Ecole des Métiers
Berufsfachschule
Fribourg
Informatique

133 – tomcat

Rapport personnel

Version 1 du 07.05.23

Schwander Nicolas

Module du 09.08.2019 au 19.11.2019

Table des matières

1. Introduction

1.1 Objectifs du module

- Côté client

La liaison au métier par protocole http (XML et JSON)

HTML5-JS

Pattern MVC

Contrôle des évènements produits par l'utilisateur

- Côté serveur

Métier en JAVA (servlets et JSP)

Création et utilisation de WebServices

L'application Web en JSP

Pattern MVC

Contrôle des évènements produits par les appels des clients

Connexion au serveur MySQL

- En global

Hébergement APACHE et TOMCAT

Documentation et JAVADoc

2. TestsTechno

2.1 Déploiement Tomcat

2.1.1 Théorie

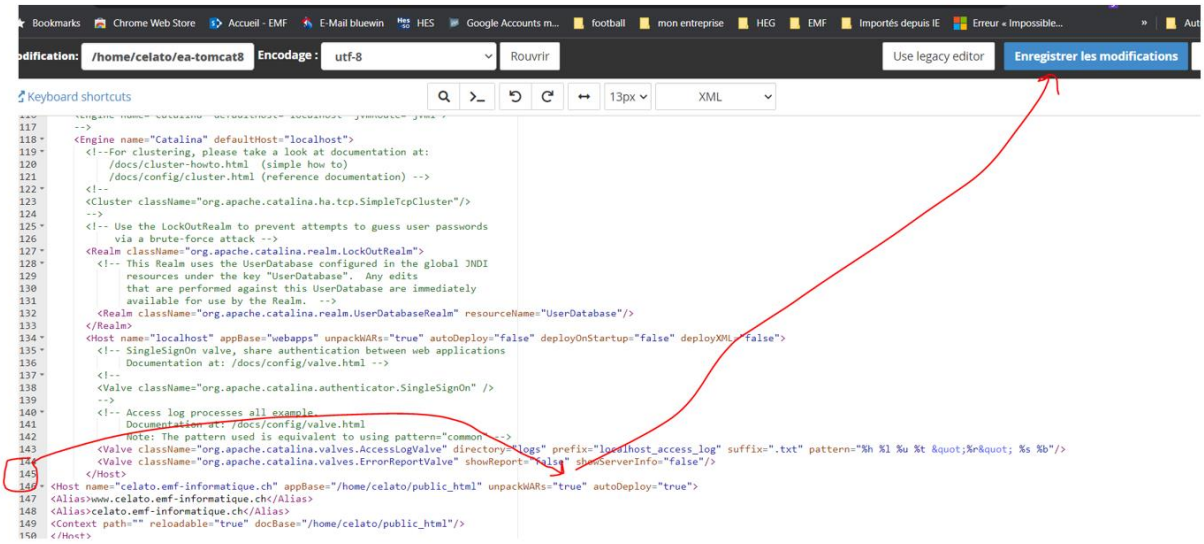
Tomcat est un serveur web principalement utilisé pour exécuter des applications web Java, telles que des sites web dynamiques et des applications d'entreprise.

Tomcat fonctionne en tant que conteneur de servlets, il peut gérer les requêtes et les réponses pour les applications web Java. Il utilise le protocole HTTP pour

communiquer avec les clients et il peut être configuré pour prendre en charge HTTPS pour les connexions sécurisées.

2.1.2 Tizoo Cpanel

Une version de tomcate était préalablement installé sur nos comptes Tizoo, il fallait simplement mofier un fichier de config.

