**TP : IMPLÉMENTATION DES SERVICES AVEC JERSEY.**

Nous allons commencer par développer une API simple sans persistance pour gérer une ToDoList. L’API va permettre d’afficher les todo, d’en ajouter et d’une supprimer.

Commençons par créer le projet maven avec le pom suivant :

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.formation</groupId>

<artifactId>todo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>todo</name>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<junit.version>5.7.0</junit.version>

</properties>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

<version>2.12.2</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.inject</groupId>

<artifactId>jersey-hk2</artifactId>

<version>2.12.2</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet</artifactId>

<version>3.0.2</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

<version>3.0.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.inject</groupId>

<artifactId>jersey-hk2</artifactId>

<version>3.0.2</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-grizzly2-http</artifactId>

<version>3.0.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>3.3.0</version>

</plugin>

</plugins>

</build>

</project>

Créer un bean Java Todo avec un id et une description dans un package model:

**public class** Todo{

**private** **int** id;

**private** String description;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

}

Dans un package controler, créer une classe TodoResource avec une route pour ajouter, supprimer un todo et lister tous les todo :

@Path("/todo")

**public** **class** TodoResource {

**private** List<Todo> todos = **new** ArrayList<>();

@GET

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Response all() {

**return** Response.*status*(Response.Status.***OK***).entity(todos).build();

}

@Path("/{id}")

@GET

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Todo hello(@PathParam("id") Integer id) {

**return** todos.get(id);

}

@Path("/create")

@POST

@Consumes(MediaType.***APPLICATION\_JSON***)

@Produces(MediaType.***APPLICATION\_JSON***)

**public** Response create(Todo todo) {

todos.add(todo);

**return** Response.*status*(Response.Status.***CREATED***).entity(todo).build();

}

}

Ajoutons une classe avec un Main pour exécuter notre service sur un serveur embarqué Grizzly :

**public** **class** MainApp {

**public** **static** **final** URI ***BASE\_URI*** = URI.*create*("http://localhost:8084");

// Starts Grizzly HTTP server

**public** **static** HttpServer startHttpServer() {

**final** ResourceConfig config = **new** ResourceConfig();

config.register(TodoResource.**class**);

**return** GrizzlyHttpServerFactory.*createHttpServer*(***BASE\_URI***, config);

}

**public** **static** **void** main(String[] args) {

**try** {

**final** HttpServer server = *startHttpServer*();

server.start();

// shut down hook

Runtime.*getRuntime*().addShutdownHook(**new** Thread(server::shutdownNow));

System.***out***.println(

String.*format*("Application started.%n"+***BASE\_URI***+"%nStop the application using CTRL+C"));

// block and wait shut down signal, like CTRL+C

Thread.*currentThread*().join();

} **catch** (InterruptedException | IOException ex) {

Logger.*getLogger*(MainApp.**class**.getName()).log(Level.***SEVERE***, **null**, ex);

}

}

}

Lancer votre projet et tester le.