Assignment 3: Testo e soluzione

Un sito ha un messaggio di benvenuto (mot of the day) che viene modificato ogni giorno a cura dell'amministratore.
Una URL richiama una pagina nella quale l'utente può entrare se riconosciuto. Se non viene riconosciuto, la pagina richiede username e password (admin, nimda). Credenziali non valide segnalano all'utente l'errore, se valide danno accesso ad una pagina che consente di vedere la frase di benvenuto corrente, e di sostituirla con un'altra frase.
Un utente validato può tornare alla stessa pagina entro 10 minuti senza dover risottomettere username e password.

Il sito contiene altre pagine accessibili senza validazione.
Una home page contiene links a tre pagine:
- Quella di amministrazione
- Una che da' l'ora
- Una che da' la data

Tutte e quattro le pagine mostrano la frase di benvenuto.
Le pagine che danno la data e l'ora devono essere costruite secondo il pattern MVC, usando una servlet di accesso che costruisce un bean contenente il dato e richiama una JSP per la presentazione (che ovviamente legge il dato dal bean).

Iniziamo con il creare due frammenti di pagine che riuseremo in varie occasioni:

motd.jsp:

```
Hi! here is our message of the day:
<h2><%=application.getAttribute("motd") %></h2>
```

Questo mostra una informazione (il testo del mot of the day) che è trasversale a tutta la web app: deve essere accessibile ovunque, quindi va messo nel ServletContext (hiamato application nelle jsp).

links.html:

```
<hr>
Here is what you can do:
<ul>
    <li><a href="setMotdServlet">Set message of the day</a></li>
    <li><a href="getDateServlet">What's the date?</a></li>
    <li><a href="getTimeServlet">What's the time?</a></li>
</ul>
```

I link qui presenti rimandano ai controller delle varie operazioni (definisci il motd, ottieni la data, ottieni l'ora).

Usiamo poi i due files per costruire la home della webapp:

index.jsp:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Assignment 3</title>
</head>
<body>
<jsp:include page="motd.jsp"></jsp:include>
<%@ include file="links.html" %>
</body>
</html>
```

Notiamo come I due files da includere siano di tipo diverso: uno dinamico (motd.jsp) perché deve accedere ad una variabile, l'altro statico (links.html) perché non ha alcun parametro che varia. Quindi le forme di inclusione usate sono diverse, anche se avremmo potuto usare la <jsp:include> ovunque (ma usare la direttiva per inclusioni statiche è più efficiente - perché?).

Resta aperta una questione: come viene inizializzato l'attributo motd all'avvio della webApp? Avremmo potuto scrivere una cosa di questo tipo:

```
Hi! here is our message of the day:
<%if (application.getAttribute("motd")==null) {
      application.setAttribute("motd",("Have a nice day!") %>
<h2><%=application.getAttribute("motd") %></h2>
```

In questo modo alla prima esecuzione l'attributo "motd" non inizializzato assume un valore standard. Questa soluzione però mette troppo Java nella JSP, e sarebbe meglio fare in altro modo.

Una possibilità è di usare cose che a lezione abbiamo appena accennato, cioè un ServletContextListener. Scriviamo una classe che chiameremo StartupInitializer:

```
package it.unitn.disi.ronchet.assignment3;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class StartupInitializer implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent)
{
        ServletContext ctx = servletContextEvent.getServletContext();
        String message = ctx.getInitParameter("defaultMessage");
        ctx.setAttribute("motd",message);
        System.out.println("Context initialized");
    }
}
```

Come si vede, la classe implementa l'interfaccia ServletContextListener, ovvero risponde agli eventi del ServletContext. Tra questi, noi gestiamo il contextInitialized, ovvero scriviamo una callback che viene chiamata quando il contesto viene inizializzato, ovvero appena vien fatto il deploy dell'applicazione. Affinché la classe che abbiamo scritto sia utilizzata, dobbiamo poi registrarla come ascoltatore nel web.xml:

```
<listener>
    <listener-
class>it.unitn.disi.ronchet.assignment3.StartupInitializer</listener-class>
</listener>
```

Nel codice del nostro Listener avremmo potuto definire il messaggio di default scrivendo

```
        String message = "Have a nice day!";
```

Ma abbiamo fatto di meglio, definendo il testo del messaggio nel web.xml e lasciando nel codice solo un riferimento ad esso. In questo modo possiamo cambiare il messaggio di default senza ricompilare il codice.

