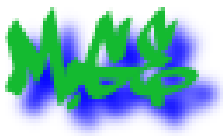


# 133 – Réaliser des applications Web en Session-Handling

## Rapport personnel

Version 2 du 22.06.2023

Malcolm Gfeller



Module du 23.03.2023 au 05.05.2023

# Table des matières

1	Introduction .....	1
2	Tests technos .....	1
2.1	TT1 Préparation Tomcat local .....	1
2.2	TT2 .....	1
2.3	TT3 .....	2
2.4	TT4 .....	2
2.5	TT5 .....	3
2.6	TT6 .....	5
2.7	TT7 .....	5
2.8	TT8 .....	6
2.9	TT10 .....	7
2.9.1	Partie cliente .....	7
2.9.2	Partie serveur .....	8
3	Conclusion .....	9

# 1 Introduction

Pendant ce module nous allons apprendre à réaliser et tester une application Web selon cahier des charges avec un langage de programmation.

Voici les objectifs opérationnels de ce module :

- 1- Analyser la donnée, projeter la fonctionnalité et déterminer le concept de la réalisation.
- 2- Réaliser une fonctionnalité spécifique d'une application Web par Session-Handling, authentification et vérification de formulaire.
- 3- Programmer une application Web à l'aide d'un langage de programmation compte tenu des exigences liées à la sécurité.
- 4- Vérifier la fonctionnalité et la sécurité de l'application Web à l'aide du plan tests, verbaliser les résultats et, le cas échéant, corriger les erreurs.

## 2 Tests technos

### 2.1 TT1 Préparation Tomcat local

Dans cet exercice j'ai préparé le serveur Tomcat pour mes futurs projets en suivant les étapes données dans le OneNote de la classe et j'ai appris à ajouter également dans NetBeans le serveur Tomcat.

**Voici des commandes utiles pour Tomcat :**

En cas de problème pour stopper Tomcat : lancez en mode CMD :

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

Prendre le PID du process qui tourne et lancez en CMD :

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

### 2.2 TT2

Pour cet exercice j'ai dû créer ma première page WEB en JSP. Pour faire cela, j'ai créé un nouveau projet NetBeans (**Java with Ant / Java Web / Web Application**). Une fois qu'on crée le projet, on reçoit également une page index.html auquel on peut modifier et quand on fait un « Run » dans NetBeans la page index est ouverte dans un navigateur. Ensuite j'ai créé également une page JSP en faisant un clic droit dans mon projet dans l'onglet New → JSP.

## 2.3 TT3

Pour le test techno 3 j'ai utilisé également comme le test 2 un index et un JSP, mais cette fois j'ai fait en sorte qu'ils puissent interagir. Pour cela j'ai fait un POST en appelant le JSP depuis mon index.

```
<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>
```

## 2.4 TT4

Pour le test techno 4 j'ai repris la même base de l'exercice précédent et j'ai laissé simplement un textfield pour insérer le nom de la ville souhaitée et j'ai modifié ensuite le JSP de la manière suivante :

```
<body>

  <%
    String nomVille = request.getParameter("nomVille");//récupération des
paramètres de la méthode POST dans index.html

  %>

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

  <!-- Insertion de code java -->


  <a href="https://www.prevision-meteo.ch/meteo/localite/<%=nomVille%>"></a>

</body>
```




## 2.5 TT5




Dans ce test technologique, l'objectif était d'afficher les villes provenant d'une base de données de deux manières différentes. Pour cela, il était nécessaire d'utiliser JDBC afin d'effectuer une requête dans la partie Java.


Pour faire cela, nous avons en premier créé une base de données, de la manière suivante :

 Bases de données

---

 phpMyAdmin
  Bases de données MySQL®
  Assistant de base de données MySQL®

 MySQL distant®
  Bases de données PostgreSQL
  Assistant de base de données PostgreSQL

 phpPgAdmin

---

### Créer une base de données

Nouvelle base de données :

[Créer une base de données](#)

Ensuite on a créé un utilisateur et on lui a mis des privilèges sur la base de données que je venais de créer.

