

Bilgi Teknolojileri Notları

Vim, Tmux, Linux, Java, Database, Web Development ve daha fazlası

Çağatay Çakır

Copyright © 2018 Çağatay Çakır

PUBLISHED BY STARGATE INC.

BOOK-WEBSITE.COM

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, March 2018

İçindekiler

I

Bölüm Bir

1	Servlets	7
1.1	Örnek Bir Program: Ön Giriş	7
1.1.1	Maven` a Kısa Bir Giriş	7
1.1.2	Vücut Kütle Endeksi Programı, Örnek Projenin Ayakları	9

Bölüm Bir

1	Servlets	7
1.1	Örnek Bir Program: Ön Giriş	

1. Servlets

Servletler html sayfaları ile java programları arasında bir nevi arayüz olarak tanımlanabilir. Kendileri de birer java programı olan bu kod parçalarının ana amacı MVC (model view controller) yaklaşımı doğrultusunda iş (business-model) ile web arayüzü arasındaki iletişimi kontrol etmektir. Aslında sadece kontrol değil bizzat kendileri de sayfa generate etmek içinde kullanılabilirler.

1.1 Örnek Bir Program: Ön Giriş

Projemizin dosya düzeni Şekil 1.1 de görüldüğü gibidir. Ön alt yapıyı maven ile otomatik de oluşturabiliriz. Bunun için bilgisayarınıza maven' ı yükledikten sonra konsolda `mvn archetype:generate` yazın. Sizden bir iskelet yapı seçmenizi isteyen webapp yazın, ikinci çıkan listeden en temel web applicasyonunu seçin (güncel olarak ikinci liste numarası 38). Projeyi derlemek ve hemen kullanıma koymak için maven aracını kullanmak bize büyük esneklik ve kolaylık sağlar. Altta olup bitenden çok kopmadan projenin kontrolü tamamen sizde olur. Özellikle ilk başta bir IDE kullanarak yapılan projeler genellikle ayrıntıya hakim olamama sorununa yol açabilir. Diğer bir yol ise javac ile derlemeleri yapıp manuel olarak classları yerlerine koymaktır ancak küçük projelerde bu yapılabilir olsa da biraz büyük projeler için ayrı bir iş yükü getireceğinden tavsiye edilmemektedir.

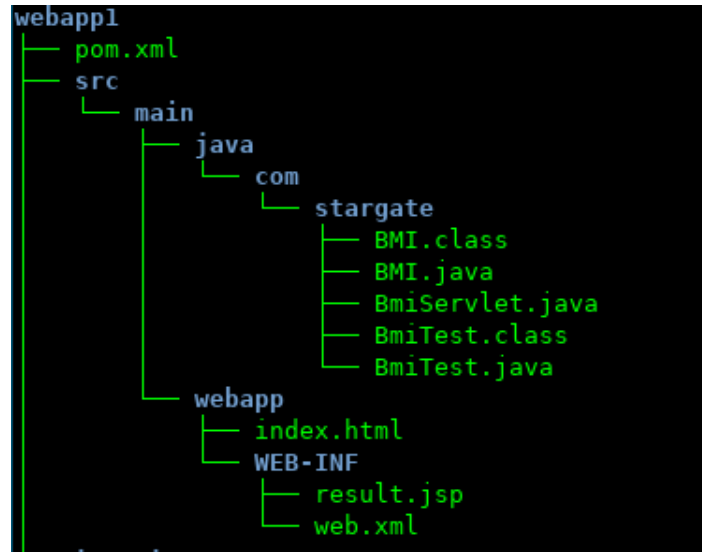
1.1.1 Maven' a Kısa Bir Giriş

C++ için *make* neyse java için *textbfemph*Maven odur. Maven aslında sadece bir yapılandırma aracı (build tool) değil eklentili yapısı ile tüm projeyi ve deployment' ı ayarlamaya ve şekillendirmeye yarayan bir araçtır.

Projenin derlenmesi için kaynak (src) klasörünün yanında *pom.xml* adındaki proje konfigurasyon dosyasının oluşturulması gerekir (Bkz. Şekil 1.1 ve Liste 1.1). Maven *pom.xml* dosyasında belirttiğimiz bağımlılıkları ve kütüphaneleri de otomatik olarak kurar.

