UNIVERSITE DE RENNES 1	**Des Servlets à la notion de service Web**	istic Informatique Électronique

Les objectifs de ce travail pratique sont les suivants :

- Comprendre les mécanismes des Servlet
- Réaliser une application Web en utilisant Combinant JPA et les Servlet
- Comprendre les principes d'une architecture Rest
- Comprendre les bénéfices d'un framework comme Jersey

Recommandations. Ce TP utilise des technologies abordées en cours d'un point de vue théorique, mais la documentation technique peut être facilement trouvée sur internet.

Être autodidacte est une compétence essentielle pour tout informaticien; n'hésitez pas à chercher des tutoriels si vous êtes bloqués.

Sujet

L'objectif de ce projet est de continuer le développement d'une application type kanban.

Organisation :

Quand votre application JPA fonctionne correctement. Laissez votre base de données démarrée. Si vous avez déjà travaillé avec les servlets, n'hésitez pas à aller directement à la question 6.

Partie 1 Servlet

Question 1.

Tout d'abord, modifiez votre fichier pom.xml.

a. Changez le type de packaging vers un packaging de type war pour les applications webs

```
<packaging>war</packaging>
```

b. Ajoutez une dépendance à l'API des servlets

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.0.1</version>
         <scope>provided</scope>
</dependency>
```

Vous noterez le scope qui est placé à *provided* ce qui veut dire que cette librairie sera fournie par le conteneur d'application et ne doit pas être embarquée au sein de l'application Web.

c. Ajoutez enfin dans les plugins de build, celui de jetty qui permet de démarrer jetty depuis maven.

```
<plugin>
<!--
https://mvnrepository.com/artifact/org.eclipse.jetty/jetty-maven-plugin -->
<groupId>org.eclipse.jetty</groupId>
<artifactId>jetty-maven-plugin</artifactId>
<version>9.4.31.v20200723</version>
<configuration>
<webApp>
<contextPath>/</contextPath>
</webApp>
<httpConnector>
<port>8080</port>
</httpConnector>
</configuration>
</plugin>
```

Testez votre config. Clic droit sur votre projet. run as -> maven build …-> mettre **compile jetty:run** dans le goal. équivalent de lancer compile jetty:run si vous utilisez jetty dans la console. Vous pouvez stopper le jetty en cliquant sur le bouton rouge stop de la console eclipse.

Question 2. Insertion de ressources statiques

Créer un répertoire src/main/webapp.

Ajoutez-y un fichier index.html contenant Hello world.

Relancez jetty.
http://localhost:8080/index.html

Vous devriez voir votre page web. L'ensemble des ressources statiques (html files, javascript file, images) de votre projet doivent se trouver dans le répertoire src/main/webapp

Question 3. Création de votre première Servlet

Créer une classe qui étend HttpServlet. Surchargez les méthodes doGet et doPost qui seront appelées lors de la réception d'un GET et d'un POST sur l'url "/myurl".

```
package servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet(name="mytest",
urlPatterns={"/myurl"})
public class MyServlet extends HttpServlet {

	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {

		PrintWriter p = new PrintWriter(resp.getOutputStream());
		p.print("Hello world");
		p.flush();

	}

	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {
		// TODO Auto-generated method stub
		super.doPost(req, resp);
	}
}
```

Question 4. Création de votre première Servlet qui consomme les données d'un formulaire.

Créer un fichier myform.html et placez y le code suivant.