```
      String message = ctx.getInitParameter("defaultMessage");
```

Nel web.xml avremo la seguente sezione:

```
<context-param>
    <param-name>defaultMessage</param-name>
    <param-value>Have a nice day!</param-value>
</context-param>
```

I context-param sono accessibili, come abbiamo visto, tramite il ServletContext, e quindi sono trasversali all'intera web-app (usabili ovunque).
Giacché ci siamo occupati del web.xml, vi definiamo anche un'altra proprietà: il nome del file da chiamare quando la webApp viene invocata in una URL senza specificare la pagina. (index.jsp è già tra i default, quindi avremmo anche potuto omettere questa specificazione.)

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

Settiamo anche la durata della sessione:

```
<session-config>
<session-timeout>10</session-timeout>
</session-config>
```

Ora possiamo passare alla scrittura dei controller, con i loro model e view. Iniziamo da quelli più facili, che ci danno la data e l'ora. Il controller in generale si occuperà di

- ottenere i parametri dati dall'utente (in questo caso nessuno),
- implementare la business logic per produrre l'output da mostrare,
- incapsulare i dati prodotti nel Model (il nostro bean)
- invocare la View passandole i bean (che in casi più complessi potrebbero essere più di uno).

La View preparerà la presentazione, iniettando in essa i valori recuperati dal Model (i beans).

La logica in questo caso è pochissima, e le informazioni che scriviamo nel Model sono elementari. Iniziamo dunque con il Model, e poi scriviamo il controller

```
package it.unitn.disi.ronchet.assignment3;
import java.io.Serializable;
public class TimeBean implements Serializable {
    String time;
    public TimeBean(){} // empty constructor required by the JavBean
definition
    public void setTime(String time) {
        this.time=time;
    }
    public String toString(){
        return time;
    }
}
```

Importante nel nostro bean definire la toString, che ci permetterà di usare il bean nella View in forma estremamente compatta.

Vediamo ora il controller:

```
package it.unitn.disi.ronchet.assignment3;
import … // import all the needed classes
@WebServlet(name = "getTimeServlet", value = "/getTimeServlet")
public class GetTimeServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
        // ---- business logic
        LocalTime d=LocalTime.now();
        String time=d.getHour()+":"+d.getMinute()+":"+d.getSecond();
        // ---- bean creation
        TimeBean tb=new TimeBean();
        tb.setTime(time);
        // ---- passing control to the view
        request.setAttribute("timeBean",tb);
        RequestDispatcher
rd=getServletContext().getRequestDispatcher("/time.jsp");
        rd.forward(request,response);
```

```
        System.err.println("should never print this line!");
    }
}
```

Passiamo infine alla view:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<jsp:useBean id="timeBean" scope="request"
class="it.unitn.disi.ronchet.assignment3.TimeBean"></jsp:useBean>
<html>
<head>
    <title>Assignment 3 - Time</title>
</head>
<body>
<jsp:include page="motd.jsp"></jsp:include>
<hr>
It's <%=timeBean%> o'clock.
<%@ include file="links.html" %>
</body>
</html>
```

Come possiamo notare, aver definito il nostro bean che incapsula i risultati della business logic ed avendogli dato il metodo toString, abbiamo sostanzialmente introdotto una pseudotag `<%=timeBean%>` che ci stampa il valore calcolato!
La nostra view non include istruzioni Java: usa solo le inclusioni dei frammenti di files, del bean e la pseudotag.

Possiamo ripetere le stesse cose per la data:
DateBean (model):

```
public class DateBean implements Serializable {
    String date;
    public DateBean() {}
    public String toString(){
        return date;
    }

    public void setDate(String date) {
        this.date=date;
    }
}
```

GetDateServlet (controller):

```
@WebServlet(name = "getDateServlet", value = "/getDateServlet")
public class GetDateServlet extends HttpServlet {
    private String message;
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
        // ---- business logic
        LocalDate d=LocalDate.now();
        String date=d.getDayOfWeek().name()+", "+
                d.getDayOfMonth()+" "+
                d.getMonth().name()+" "+
                d.getYear();
        // ---- creazione del Bean
        DateBean db=new DateBean();
        db.setDate(date);
        // ---- passing control to the view
        request.setAttribute("dateBean",db);
        RequestDispatcher
```

```
rd=getServletContext().getRequestDispatcher("/date.jsp");
        rd.forward(request,response);
        System.err.println("should never print this after forwarding!");
    }
}
```

time.jsp(View)