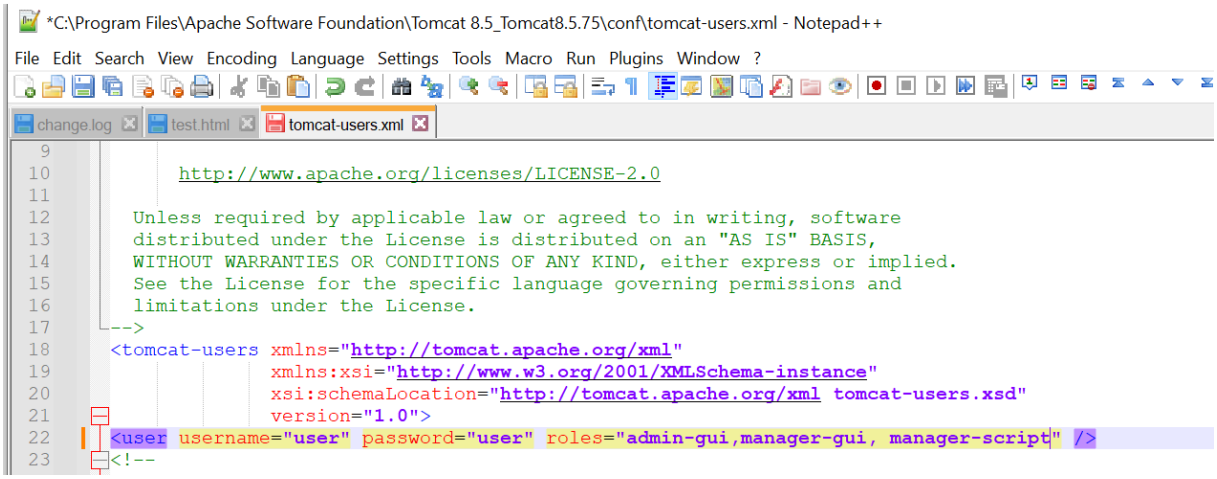


Cette variable permet a tomcat de créer automatiquement un repertoire avec le war unpacké. Ce qui rend la page utilisable.

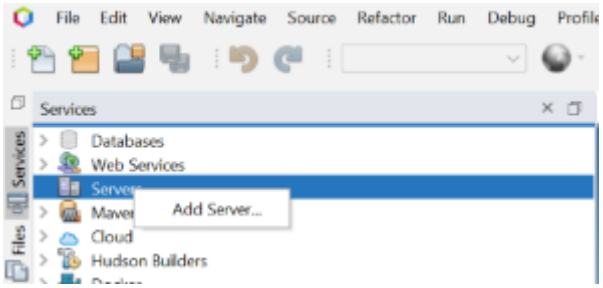
2.1.3 Local

Il a fallu simplement installer la version la plus recente de tomcat avec quelques modification notamment :

On a ajouté le rôle manager-script



Ensuite j'ai exécuté le script de startup de tomcat par le cmd et j'ai ajouté le server tomcat a netbeans :



2.2 JSP

Les JSP sont une technologie de développement web basée sur Java qui permet aux développeurs de créer des pages web dynamiques en combinant des fragments de code Java avec des modèles HTML.

Lorsqu'un navigateur demande une page JSP à un serveur web Tomcat, le serveur interprète la page JSP et génère du code HTML qui est renvoyé au navigateur. Le code Java inclus dans la page JSP est exécuté côté serveur pour effectuer des traitements et des calculs.

2.2.1 Exercice 2

Dans cet exercice il suffisait simplement de lancer une jsp qui contenait une architecture html et un hello world en titre.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World from JSP!</h1>
    </body>
</html>
```

2.2.2 Exercice 3

Dans cet exercice on fait un formulaire html qui pointe vers une jsp qui effectue simplement un if sur le mot de passe et nom d'utilisateur

2.2.3 Exercice 4

Dans cet exercice il a fallu utiliser un petit webservice, pour ce faire j'ai créé une petite jsp.

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>

        <%
            String ville = request.getParameter("ville");
        %>
        <h1>Meteo de <%=ville%></h1>
        <img src="https://www.prevision-meteo.ch/uploads/widget/<%=ville%>_0.png" width="650" height="250" />

    </p>
</body>
</html>
```

Celle-ci est appelé par un formulaire dans l'index.html

2.3 JDBC

2.3.1 Exercice 5

Le but dans cet exercice était de créer un petite application web java qui lis des données dans une DB.

