TP5 – Introduction au serveur d'applications Tomcat, aux API Jakarta HttpServlet, HttpUrlConnection et XmlHttpRequest

Avant-Propos

Ouvrez deux terminaux

Dans le premier terminal, lancez un **conteneur Docker `r401-runtime-container`** conformément au **TP3** en suivant la configuration de votre machine.

Dans le second terminal, lancez :

- `docker image pull` **`r401-tomcat`**
- `docker run -it --name` **`r401-tomcat-container`** `-p 8081:8080` **`r401-tomcat`**

Dans un navigateur, ouvrez http://localhost:8081/0-WebTest/test et normalement vous devriez voir que « **Ça marche…** ».

> Si vous n'êtes pas sur une machine fixe de l'IUT, n'oubliez pas de changer le nom de l'**image Docker `r401-tomcat`** en **`fripouillon/r401-tomcat`**. Attention de ne pas changer le nom du **conteneur Docker `r401-tomcat-container`**.

Au cours de ce TP, vous allez donc utiliser **deux conteneurs Docker**.

Exercice 1 : API Jakarta HttpServlet
(Java)

Préambule :

Tomcat est un serveur d'applications issu du projet **Jakarta**. C'est un des nombreux projets de l'**Apache Software Foundation**. Il implémente notamment les spécifications des **Servlets** (classes Java qui permettent de créer dynamiquement des données au sein d'un serveur HTTP), des **WebSockets** (protocole permettant d'ouvrir un canal de communication bidirectionnel encapsulé dans des classes Java), de **JSP** (pages web embarquant du code dynamique à la manière de PHP mais en Java) et intègre un serveur HTTP.

Les **Servlets** permettent donc de générer du contenu dynamique au sein d'un serveur HTTP à l'instar de PHP mais en Java.

Imaginons que nous voulions développer une application web **MonProjet**, au sein de laquelle nous souhaitons accéder à un contenu dynamique depuis l'adresse **moncontenu** à laquelle on passerait un paramètre de type **cle=valeur**. L'URL du contenu dynamique sur la machine locale aurait alors une forme du type :

localhost:8080/MonProjet/moncontenu?cle=valeur…

Comment l'implémenter ? Il faut d'abord créer un fichier **MyServlet.java** dans lequel la classe en charge de la génération du contenu dynamique hériterait de la classe **HttpServlet**. En tête du fichier **MyServlet.java**, il faut spécifier le chemin d'accès au contenu en annotant la classe :

```
@WebServlet(
   urlPatterns = { "/moncontenu" }, asyncSupported = true
)
public class MyServlet extends HttpServlet {
   @Override
   void doGet(HttpServletRequest _request, HttpServletResponse _response) {
   }
}
```

En terme de développement, pour répondre à une requête **GET**, il suffit simplement de surcharger la méthode **doGet()**. La méthode prend en arguments une requête de type **HttpServletRequest** et une réponse de type **HttpServletResponse**. La requête permet de récupérer les paramètres passés dans l'adresse tels que **cle=valeur**, tandis que la réponse sert de réceptacle pour générer le contenu dynamique. Pour répondre à une requête **POST**, il suffirait de procéder de la même manière et de surcharger la méthode **doPost()**.

La documentation des **Servlets** est sur documentation HttpServlet.

Exercice :

Le but est d'implémenter une **Servlet** générant dynamiquement un message de bienvenue.

Créez un fichier :

- **MyServletHello.java** (contenant la classe encapsulant la **Servlet**) relié à l'adresse localhost:8081/3-ServletAPI/hello qui retournera alors simplement la chaine « **Hello World** » ou « **Hello Luc** » si par exemple on passe **?name=Luc** en paramètre

dans le dossier **/workspace/3-ServletAPI/src** du **conteneur Docker `r401-tomcat-container`**.

Pensez à récupérer les paramètres sur la requête et à utiliser un **PrintWriter** sur la réponse pour générer le contenu dynamique.

La documentation des requêtes est sur documentation HttpServletRequest.
La documentation des réponses est sur documentation HttpServletResponse.

Pour compiler et déployer votre code, dans le **terminal Docker de `r401-tomcat-container`**, lancez :

- `deploy.sh 3-ServletAPI`

Pour tester votre code, dans un **navigateur**, ouvrez :

- http://localhost:8081/3-ServletAPI/hello
- http://localhost:8081/3-ServletAPI/hello?name=toto

Exercice 2 : API HttpUrlConnection (Java)

Préambule :

Lorsqu'une application web est créée, on peut vouloir récupérer du contenu dynamique hors du navigateur, dans une application native par exemple.

Le langage Java permet d'envoyer une requête à un serveur et de récupérer le contenu grâce à la classe **HttpUrlConnection** dont la documentation est sur documentation HttpUrlConnection. Après la requête, le contenu peut être lu dans un flux d'entrée de type **InputStream** qui gère la réception asynchrone.

Exercice :

Le but est d'implémenter une application en ligne de commande pour afficher le message de la **Servlet**.

Créez un fichier :

- **MyApplication.java** (contenant la classe d'entrée) afin d'envoyer une requête à votre précédente **Servlet** et d'afficher le message correspondant

dans le dossier **/workspace/4-RemoteAPI/src** du **conteneur Docker `r401-runtime-container`**.

Pour compiler et exécuter votre code, dans le **terminal Docker de `r401-runtime-container`**, lancez :

- `run.sh 4-RemoteAPI`

Exercice 3 : API XmlHttpRequest
(JavaScript)

Préambule :

Lorsqu'une application web est créée, on peut vouloir récupérer du contenu dynamique pour enrichir un élément

Le langage JavaScript permet d'envoyer une requête à un serveur et de récupérer le contenu grâce à l'objet **XMLHttpRequest** dont la documentation est sur documentation XMLHttpRequest. Après la requête, le contenu peut être lu suite à un événement capté par **onreadystatechange**.

Exercice :

Le but est d'implémenter une application web interactive pour afficher le message de la **Servlet** dans une page HTML suite à l'appui sur un bouton.

Créez des fichiers :

- **myapplication.html** (contenant l'interface de la page) constitué :
  - d'un bouton **<button>** dont l'appui appellera une fonction **handleSayHello()**
  - d'un ** ** vide avec l'identifiant unique **m_message**

dans le dossier **/workspace/5-HttpRequestAPI/www** du **conteneur Docker `r401-tomcat-container`**

- **hello.js** avec :
  - la fonction **handleSayHello()** chargée d'envoyer une requête à l'adresse http://localhost:8081/0-WebTest/test
  - une fonction **sayHello()** chargée de répondre à l'événement capté par **onreadystatechange** en affectant le message reçu à **m_message**.

dans le dossier **/workspace/5-HttpRequestAPI/scripts** du **conteneur Docker `r401-tomcat-container`**.

Pour compiler et déployer votre code, dans le **terminal Docker de `r401-tomcat-container`**, lancez :

- `deploy.sh 5-HttpRequestAPI`

Pour tester votre code, dans un **navigateur**, ouvrez :