Exercice 1

L'exercice 1 met en évidence la distinction entre une page en format .html et une page en format .jsp. Une page en format html est statique et est affichée directement dans le navigateur web. En revanche, une page en format jsp est d'abord traitée par un serveur TomCat, qui exécute le code Java présent dans la page afin de générer un contenu dynamique. Voici comment le code Java est intégré dans une page jsp.

Les commandes suivantes sont utiles pour voir quelles applications sont en cours d'exécution sur le port 8080 et pour arrêter ces applications si nécessaire :

- Permet de voir quelles connexions réseau sont actives sur le port 8080, y compris les applications qui l'utilisent.

```
netstat -ano | findstr :8080
```

- Permet d'arrêter l'application qui utilise le port 8080 en spécifiant le PID (Process ID) de l'application en question. Cela permet d'arrêter l'application de manière forcée si elle ne répond pas à une commande d'arrêt normale.

```
taskkill /PID 29648 /F
```

Exercice 2

Fichier jsp

```
<%--
    Document   : maJsp
    Created on : 23 mars 2023, 14:43:07
    Author     : LeonettiM
--%>

<%@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 Word from JSP!</h1>
    </body>
</html>
```

Résultat :

Hello World from JSP!

Exercice 3

Dans l'exercice, nous avons utilisé un index et un fichier JSP, tout comme dans l'exercice précédent. Cependant, contrairement au deuxième exercice, nous avons permis aux utilisateurs d'interagir avec la page. Pour permettre cette interaction dans la page d'index, nous avons créé un formulaire de type "post" qui appelle le fichier JSP correspondant.

```
<html>
    <head>
        <title>Login Html et check Jsp</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    </head>
    <body>
        <h1>Page de login </h1><br>
        <form method="post" action="checkLogin.jsp">  <!-- la page
checkLogin.jsp sera appelée lors du submit -->
            <label for="user">Votre nom d'utilisateur :</label>
            <input type="text" name="username" id="user" size="50"
maxlength="50" /><br>
            <label for="pass">Votre mot de passe :</label>
            <input type="password" name="password" id="pass" size="50"
maxlength="50" /><br>
            <input type="submit" value="Soumettre">
        </form>
    </body>
</html>
```

Lorsque l'on appuis sur le bouton on récupère un username et un password pour l'afficher grâce aux scriptlet tags.

Résultat :

Contrôle du login par JSP

Vote nom d'utilisateur est : a
Votre mot de passe est : a

Exercice 4

Cet exercice présente des similitudes avec l'exercice précédent, à la différence qu'il requiert l'utilisation d'une API. Nous avons encore une fois utilisé un index et un fichier JSP, mais cette fois-ci, nous avons fait appel à une API pour récupérer des données dynamiques et les afficher dans la page web.

Pour cela nous prenons de ce site l'exemple d'une ville.

```
<a href="https://www.prevision-meteo.ch/meteo/localite/fribourg-fr"><img
src="https://www.prevision-meteo.ch/uploads/widget/fribourg-fr_0.png"
width="650" height="250" /></a>
```

Mets à la place du « fribourg-fr » nous mettons ce que l'utilisateur à rentrer dans l'index.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
                <%
            String nomVille = request.getParameter("nomVille");


         %>
        <h1>Voici la météo de <%=nomVille%></h1>


<a href="https://www.prevision-
meteo.ch/meteo/localite/<%=nomVille%>"><img src="https://www.prevision-
meteo.ch/uploads/widget/<%=nomVille%>_0.png" width="650" height="250"
/></a>
    </body>
</html>
```

Voici le rendu final, la première image c'est quand nous arrivons sur le site et la 2ème page c'est lorsque nous avons entré une ville valide pour l'api dans l'input.

On récupère ce que l'on a mis dans notre textfield, et on va le mettre dans la requête qui va nous retourner la météo grâce à une API.

Résultat :

METEO: Sélection d'une ville

Indiquer un nom d'une ville suisse :
Soumettre

Météo

Exercice 5

Pendant cet exercice, nous avons affiché des villes qui étaient stockées dans une base de données. Pour cela, nous avons utilisé deux méthodes différentes : la première consistait à utiliser des instructions "print" pour afficher les noms des villes directement dans le code HTML généré par la page JSP. La seconde méthode consistait à stocker les noms de villes dans des éléments HTML "div" pour les afficher à l'écran.