```
<html>
	<body>
		<FORM Method="POST" Action="/UserInfo">
			Name :		<INPUT type="text" size="20" name="name"><BR>
			Firstname :	<INPUT type="text" size="20" name='firstname'><BR>
			Age :		<INPUT type="text" size="2" name='age'><BR>
					<INPUT type="submit" value="Send">
		</FORM>
	</body>
</html>
```

Puis créer une classe Java UserInfo

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name="userinfo",
urlPatterns={"/UserInfo"})
public class UserInfo extends HttpServlet {
public void doPost(HttpServletRequest request,
				HttpServletResponse response)
	 throws ServletException, IOException {
	response.setContentType("text/html");

	PrintWriter out = response.getWriter();
```

```
        out.println("<HTML>\n<BODY>\n" +
                            "<H1>Recapitulatif des informations</H1>\n" +
                            "<UL>\n" +
                " <LI>Nom: "
                            + request.getParameter("name") + "\n" +
                            " <LI>Prenom: "
                            + request.getParameter("firstname") + "\n" +
                            " <LI>Age: "
                            + request.getParameter("age") + "\n" +
                            "</UL>\n" +
                "</BODY></HTML>");
}
}
```

Testez votre application en vous rendant ici
http://localhost:8080/myform.html

Parcourez ensuite les exemples suivants.
http://www.commentcamarche.net/contents/servlets-330845945

Question 5. Retour sur l'application de tableau Kanban

À partir de cette base, construisez une page web qui retourne des informations issues de la base de données et un formulaire qui permet d'ajouter des éléments dans la base de données.

Partie 2 JaxRS et OpenAPI

Question 6. En avant pour les architectures Rest.

Évidemment à partir de là, nous pourrions construire une architecture Rest manuellement. Cependant, gérer toutes les routes et les formats de sérialisation de données à la main est une tâche fastidieuse.
Pour cela des frameworks comme Jersey et les APIs JaxRS permettent de simplifier ce travail.

Vue d'ensemble de JAX-RS

JAX-RS est une API récente mais déjà bien fournie : Elle propose notamment des annotations spécifiques pour lier une URI et des verbes HTTP à des méthodes d'une classe Java. JAX-RS dispose également d'annotations capables d'injecter et de récupérer des données dans les entêtes d'une réponse ou depuis celles d'une requête. De même l'API JAX-RS fournit ce qu'il faut pour extraire et écrire ce que vous voulez dans le corps d'une requête/réponse HTTP. Enfin, en cas d'exception Java, JAX-RS est capable de produire une réponse avec le code HTTP le plus pertinent.

Pour simplifier l'installation d'un environnement, je vous ai fourni un exemple fonctionnel avec JaxRS, et RestEasy.
RESTEasy est un projet JBoss / Red Hat qui fournit divers frameworks pour vous aider à construire des services Web RESTful et des applications Java RESTful. Il s'agit d'une implémentation des RESTful Web Services de Jakarta (JaxRS), une spécification de la

fondation Eclipse qui fournit une API Java pour les RESTful Web Services sur le protocole HTTP. De plus, RESTEasy implémente également l'API de la spécification MicroProfile REST Client.

https://github.com/barais/JaxRSOpenAPI

Forker ce projet et cloner le fork pour pouvoir travailler. Vous pourrez intégrer vos classes JPA et vos DAOs facilement.

Observer le code et créer un premier service Web.
Tester le service implanter l'aide de votre navigateur en vous rendant sur http://localhost:8080/pet/1

Créer quelques services REST simples et tester votre couche de service avec Postman. https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop

Question 7. Créez la couche de service pour votre application.

Écrire des exemples de WS REST correspondant à votre projet d'application de tableau Kanban
Examples: http://www.mkyong.com/tutorials/jax-rs-tutorials/

Vous pouvez la tester grâce à des outils comme:
https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop

Il faut ajouter l'extension Postman rest client sur Mozilla ou Chrome.

Question 8.

Comprendre openAPI. Dans le readme.md du projet ici.
https://github.com/barais/JaxRSOpenAPI
Je montre comment ajouter la génération automatique du descripteur d'API conforme au standard openAPI.
Je montre aussi comment intégrer SwaggerUI afin de fournir une documentation lisible par un humain de votre API.

Nous le faisons à la main, dans les comme SpringBoot, des modules particuliers permettent d'automatiser ce travail.

Bon TP

Olivier et Adrien