Listing 1.1: Maven Yapılandırma Dosyası: pom.xml

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



Şekil 1.1: "webapp1" Projesinin Dosya Yapısı

```

2  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0_
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
3  <modelVersion>4.0.0</modelVersion>
4
5  <groupId>com.stargate</groupId>
6  <artifactId>webapp1</artifactId>
7  <version>1.0</version>
8  <packaging>war</packaging>
9
10 <name>webapp1</name>
11 <url>http://maven.apache.org</url>
12
13 <dependencies>
14   <dependency>
15     <groupId>javax.servlet</groupId>
16     <artifactId>servlet-api</artifactId>
17     <version>2.4</version>
18     <scope>provided</scope>
19   </dependency>
20   <dependency>
21     <groupId>javax.servlet.jsp</groupId>
22     <artifactId>jsp-api</artifactId>
23     <version>2.1</version>
24     <scope>provided</scope>
25   </dependency>
26 </dependencies>
27
28 <build>
29   <plugins>
30     <plugin>
31       <groupId>org.apache.maven.plugins</groupId>
32       <artifactId>maven-compiler-plugin</artifactId>
33       <version>2.0.2</version>
34       <configuration>
35         <source>1.7</source>

```



```

36     <target>1.7</target>
37   </configuration>
38 </plugin>
39 <plugin>
40 <groupId>org.apache.tomcat.maven</groupId>
41 <artifactId>tomcat7-maven-plugin</artifactId>
42 <version>2.2</version>
43   <configuration>
44     <path>/${project.build.finalName}</path>
45     <update>true</update>
46     <url>http://localhost:8080/manager/text</url>
47     <username>caga</username>
48     <password>Peace112</password>
49   </configuration>
50 </plugin>
51 </plugins>
52 </build>
53 </project>

```

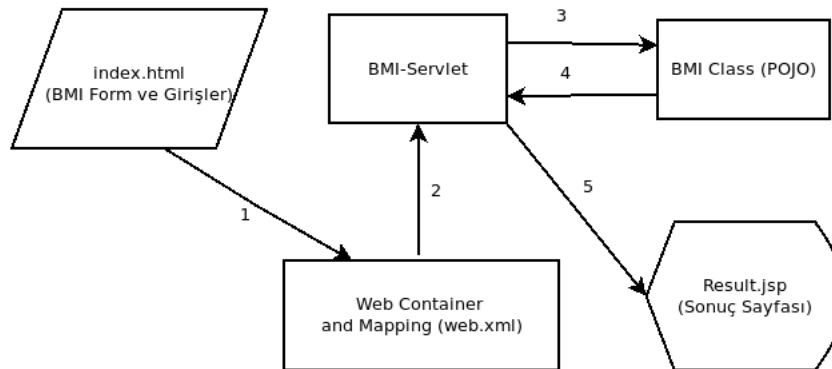
Projeyi derlemek için pom dosyasının olduğu proje klasörü içerisinde *mvn:compile* yazılır. Projeyi server'a yüklemek için "*mvn tomcat7:deploy*" veya daha önce yüklemiş ama güncellemek istiyorsak "*mvn tomcat7:redploy*" yazmamız yeterlidir. Local serverımız da (<http://localhost:8080/webapp1-1.0>) altında projemizi inceleyebiliriz. Pom.xml içerisinde projenin yükleneceği adresi ve diğer ayarları yapabiliriz.

1.1.2 Vücut Kütle Endeksi Programı, Örnek Projenin Ayakları

Projemiz MVC altyapısına uygun olarak biri karşılama diğeri sonuç sayfası olmak üzere iki adet seyir (view) sayfası, istekleri (request) alan işleyen ve yönlendiren bir adet servlet ve Vücut Kütle Endeksi (BMI) hesabını yapan bir adet Java (POJO) sınıfından oluşmaktadır.

Şekil 1.2 de görüldüğü üzere arada bir de Web Container (Tomcat veya Jetty gibi bir server) bir katman daha ve bir mapping dosyası (web.xml) bulunmaktadır.

Önce index sayfasından (Liste 1.2) servlet'e bir istek gelmekte, container tarafından bu istek çözülüp BMI servlete iletilmektedir.