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<jsp:useBean id="dateBean" scope="request"
class="it.unitn.disi.ronchet.assignment3.DateBean"></jsp:useBean>
<html>
<head>
    <title>Assignment 3 - Date</title>
</head>
<body>
<jsp:include page="motd.jsp"></jsp:include>
<hr>
Today is <%=dateBean%>
<%@ include file="links.html" %>
</body>
</html>
```

Passiamo ora alla parte di settaggio del motd, che richiede autenticazione.
Questa sezione è un pochino più complessa.

Nella home page abbiamo il link richiama (tramite GET, senza parametri) un controller per la gestione: setMotdServlet
In esso dovremo verificare che l'utente abbia le necessarie autorizzazioni: quindi in primo luogo verifichiamo se possiede una sessione nella quale vi sia un "timbro" che dice che l'utente è autenticato. Se non lo ha, proviamo a vedere se ci ha fornito username e password. Se non le troviamo (o se non sono valide) rimandiamo l'utente a una view di autenticazione (login.jsp) dalla quale tornerà da noi dopo aver fornito username e password (con una POST, con parametri). Se invece l'utente è autorizzato, gli mostriamo un view nella quale ci può fornire il nuovo modt (setMotd.jsp).

```
public class SetMotdServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        RequestDispatcher rd=null;
        HttpSession session=request.getSession();
        boolean isUserAuthorized=false;
        Object isAuthorizedObj=session.getAttribute("isAuthorized");
        if (isAuthorizedObj==null) {
            String validUsername=getInitParameter("username");
            String username=request.getParameter("username");
            String password=request.getParameter("password");
            String validPassword=getInitParameter("password");
            if (validUsername.equals(username) &&
validPassword.equals(password)) {
                isUserAuthorized=true;
                session.setAttribute("isAuthorized",isUserAuthorized);
            }
        } else if (isAuthorizedObj instanceof Boolean) {
            isUserAuthorized=(Boolean)isAuthorizedObj;
        }
        if (!isUserAuthorized) {
            rd=getServletContext().getRequestDispatcher("/login.jsp");
```

```java
            rd.forward(request,response);
            System.err.println("Printing after forwarding!");
        }
        rd=getServletContext().getRequestDispatcher("/setMotd.jsp");
        rd.forward(request,response);
        System.err.println("Printing after forwarding!");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }
```

I valori legali di username e password li abbiamo messi come parametri nel web.xml, e letti con `getInitParameter()`. A differenza di quanto abbiamo fatto inizialmente, non abbiamo usato `ctx.getInitParameter`. La chiamata di `getInitParameter()` sul servletContext legge parametri *globali* della WebApp (ovvero disponibili ovunque nella webApp), mentre la chiamata sulla HttpServlet (infatti è implicitamente chiamato su this) legge parametri locali, disponibili solo per la servlet corrente e dichiarati nel web.xml nella sezione della servlet stessa:

```xml
<servlet>
    <servlet-name>setMotdServlet</servlet-name>
    <servlet-
class>it.unitn.disi.ronchet.assignment3.SetMotdServlet</servlet-class>
    <init-param>
        <param-name>username</param-name>
        <param-value>admin</param-value>
    </init-param>
    <init-param>
        <param-name>password</param-name>
        <param-value>nimda</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>setMotdServlet</servlet-name>
    <url-pattern>/setMotdServlet</url-pattern>
</servlet-mapping>
```

Nota: il controller SetMotdServlet che abbiamo scritto risponde sia alla GET (richieste che arrivano da link.htm, e che si appoggiano sull'esistenza di credenziali nella session) che alla POST (richieste che dalla pagina di login, e quindi con credenziali nella request). Avremmo potuto separare il codice dei due casi, invece abbiamo preferito uniformarlo in un solo metodo che gestisce sia le GET che le POST (tramite il fatto che la doPost richiama la doGet).

Di seguito la login.jsp:

```jsp
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<jsp:useBean id="dateBean" scope="request"
class="it.unitn.disi.ronchet.assignment3.DateBean"></jsp:useBean>
<html>
<head>
    <title>Assignment 3 - Date</title>
</head>
<body>
<jsp:include page="motd.jsp"></jsp:include>
```

