UNIVERSITE DE RENNES 1	**Master 1** **SIR**	istic Informatique Électronique

à la découverte de la notion de Servlet

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

Recommendations. Ce TP utilise des technologies abordées en cours d'un point de vu 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 doodle permettant la prise de RDV avec des services annexes supplémentaires.

Organisation.

Quand votre application JPA fonctionne correctement. Laissez votre base de données démarrée.

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 dépendances 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 embarqué au sein de l'application Web.

c. Ajoutez enfin dans les plugin de build, celui de tomcat qui permet de démarrer tomcat depuis maven.

```
<build>
		<plugins>
			<!-- * Start of user code for plugins -->
			<plugin>
				<groupId>org.apache.tomcat.maven</groupId>
				<artifactId>tomcat7-maven-plugin</artifactId>
				<version>2.2</version>
				<configuration>
					<path>/</path>
				</configuration>
			</plugin>
			....
		...
	</build>
```

Testez votre config. Clic droit sur votre projet. run as -> maven build …-> mettre tomcat7:run dans le goal. équivalent de lancer mvn tomcat7:run dans la console. Vous pouvez stoppez tomcat 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 Sir.

Relancez tomcat.
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 méthodes 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 SIR");
        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 prise de RDV type doodle

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

Question 6. En avant pour les architectures Rest.

Évidemment à partir de là, nous pourrions construire une architecture Rest manuellement. Cependant, gérez toutes les routes et les format 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.

Jersey

Jersey est un framework open-source écrit en langage Java permettant de développer des services web selon l'architecture REST suivant les spécifications de JAX-RS (JSR 311 et JSR 339).
Jersey a été développé par Oracle. Il est distribué sous la double licence CDDL (Common Development and Distribution License) et GPL version 2 (General Public License).
Jersey supporte plusieurs formats parmi lesquels Atom, JSON et MIME Multipart data.

a. Pour construire une application avec Jersey, il faut tout d'abord modifiez le fichier pom.xml pour ajouter la dépendance à Jersey et celle de jersey-json.

```
<dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-servlet</artifactId>
        <version>1.18.3</version>
</dependency>
```

```
<dependency>
	<groupId>com.sun.jersey</groupId>
	<artifactId>jersey-json</artifactId>
	<version>1.18.3</version>
</dependency>
```

b. Ajoutez ensuite le descripteur d'applications web c'est à dire le fichier web.xml dans le répertoire src/main/webapp/WEB-INF/. Ce fichier a pour rôle de configurez les servlets. (Ce que l'on a fait par annotation précédemment).

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns="http://java.sun.com/xml/ns/javaee"
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
	id="WebApp_ID" version="3.0">
	<display-name>MyWebProject</display-name>
	<servlet>
		<servlet-name>Jersey Web Application</servlet-name>
		<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
		<init-param>
			<param-name>com.sun.jersey.config.property.packages</param-
name>
			<param-value>
				fr.istic.sir.rest
			 </param-value>
		</init-param>
				<init-param>
			<param-name>com.sun.jersey.api.json.POJOMappingFeature</param-
name>
			<param-value>true</param-value>
		</init-param>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>Jersey Web Application</servlet-name>
		<url-pattern>/rest/*</url-pattern>
	</servlet-mapping>
</web-app>
```

Vous noterez que dans ce fichier, vous configurez le package dans lequel jersey ira chercher les classes annotées avec l'API JAX-RS (ici fr.istic.sir.rest) et l'URL à partir de laquelle jersey prend la main ici */rest*

c. Créez le package Java fr.istic.sir.rest et placez y la classe suivante.

```
package fr.istic.sir.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
```

```
public class SampleWebService {
	@GET
	@Produces(MediaType.TEXT_PLAIN)
	public String sayHello() {
		return "Hello, how are you?";
	}
}
```

d. Relancez tomcat à l'aide de maven. et rendez vous ici http://localhost:8080/rest/hello

e. Ajoutez la méthode suivante à adapter par rapport à votre classe **Home (ici "Home" est à remplacer par le nom de la classe utilisée pour votre projet type doodle)**

```
@GET
@Path("/home")
@Produces(MediaType.APPLICATION_JSON)
public Home getHome() {
	Home h = new Home();
	h.setName("toto");
	Heater h1 = new Heater();
	h1.setPower("500w");
	Heater h2 = new Heater();
	h2.setPower("600w");
	h.addDevice(h1);
	h.addDevice(h2);
	return h;
}
```

RDV à l'URL http://localhost:8080/rest/hello/home pour tester.

Observez cette classe pour voir les mapping JSON disponibles.

https://github.com/barais/taa_angularjs_jersey/blob/master/src/main/java/com/github/trecloux/yeoman/BookResource.java

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

Ecrire des exemples de WS REST correspondant au projet type doodle
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.

Bon TP

Mohammed, June et Olivier