```
<%
    out.println("Solution avec instruction java out.println()");
    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>
        <%  } }   %>
```

Résultat :

Ma requête SQL depuis JSP!

Liste des pays

Solution avec instruction java out.println()
Suisse
Autriche
France
Slovénie
Allemagne
Canada
USA
Norvège
Italie
Bulgarie
Suède
Solution avec intégration de variables java dans HTML
Suisse
Autriche
France
Slovénie
Allemagne
Canada
USA
Norvège
Italie
Bulgarie
Suède

Exercice 6

Cet exercice consistait à créer un système de connexion qui vérifie les informations saisies par l'utilisateur. Si les informations sont correctes, l'utilisateur est redirigé vers une page spécifique, tandis que s'il y a une erreur dans les informations de connexion, il est redirigé vers une autre page.

Ci-dessous nous pouvons voir le jsp qui nous permet vérifier les donné et rediriger vers les bonnes pages.

```
<body>
    <% String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");
    String motDePasse = request.getParameter("password");
    String remoteHost = request.getRemoteHost();
    String msg = "La connexion n'a pas réussi";
```

```
   if (nom.equals("Gambera") && motDePasse.equals("Pa$$w0rd") &&
prenom.equals("Luca") ) {
            Info.setNom(nom);
            Info.setPrenom(prenom);
            Info.setPassword(motDePasse);
    %>
    <jsp:forward page="maJspLogge.jsp"/>
    <%} else {
        Error.setHost(remoteHost);
        Error.setMsg(msg);
    %>
    <jsp:forward page="erreur.jsp"/>
    <%            }%>
</body>
```

Pour les beans info et Error il ne faut pas oublier d'implémenter « Serializable »

```
public class BeanError implements Serializable {
```

Pour la page html c'est comme les exercice précédent nous devons mettre le jsp vers le quelle il devra être redirigé.

```
<form method="post" action="maJspTraitement.jsp">
```

Pour la page jsp erreur et logged nous avons dû importer les beans comme le jsptraitement.

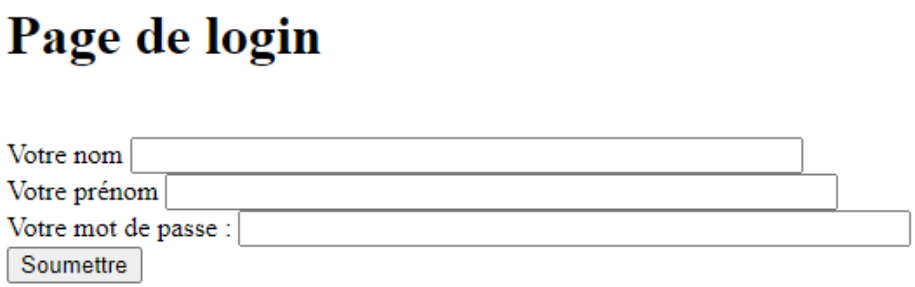
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="beans.BeanInfo"%>
<jsp:useBean id="Info"  scope="session" class="beans.BeanInfo"/>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Vous êtes loggé, voici les informations de
l'utilisateur!</h1>
        <p>
            Nom: <%=Info.getNom()%> <br>
            Prenom: <%=Info.getPrenom()%>
        </p>
    </body>
</html>
```

Voici le jspErreur

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="beans.BeanInfo"%>
<%@ page import="beans.BeanError"%>
```

```
<jsp:useBean id="Error"  scope="session" class="beans.BeanError"/>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>Erreur</title>
    </head>
    <body>
        <h1>Page d'erreur</h1>
        <p>
            Accès non autorisé pour hôte <%=Error.getHost()%>
        </p>
        <a href="index.html">
            Retour à la page de login
        </a>
    </body>
</html>
```

Résultat :



Exercice 7

Cet exercice a impliqué l'utilisation d'un Servlet, qui permet de générer dynamiquement du contenu HTML sur un serveur web. Nous avons repris l'exercice 3 et avons modifié l'action du formulaire pour pointer vers le Servlet que nous avons créé. Cela nous a permis d'utiliser le Servlet pour générer la page web dynamiquement, en récupérant les données à partir de la base de données et en les affichant dans la page HTML générée par le Servlet.