```
<hr>
Restricted operation. Please provide valid credentials.<br>
Enter username and password, or <a href="index.jsp">go back home</a>
<form action="setMotdServlet" method="POST">
    <label>Username</label><input type="text" name="username"><br>
    <label>Password</label><input type="password" name="password"><br>
    <input type="submit">
</form>
</body>
</html>
```

La view setMotd.jsp chiede all’utente il nuovo motd:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<jsp:useBean id="dateBean" scope="request"
class="it.unitn.disi.ronchet.assignment3.DateBean"></jsp:useBean>
<html>
<head>
    <title>Assignment 3 - Date</title>
</head>
<body>
<jsp:include page="motd.jsp"></jsp:include>
<hr>
Set new mot of the day, or <a href="index.jsp">go back home</a>
<form action="updateMotdServlet" method="POST">
    <label>new mot of the day</label><input type="text" name="newMotd"><br>
    <input type="submit">
</form>
</body>
</html>
```

e invoca il controller updateMotdServlet che ha lo scopo di aggiornare il motd. E’ necessario, all’inizio di questo controller, ricontrollare le credenziali, perché un utente malizioso potrebbe provare ad accedere direttamente a setMotd.jsp o a updateMotdServlet senza passare per la strada canonica che abbiamo previsto! In questo caso è sufficiente che controlliamo la sessione (un utente non malizioso giunto a questo punto deve averla), e altrimenti non effettuiamo l’aggiornamento richiesto. A operazione finita ridirigiamo alla home, e con questo chiudiamo il giro.

```
@WebServlet(name = "UpdateMotdServlet", value = "/updateMotdServlet")
public class UpdateMotdServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        HttpSession session=request.getSession();
        Object isAuthorizedObj=session.getAttribute("isAuthorized");
        if ((!(isAuthorizedObj==null))&&
                (isAuthorizedObj instanceof Boolean)&&
                ((Boolean)isAuthorizedObj)) {
            String newModt = request.getParameter("newMotd");
            getServletContext().setAttribute("motd", newModt);
        }
        RequestDispatcher rd=request.getRequestDispatcher("/index.jsp");
        rd.forward(request,response);
    }
}
```

Note ulteriori:
avere le view esposte all'utente può essere problematico. Ad esempio, l'utente potrebbe richiamare direttamente la seguente url:
http://localhost:8080/assignment3_war_exploded/date.jsp

Questa genererebbe errore, perché non essendo passata prima dal relativo controller troverebbe il bean null.

Andrebbe quindi impedito all'utente di accedere a passaggi intermedi. Per le jsp questo si può fare spostandole in WEB-INF, e aggiornando tutti i riferimenti (ad esempio, invece di
```
rd=getServletContext().getRequestDispatcher("/setMotd.jsp");
```

si dovrà mettere

```
rd=getServletContext().getRequestDispatcher("/WEB-INF/setMotd.jsp");
```

Va lasciata esposta (cioè fuori da WEB-INF) solamente index.jsp (oltre ai fragments: links.html e motd.jsp)

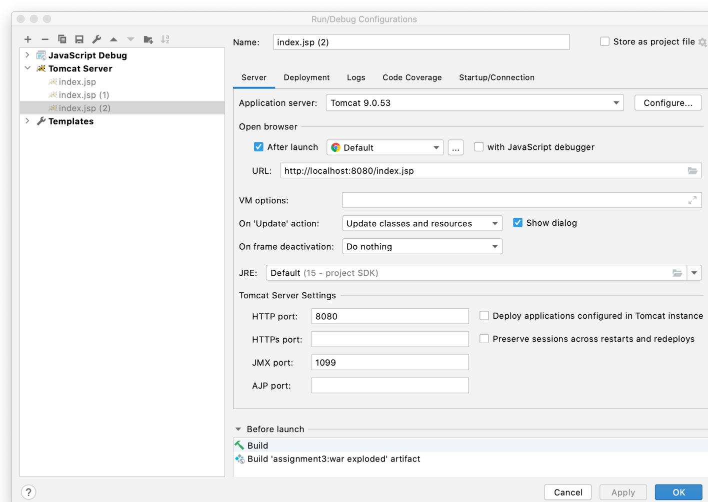
Quanto al fatto che l'utente potrebbe accedere direttamente alle URL dei controller, questo è meno grave, ma potremo controllare anche questo aspetto introducendo il concetto di Filtro (che sarà argomento di una prossima lezione).