Şekil 1.2: Proje (webapp1) Akış Diagramı

Listing 1.2: Index Sayfa Kodu

```

1 <!DOCTYPE html>
2 <html>

```

```

3  <head>
4    <meta charset="UTF-8" />
5    <meta name="viewport" content="width=device-width" />
6    <title>Vucut Kute Endeksi</title>
7  </head>
8  <body> <center>
9    <h1>Merhaba, VKE degerinizi ogrenmek icin lutfen asagıdaki
      degerleri giririniz</h1>
10
11    <form action="bmi" method="post" accept-charset="utf-8">
12      <p>İsim: </p> <input type="text" name="name" id="name"
        value="Mahmut" />
13      <p>Soyisim: </p> <input type="text" name="surname"
        id="surname" value="Işık" />
14      <p>Kilo: </p> <input type="text" name="kilo" id="kilo"
        value="64" />
15      <p>Boy: </p> <input type="text" name="boy" id="boy"
        value="1.68" />
16      <input type="submit" value="Gönder" />
17
18    </form>
19  </center> </body>
20 </html>

```

Giriş sayfasından gelen request objesini servletimiz içerisine yazdığımız kodlarla parse edebiliyoruz. Mesela Liste (1.3) de 11. satırda name parametresini request objesinden çekip program içerisinde string olarak tanımladığımız yine name isimli bir değişkene atıyoruz. Aynı şekilde giriş (index) sayfasından request objesine yazılan diğer parametreleri de çekiyoruz. Burada dikkatinizi celb edecek bir şey belki double tipinde olan değişkenleri, mesela kilo, double olarak almak için Double wrap classının parsırını kullanmamız ki bu parser string den double a değişkeni dönüştürmemizi sağlıyor. Zira request objesinin içindeki ifadeler string veya char biçiminde. Bunları direkt atamak hataya sebep olacaktır.

Listing 1.3: Servlet Kodu

```

1  package com.stargate;
2  import com.stargate.BMI;
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import java.io.*;
6  public class BmiServlet extends HttpServlet{
7    private static final long serialVersionUID = 1L;
8
9    public void doPost(HttpServletRequest
      request,HttpServletResponse response)
10     throws ServletException,IOException{
11     request.setCharacterEncoding("UTF-8");
12     String name=request.getParameter("name");
13     String surname=request.getParameter("surname");
14     double kilo=Double.parseDouble(request.getParameter("kilo"));
15     double boy=Double.parseDouble(request.getParameter("boy"));
16     BMI bmi=new BMI(kilo,boy);
17     String result=bmi.toString();
18     System.out.println("boy: "+boy+" kilo: "+kilo+" result: "+
      result);
19     request.setAttribute("bmi",result);

```

```

20 System.out.println(request.getAttributeNames());
21 System.out.println(request.getAttribute("bmi"));
22 RequestDispatcher
    view=request.getRequestDispatcher("WEB-INF/result.jsp");
23 view.forward(request,response);
24 }
25 }

```

Çekilen parametreleri daha sonra Liste (1.4) de görülen Java classına hesap için gönderiyoruz.

Listing 1.4: BMI Java Kodu

```

1 package com.stargate;
2
3 public class BMI {
4     double kilo;
5     double boy;
6     double result;
7     public void calculateBmi() {
8         this.result=kilo/(boy*boy);
9     }
10
11     public BMI(double kilo, double boy) {
12         this.kilo = kilo;
13         this.boy = boy;
14         calculateBmi();
15     }
16     public String toString(){
17         String s=String.format("%.2f",result);
18         return s;
19     }
20
21 }

```

Liste 1.3 da, 17. satırda BMI dan alınan sonucu result stringine depoluyoruz, daha sonra bu result ı 19. satırda bir attribute olarak request objesine ekliyoruz. 18, 20 ve 21 no' lu satırlar debug için koyduğumuz shell ekranına verileri basan satırlar. 22 nolu satırda request objesini view kısmına yani result.jsp sayfamıza aktarmak için view objesi tanımlıyoruz. 23 nolu satırda ise bu objenin forward methodu ile eklemeler yaptığımız request ve response objelerini sayfaya aktarıyoruz.

Listing 1.5: Sonuç Sayfa Kodu

```

1 <%@page import="java.util.*" contentType="text/html"
    pageEncoding="UTF-8"%>
2 <html>
3     <body>
4         <br>
5         <h2>      <%
6
7             String bmi=(String)request.getAttribute("bmi");
8             String name=request.getParameter("name");
9             String surname=request.getParameter("surname");
10            out.print("Merhaba "+name+" "+surname+"\n Vücut Kütle
                endeksin: "+bmi);
11            System.out.println(request);
12            %></h2>
13

```

```
14  
15     </body>  
16  
17  
18  
19 </html>
```