Puis, dans la page JSP, nous avons utilisé l'appel au WrkDB qui nous permet de faire les requêtes à la base de données et de recevoir les villes.

```
<%
    WrkDB wrkDB = new WrkDB("3306","gfellerm01_kitzbuehl");
    ArrayList<String> lstPays = wrkDB.getPays();

    if (lstPays != null) {
%>
<h2>Liste des pays </h2>
<%
    out.write("Solution avec instruction java out.println()");
    for (String pays : lstPays) {
        out.write(pays + "<br>");
    }
%>
<div>Solution avec intégration de variables java dans HTML</div>
```

```
<%  
  
    for (String pays : lstPays) {  
  
        out.write("<p> " + pays + "</p>");  
  
    }  
  
%>
```

Voici ce que j'ai changé sur le WrkDb :

```
public boolean openConnexion() {  
  
    final String url = "jdbc:mysql://5.182.248.183:" + port + "/" + dbName +  
"?serverTimezone=CET";  
  
    final String user = "gfellerm01_admin";  
  
    final String pw = "CONFIDENTIAL";  
  
    boolean result = false;  
  
}
```

Voici le resultat de l'appel de la base de données :

## Ma requête SQL depuis JSP!

### Liste des pays

Solution avec instruction java out.println()

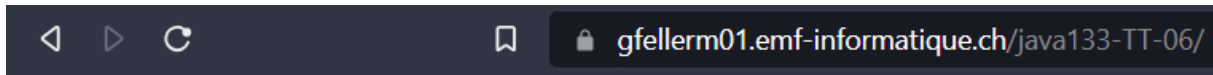
Suisse  
Autriche  
France  
Slovénie  
Allemagne  
Canada  
USA  
Norvège  
Italie  
Bulgarie  
Suède

Solution avec intégration de variables java dans HTML

Suisse  
  
Autriche  
  
France  
  
Slovénie  
  
Allemagne  
  
Canada  
  
USA  
  
Norvège  
  
Italie  
  
Bulgarie  
  
Suède

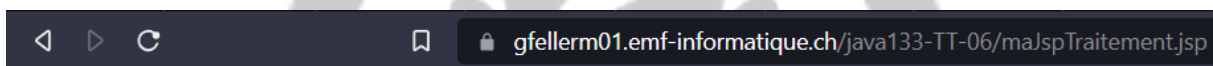
## 2.6 TT6

Pour ce test technologique nous avons dû mettre en place un login fonctionnel avec du java et du html. On a dû utiliser des bean pour transmettre des informations de connexion ou d'erreur. Il a fallu créer deux beans qui se nomment « BeanError » et « BeanInfo ». Un sert à afficher les infos de connexion (BeanInfo) et l'autre à afficher des erreurs (BeanError). Pour les deux beans il a fallu implémenter l'interface sérialisable afin de pouvoir les utiliser.



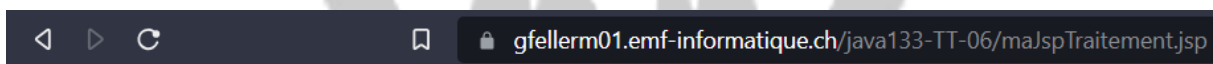
### Page de login

Votre nom   
Votre prénom   
Votre mot de passe :



### Vous êtes loggé, voici les informations de l'utilisateur !

Nom: Gfeller  
Prenom: Malcolm



### Page d'erreur

Accès non autorisé pour hôte 127.0.0.1

[Retour à la page de login](#)

## 2.7 TT7

L'objectif de ce test technologique était de créer un Servlet pour gérer une requête. Un Servlet permet de générer dynamiquement des données sur un serveur HTTP, qui seront ensuite présentées en HTML.

Pour commencer, j'ai repris la page HTML de l'exercice 3 et j'ai modifié l'action en remplaçant par le nom de la classe Servlet.

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

Ensuite, j'ai dû créer une classe Servlet, ça ne se crée pas comme une classe normale. J'ai dû sélectionner « servlet » pour créer une classe Servlet.

Ensuite, Il faut remplir la méthode "processRequest" où je crée dynamiquement une page HTML.

Voici le code du Servlet :

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {

    response.setContentType("text/html;charset=UTF-8");

    try (PrintWriter out = response.getWriter()) {

        out.println("<!DOCTYPE html>");

        out.println("<html>");

        out.println("<head>");

        out.println("<title>Servlet Servlet</title>");

        out.println("</head>");

        out.println("<body>");

        out.println("<h1>Servlet Servlet at " + request.getContextPath() +

"</h1>");

        out.println("<p>Username = " + request.getParameter("username") + "</p>");

        out.println("<p>Password = " + request.getParameter("password") + "</p>");

        out.println("</body>");

        out.println("</html>");