NOTA PER IL DEPLOYMENT

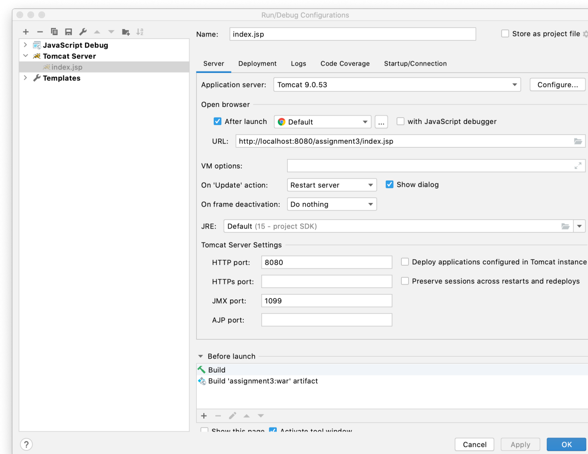
Trovo i default di IntelliJ particolarmente antipatici.
Quindi nel progetto contenente la soluzione ho proceduto nel seguente modo:

Run-> Edit Configuration

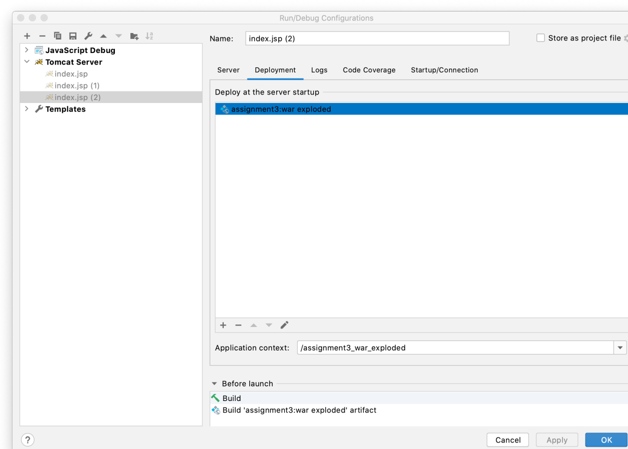
Si presenta così:



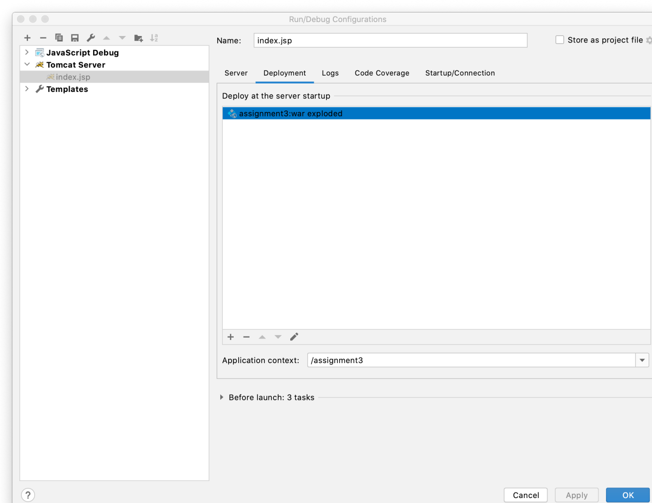
Cambio la URL da http://localhost:8080/index.jsp a http://localhost:8080/assignment3/index.html (Ho introdotto il nome che voglio usare per la mia webapp)



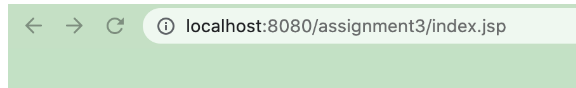
Ora devo settare tahe valore in modo che sia associato al deployment. Vado quindi nel tab Deployment che si presenta così:



Rinomino l'associazione (nella riga "Application Context") con il nome che ho scelto: assignment3, in modo da essere coerente con quanto settato poco fa.



Adesso finalmente la mia web app si comporta nel modo standard!



Hi! here is our message of the day:

Have a nice day!

Here is what you can do:

- Set message of the day
- What's the date?
- What's the time?