Pour cela on a un workerDB qui permet d'ouvrir une connexion et de récupérer les pays dedans.

Du coté de la jsp il fallait simplement iterer a travers les pays et les afficher :

```jsp
<body>
    <h1>Ma requête SQL depuis JSP!</h1>
    <%
        WrkDB wrkDB = new WrkDB("3306", "schwandern_Ex5");
        ArrayList<String> lstPays = wrkDB.getPays();
        if (lstPays != null) {
    %>
    <h2>Liste des pays </h2>
    <%
        out.println("Solution avec instruction java out.println()");
        //A compléter

        for (String pays : lstPays) {
            out.print("<div>");
            out.print(pays);
            out.print("</div>");

        }

    %>
    <div>Solution avec intégration de variables java dans HTML</div>
    <%                    for (String pays : lstPays) {
    %>
    <div><%=pays%></div>

    <%

            }
        }
    %>
```

2.4 Bean with scope

Il existe en java web des manières de faire transiter des beans avec un scope qui définit jusqu'à ou il est atteignable

```
<jsp:useBean id="beanInfo"  scope="session" class="beans.BeanInfo"/>
<jsp:useBean id="beanError"  scope="session" class="beans.BeanError"/>
```

Ici le scope est définit comme « session » ce qui veut dire que c'est le même partout dans la même session.

Ensuite on set ce Bean :

```
   beanInfo.setNom(username);
   beanInfo.setPrenom(password);
```

et pour le réutiliser au sein d'une même session,

```
<%@page import="beans.BeanInfo"%>
<%@page import="beans.BeanError"%>
<jsp:useBean id="beanInfo"  scope="session" class="beans.BeanInfo"/>
<jsp:useBean id="beanError"  scope="session" class="beans.BeanError"/>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>vous etes loggué voici les informaions de l'utilisateur</h1>
        <p>Nom : <%=beanInfo.getNom()%></p>
        <p>Mot de passe : <%=beanInfo.getPrenom()%></p>

    </body>
</html>
```

2.5 Servlet

2.5.1 Exercice 7

Dans cet exercice nous avons découvert les servlet, il suffisait de faire un requête post avec un formulaire en html sur le servlet et on renvoyai les données traitées.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");


    try ( PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet MaServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet MaServlet at " + request.getContextPath() + "</h1>");
        out.println("<p>votre nom d'utilisateur est "+username+" et votre mot de passe est "+password+"</p>");
        out.println("</body>");
        out.println("</html>");
    }

}
```

2.6 Session varibles/beans

2.6.1 Exercice 8

En java web il existe des variables de session, dans l'exercice suivant il suffisait de faire transiter quelques donnes dans un bean stocké dans la session.

J'ai fait ceci de la manière suivante dans le servlet :

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");

    HttpSession session = request.getSession();
    session.setMaxInactiveInterval(100); //temps en seconde

    BeanInfo info = new BeanInfo();
    info.setNom(request.getParameter("username"));
    info.setmdp(request.getParameter("password"));

    BeanError error = new BeanError();
    error.setHost("127.0.0.1");
    error.setMessage("login incorrect pour host :");

    request.getSession().setAttribute("beanInfo", info);
    request.getSession().setAttribute("beanError", error);

    if (info.getNom().equals("admin") && info.getmdp().equals("emf")) {
        response.sendRedirect("pageAutorite.jsp");

    } else {
        response.sendRedirect("erreur.jsp");
    }
```

Ensuite pour récupérer les variables coté client :

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="beans.BeanInfo"%>
<%@page import="beans.BeanError"%>
<jsp:useBean id="beanInfo"  scope="session" class="beans.BeanInfo"/>