    }

}
```

Lorsque j'appuierai sur le bouton de la page HTML de base, il se produira ce qui suit : une page HTML sera créée à l'aide de ma classe Servlet, et j'afficherai "Servlet Servlet At" ainsi que les identifiants (IDs).

## 2.8 TT8

Pour ce test technologique, l'objectif était de réaliser une tâche similaire à celle du test technologique 6, qui consistait à vérifier les identifiants de connexion, mais cette fois-ci en utilisant un Servlet. Ensuite, il fallait utiliser des pages JSP pour afficher soit l'erreur, soit la réussite de la connexion.

Tout comme dans le test technologique 3, une page HTML a été créée avec un formulaire où les identifiants doivent être saisis. Dans la classe Servlet, la méthode "doPost" a été utilisée. Elle permet de récupérer la session et de définir une durée maximale d'inactivité (exprimée en secondes). Ensuite, nous avons extrait le nom d'utilisateur et le mot de passe saisis et vérifié s'ils correspondent à ceux présents dans le code. Si les identifiants sont corrects, nous avons attribué un nouveau nom et défini cet attribut en utilisant son identifiant sur la page JSP "maJspLogge.jsp". Ensuite, nous avons chargé cette page JSP à l'aide d'un RequestDispatcher.

Si les identifiants ne sont pas valides, nous avons chargé la page JSP "erreur.jsp".



## 2.9 TT10

Pour cet exercice, on a dû créer une application cliente et une application serveur pour accéder à différentes ressources qu'on a sur le serveur.

### 2.9.1 Partie cliente

On a commencé par faire deux boutons dans une page html qui va ensuite nous rendre la ressource qu'on a demandé.

```
body>

  <div>Requêtes tutoriels</div>

  <form method="post" action="ServletCtrl">

    <input type="submit" name="getMessage" value="Récupérer message"/>

    <input type="submit" name="malcolm" value="Récupérer auteur"/>

  </form>

</body>
```

Ensuite, nous avons créé deux classes, dont l'une est un Servlet. La classe ClientRest est responsable de contacter le serveur afin de récupérer les ressources demandées en fonction de ce qui a été cliqué.

```
private WebTarget webTarget;

private Client client;

private static final String BASE_URI =
"http://localhost:8080/javaRestFull/websource";

public ClientRest() {

    client = javax.ws.rs.client.ClientBuilder.newClient();

    webTarget = client.target(BASE_URI).path("app");

}

@Override

public String hello() {

    WebTarget resource = webTarget;

    return
resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);

}

@Override

public String myStatus() {

    WebTarget resource = webTarget;

    resource = resource.path("malcolm");

}
```

```

    return
    resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
}

```

La classe Servlet, on a dû faire un if pour voir si l'action était « malcolm », si c'était le cas, on retourne « Malcolm est actif », si ce n'est pas le cas "hello world".

```

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

    try (PrintWriter out = response.getWriter()){

        ClientRest clientRest = new ClientRest();

        if (request.getParameter("malcolm") != null){

            out.println(clientRest.myStatus());

        }else{

            out.println(clientRest.hello());

        }

        clientRest.close();

    }catch (Exception e){

        e.printStackTrace();

    }

}

```

### 2.9.2 Partie serveur

Dans le serveur, nous avons créé deux classes. L'une d'entre elles s'appelle HelloApplication et son chemin d'accès commence par "/webservice". Cela nous permettra de rediriger vers les ressources appropriées à partir de cette classe.

```

@ApplicationPath("/webservice")
public class HelloApplication extends Application {

}

```

Pour la classe HelloResources, nous avons dû ajouter d'autres chemins. Si nous accédons à "/webservice/app", cela nous renverra "hello world", tandis que si nous accédons à "/webservice/app/malcolm", nous obtiendrons "Malcolm est actif".

```

@Path("/app")
public class HelloResource {

    @GET

    @Produces("text/plain")

    public String hello() {

        return "Hello, World!";

    }

}

```

```
@GET
@Path("/malcolm")
@Produces("text/plain;charset=UTF-8")
public String myStatus(){
    return "Malcolm est actif";
}
}
```

### 3 Conclusion

J'ai bien aimé ce module, car il m'a permis d'apprendre comment faire une application web en java avec Tomcat. On a dû faire une application avec 2 clients, 1 gateway et 2 servlets. C'était un module intéressant, mais malheureusement j'ai accumulé du retard à cause d'autres projets que j'avais à faire et j'ai eu un peu de peine.