```
<form method="post" action="MaServlet">
```

Après avoir fait cela nous devons implémenter la méthode « processRequest » du Servlet. >Vous pouvez le voir ci-dessous.

```
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.
*/
        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 Servlet at " + request.getContextPath()
+ "</h1>");
        out.println("<p>Votre nom d'utilisateur est : " +
request.getParameter("username") + " et votre mot de passe est : " +
request.getParameter("password"));
        out.println("</body>");
        out.println("</html>");
    }
}
```

résultat :

Servlet MaServelet at /ServeletSimple

Votre nom d'utilisateur est : admin et votre mot de passe est : Pa$$s0rd

Exercice 8

Dans cet exercice, nous avons utilisé un Servlet pour vérifier les données saisies par l'utilisateur dans un formulaire similaire à l'exercice 6. Le formulaire contient des champs pour le nom d'utilisateur et le mot de passe. Dans le Servlet, nous avons utilisé la méthode "doPost". Tout d'abord, nous avons récupéré la session et configuré le temps d'inactivité. Ensuite, nous avons récupéré les données (nom d'utilisateur, mot de passe). Si les crédentials sont corrects, nous avons stocké le nom d'utilisateur dans la session comme attribut et redirigé l'utilisateur vers la page "logged". Si les crédentials sont incorrects, nous avons redirigé l'utilisateur vers la page d'erreur. Voici le code correspondant :

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    try ( PrintWriter out = response.getWriter()) {
        HttpSession session = request.getSession();
        session.setMaxInactiveInterval(20);
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        RequestDispatcher dispatch;
        if (username.equals("admin") && password.equals("Pa$$w0rd")) {
            BeanInfo info = new BeanInfo();
```

```
            info.setNom(username);
            info.setPassword(password);
            session.setAttribute("Info", info);
            dispatch = request.getRequestDispatcher("pageAutorise.jsp");
        } else {
            BeanError err = new BeanError();
            err.setHost(request.getRemoteHost());
            err.setMsg("mauvais identifiant");
            session.setAttribute("Error", err);
            dispatch = request.getRequestDispatcher("erreur.jsp");
        }
        dispatch.forward(request, response);
    }
}
```

Voici le code de si non avons réussi à nous logger

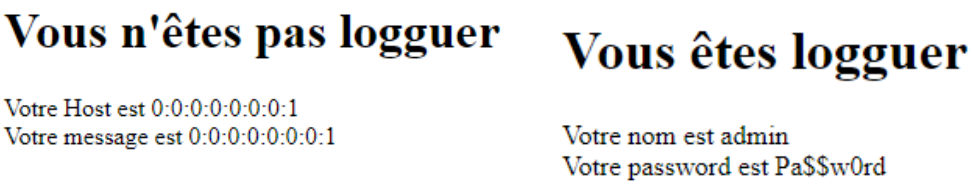
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="beans.BeanInfo"%>
<%@ page import="beans.BeanError"%>
<jsp:useBean id="Info"  scope="session" class="beans.BeanInfo"/>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Vous êtes logguer</h1>
        <p>
            Votre nom est <%=Info.getNom()%> <br>
            Votre password est <%=Info.getPassword()%><br>
        </p>
    </body>
</html>
```

Voici le code erreur

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="beans.BeanInfo"%>
<%@page import="beans.BeanError"%>
<jsp:useBean id="Error"  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 n'êtes pas logguer</h1>
        <p>
            Votre Host est <%=Error.getHost()%> <br>
            Votre message est <%=Error.getMsg()%><br>
        </p>
    </body>
</html>
```

Résultat:



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.

1.1.1 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.

```
<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.

```
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 !");
        }
    }
}
```

1.1.2 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 !";
    }
```