<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>vous etes loggué voici les informations de l'utilisateur</h1>
        <p>Nom : <%=beanInfo.getNom()%></p>
        <p>mot de passe : <%=beanInfo.getmdp()%></p>

    </body>
</html>
```

2.7 Exercice 10

Dans cet exercice, nous avons créé une application client-serveur pour accéder à des données stockées sur le serveur. La partie serveur était responsable de fournir les données en utilisant une API RESTful. La partie cliente était responsable de récupérer les données et de les afficher. Pour cela, nous avons utilisé une bibliothèque JavaScript appelée "jQuery" pour envoyer des requêtes AJAX au serveur et récupérer les données JSON. Ensuite, nous avons parcouru les données pour les afficher dans une liste sur la page web cliente.

2.7.1 Client

Dans cet exercice, nous avons développé une application client-serveur qui permet de récupérer des données stockées sur le serveur. Pour cela, nous avons créé deux boutons sur le client qui permettent de demander l'auteur et le message. Ces boutons sont liés à des fonctions JavaScript qui envoient des requêtes HTTP GET au serveur pour récupérer les données demandées. Le serveur renvoie alors les données stockées grâce à des méthodes spécifiées dans le code.

```jsp
<form method="post" action="ServletCtrl">
    <input type="submit" name="getMessage" value="Récupérer message"/>
    <input type="submit" name="getAuthor" value="Récupérer auteur"/>
</form>
```

Ensuite nous faisons 2 classes dont une classe est un servlet. La premier classe ClientMessage va nous permettre de faire la communication avec le serveur.

```jsp
public class ClientMessage {
```

```
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://gamberal01.emf-
informatique.ch/javaSimple_Rest_Server/webresources";

    public ClientMessage() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(BASE_URI).path("tutoriel");
    }

    public String getAuthor() throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path("getAuthor");
        return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class)
;
    }

    public String getMessage() throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path("getMessage");
        return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class)
;
    }

    public void close() {
        client.close();
    }

}
```

La classe ServletCtrl on a dû faire des ifs pour savoir si nous voulions recevoir le message, l'auteur ou s'il y a une erreur.

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try ( PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code.
*/
        if (request.getParameter("getMessage")!=null){
            String reponse = client.getMessage();
            out.println(reponse);
```

```
        }


        else if (request.getParameter("getAuthor") != null){
            String reponse = client.getAuthor();
            out.println(reponse);
        }
        else {
            out.println("Quelque chose ne s'est pas bien passé !");
        }
    }
}
```

2.7.2 Serveur

Dans cet exercice, nous avons dû créer deux classes dans le serveur. La première classe est appelée "ApplicationConfig" et nous avons dû spécifier son chemin de ressources web à "webresources". Ce chemin de ressources nous permet de rediriger vers les ressources appropriées pour notre application.

```
@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends Application {
```

Dans cet exercice, nous avons dû ajouter un chemin "tutoriel" pour la classe Message. Ensuite, pour chaque méthode, nous avons également dû spécifier un chemin. Si nous utilisons le chemin "/webresources/tutoriel/getMessage", cela nous renverra le message "Bonjour tout le monde".

```
@Path("tutoriel")
public class Message {
    @Context
    private UriInfo context;
    public Message() {
    }
    @GET
    @Path("getMessage")
    @Produces(javax.ws.rs.core.MediaType.TEXT_PLAIN)
    @Consumes(javax.ws.rs.core.MediaType.APPLICATION_FORM_URLENCODED)
    public String getMessage() {
        return "Bonjour tout le monde !";
    }
    @GET
    @Path("getAuthor")
    @Produces(javax.ws.rs.core.MediaType.TEXT_PLAIN)
    @Consumes(javax.ws.rs.core.MediaType.APPLICATION_FORM_URLENCODED)
    public String getAuthor() {
```

```
        return "Fait par Gambera Luca !";
    }
```