...

// initializare serializator Jackson
XmlMapper mapper = new XmlMapper();

// creare bean si populare cu date
StudentBean bean = new StudentBean();
bean.setNume(nume);
bean.setPrenume(prenume);

```

```

bean.setVarsta(varsta);

// serializare bean sub forma de string XML
mapper.writeValue(new File("/home/student/opt/1307A/Popescu
Ion/student.xml"), bean);

...

```

XmlMapper este clasa principală utilizată pentru operațiile de serializare / deserializare ale **Jackson 2.x**.

Pentru citirea datelor, creați un nou servlet numit **ReadStudentServlet** care, de această dată, va fi accesat direct din „meniul” de pe pagina principală **index.jsp**. Adăugați următorul conținut în corpul fișierului sursă:

```

import beans.StudentBean;
import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;

public class ReadStudentServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        // deserializare student din fisierul XML de pe disc
        File file = new File("/home/student/opt/1307A/Popescu
Ion/student.xml");

        // se returneaza un raspuns HTTP de tip 404 in cazul in care
nu se gaseste fisierul cu date
        if (!file.exists()) {
            response.sendError(404, "Nu a fost gasit niciun student
serializat pe disc!");
            return;
        }

        XmlMapper xmlMapper = new XmlMapper();
        StudentBean bean = xmlMapper.readValue(file,
StudentBean.class);

        request.setAttribute("nume", bean.getNum());
        request.setAttribute("prenume", bean.getPrenume());
        request.setAttribute("varsta", bean.getVarsta());

        // redirectionare date catre pagina de afisare a informatiilor
studentului
        request.getRequestDispatcher("./info-
student.jsp").forward(request, response);
    }
}

```

Acest servlet răspunde la cereri HTTP de tip GET, iar corpul metodei de tratare a

cererilor face următoarele acțiuni: verifică dacă pe disc există un fișier numit **student.xml** la calea (de exemplu) **/home/student/opt/1307A/Popescu Ion/student.xml**.

Dacă nu există, înseamnă că niciun bean de tip `StudentBean` nu a fost serializat și atunci se returnează un răspuns de tip **404 Not Found** clientului apelant (cu un mesaj explicativ, bineînțeles).

Dacă fișierul există, se folosește metoda pentru deserializare a Jackson 2.x pentru a popula un obiect de tip `StudentBean` cu datele extrase din fișier. Având datele în memorie, acestea sunt setate ca atribute ale cererii HTTP ce este redirecționată în continuare către aceeași pagină JSP pentru afișare, **info-student.jsp** (deja existentă).

Nu uitați că servlet-ul trebuie mapat sub o anumită rută de acces în fișierul cu descriptori XML **web.xml**. Adăugați o mapare pentru acest servlet sub calea **/read-student**, astfel:

```
...
<servlet>
  <servlet-name>ReadStudent</servlet-name>
  <servlet-class>ReadStudentServlet</servlet-class>
</servlet>

...
...

<servlet-mapping>
  <servlet-name>ReadStudent</servlet-name>
  <url-pattern>/read-student</url-pattern>
</servlet-mapping>
```

Observați că acest servlet nu calculează acel câmp suplimentar cu anul nașterii studentului, așadar ați putea trata acest caz în pagina JSP **info-student.jsp**, modificând modul de afișare al câmpului respectiv:

```
...
<!-- anul nasterii nu face parte din bean, il afisam separat (daca exista) -->
<li>Anul nasterii: <%
  Object anNastere = request.getAttribute("anNastere");
  if (anNastere != null) {
    out.print(anNastere);
  } else {
    out.print("necunoscut");
  }
%></li>
...
```

În paginile JSP, între simbolurile `<%` și `%>` se poate încapsula cod Java (**scriptleți**). În domeniul paginii JSP este disponibil obiectul `out` ce reprezintă fluxul de ieșire cu care se pot afișa date în pagină.

Pentru preluarea anului nașterii s-a folosit clasa de bază `Object` pentru a nu restrânge generalitatea tipului de date primit ca atribut: se poate primi un număr întreg, un șir de caractere etc.

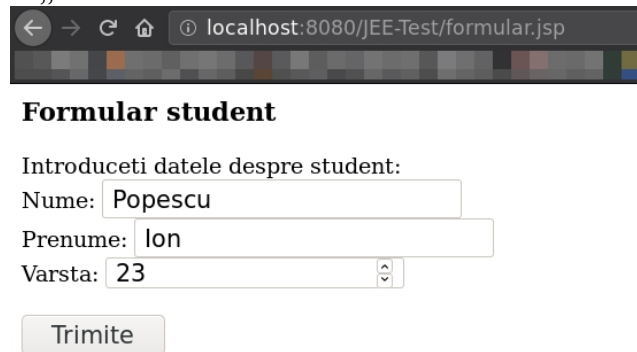
Nu în ultimul rând, adăugați în meniul principal (pagina **index.jsp**) o nouă ancoră care

face trimitere la noul servlet creat pentru citirea datelor unui student dintr-un fișier XML:

```
...  
<p>  
    <a href="./read-student">Vizualizare student</a>  
</p>  
...
```

Pentru a testa modificările aduse aplicației, urmați pașii: **compile** → **package** → **redeploy** (sau „**Start server & deploy app**” dacă server-ul GlassFish este oprit).

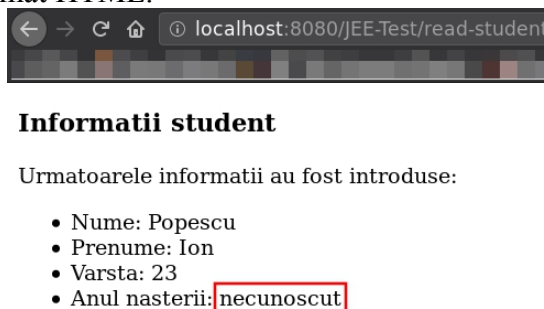
Accesați URL-ul <http://localhost:8080/JEE-Test> și adăugați un student nou prin completarea formularului „**Formular student**”.



După trimiterea formularului, veți fi redirecționat către pagina de afișare a informațiilor. Datele au fost serializate în fișierul XML (în acest exemplu) `/home/student/opt/1307A/Popescu Ion/student.xml`.



Verificați acest lucru prin accesarea noii secțiuni „**Vizualizare student**” din meniul principal. Servlet-ul care vă oferă rezultatele citește datele din acel XML și le pasează paginii JSP, ce vi le afișează în format HTML.

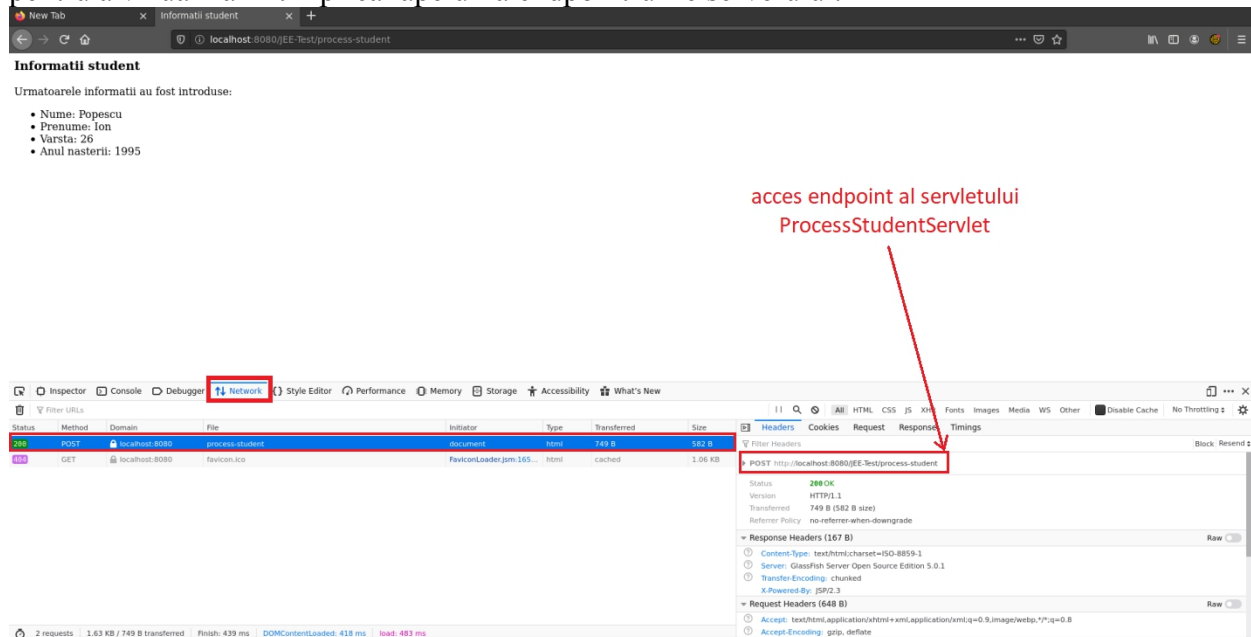


Observați că anul nașterii este afișat ca necunoscut, deoarece nu a fost calculat în acest caz. De asemenea, se poate verifica direct, rapid, care date sunt păstrate pe disc în fișierul XML, folosind comanda:

```
$ cat /home/student/opt/1307A/Popescu \ Ion/student.xml
```

Atenție la **backslash spațiu!**

Se recomandă la acest laborator, ca la testarea aplicațiilor în clientul de browser, să se deschidă Network din secțiunea de inspectare (Ctrl+Shift+I sau F12 sau click-dreapta/Inspect), pentru a vizualiza în timp real apelul la endpoint-urile serverului.



Teme de laborator

- Adăugați posibilitatea de a actualiza / șterge informațiile despre student din fișierul XML. De exemplu, puteți modifica pagina care vizualizează informațiile să completeze automat câmpuri HTML al unui alt formular, pe baza a ceea ce s-a citit din XML. Sub acel formular se poate adăuga un buton de „**Actualizare**” care va avea ca țintă un servlet sau o pagină JSP creată pentru acest scop.

Teme pe acasă

- Adăugați posibilitatea de a introduce mai multe seturi de informații despre studenți și înlocuiți fișierul xml cu o bază de date sqlite. Se vor implementa operațiile CRUD pentru studenți inclusiv căutarea unui anumit student după un criteriu stabilit (de ex. după nume sau prenume). Trebuie să fie și o opțiune suplimentară separată ca implementare care să permită exportul întregii baze de date (la un moment dat) ca JSON. Rezultatele se vor afișa ca răspuns al unui servlet sau într-o pagină JSP (la alegere).

Bibliografie

- [1] Overview JEE - <https://javaee.github.io/firstcup/java-ee001.html>
- [2] Documentație JEE 8 - <https://javaee.github.io/glassfish/documentation>
- [3] Tutorial JEE - <https://javaee.github.io/tutorial/toc.html>
- [4] Documentație Cargo Plugin (+ listă de Maven *goals* disponibile) - <https://codehaus-cargo.github.io/cargo/Maven+2+Plugin.html>
- [5] Tutorial Oracle JavaBeans - <https://docs.oracle.com/javase/tutorial/javabeans/>
- [6] Java Servlet Technology - <https://docs.oracle.com/javaee/7/tutorial/servlets.htm>
- [7] Java Server Pages Technology - <https://www.oracle.com/technetwork/java/whitepaper-142163.html>