

Compte rendu du TP 2 et 3

JPA / Servlet

Système d'Information Réparti

Contenu

TP2 : JPA	3
Question 1 et 2 : Transformation des classes en entité.	3
Classe Person	3
Class Home	5
Question 3 : Peuplement de la base et requête.....	10
Class EntityManagerHelper	10
Class JpaTest	12
Affichage des résultats	16
Résultat des requêtes de la classe JpaTest.....	17
Question 4 : Connexion à une base MySQL.....	18
Ajout dans le fichier Persistence.xml.....	18
Question 5 : Héritage.....	18
Class SmartDevice.....	19
Class Heater	21
Class ElectronicDevice	22
Question 6 : Mise en évidence du problème de n+1 select	23
N1Select.....	23
JoinFetch.....	23
TP3 : Servlet.....	24
Question 1 : Configuration du POM.xml	24
Question 2. Insertion de ressources statiques	25
Question 3. Création de votre première Servlet	25
Classe MyServlet.....	25
Question 4. Création de votre première Servlet qui consomme les données d'un formulaire.	26
Question 5. Retour sur OPOWER	27
Question 6. En avant pour les architectures Rest.	30
Question 7. Créez la couche de service pour votre application.	30

TP2 : JPA

Question 1 et 2 : Transformation des classes en entité.

Les classes sont contenues dans le package *domain*. Après avoir importé les packages correspondants depuis `javax.Persistence`, On renseigne les annotations suivantes :

- `@entity` : indique qu'une classe est une entité
- `@Id` : désigne l'attribut clé primaire de l'entité
- `@GeneratedValue` : indique que la clé primaire sera générée automatiquement par le SGBD
- `@ManyToMany` : Représente une table association
- `@OneToMany` : Représente une clé étrangère dans la table qui correspond au côté propriétaire (obligatoirement le côté « Many »)

Classe Person

```
package domain;
```

```
import java.util.List;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.ManyToMany;
```

```
@Entity
```

```
public class Person {
```

```
    private long id;
```

```
    private String name;
```

```
    private String firstname;
```

```
    private String mail;
```

```
private List<Home> homes;
```

```
public Person(long id, String name, String firstname, String mail, List<Home> homes) {  
    super();  
    this.id = id;  
    this.name = name;  
    this.firstname = firstname;  
    this.mail = mail;  
    this.homes = homes;  
}
```

```
public Person() {  
    super();  
}
```

```
@Id
```

```
@GeneratedValue
```

```
public long getId() {  
    return id;  
}
```

```
public void setId(long id) {  
    this.id = id;  
}
```

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getMail() {
        return mail;
    }

    public void setMail(String mail) {
        this.mail = mail;
    }

    @ManyToMany
    public List<Home> getHomes() {
        return homes;
    }

    public void setHomes(List<Home> homes) {
        this.homes = homes;
    }

    @Override
    public String toString() {
        return "Person [id=" + id + ", name=" + name + "]";
    }
}

```

Class Home

package domain;

```
import java.util.ArrayList;

import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

import javax.persistence.OneToMany;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlTransient;


import org.codehaus.jackson.annotate.JsonIgnore;
```

```
@Entity
```

```
@XmlRootElement
```

```
public class Home {
```

```
    private long id;
```

```
    private String name;
```

```
    private int size;
```

```
    private int nbRoom;
```

```
    private List<SmartDevice> heaters;
```

```
    private List<SmartDevice> electronicDevices;
```

```
    @JsonIgnore
```

```
    @XmlTransient
```

```
    private List<Person> people;
```

```
    public Home() {
```

```
        super();

        this.heaters = new ArrayList<SmartDevice>();

        this.electronicDevices = new ArrayList<SmartDevice>();
    }
}
```

```
public Home( String name, int size, int nbRoom, List<SmartDevice> heaters,
            List<SmartDevice> electronicDevices, List<Person> people) {

    super();

    this.name = name;

    this.size = size;

    this.nbRoom = nbRoom;

    this.heaters = heaters;

    this.electronicDevices = electronicDevices;

    this.people = people;
}
}
```

@Id

@GeneratedValue

```
public long getId() {

    return id;

}
}
```

```
public void setId(long id) {

    this.id = id;

}
}
```

```
public int getSize() {

    return size;

}
```

```

    }

    public void setSize(int size) {

        this.size = size;

    }

    public int getNbRoom() {

        return nbRoom;

    }

    public void setNbRoom(int nbRoom) {

        this.nbRoom = nbRoom;

    }


    @OneToMany(mappedBy="home",cascade=CascadeType.PERSIST)

    public List<SmartDevice> getHeaters() {

        return heaters;

    }

    public void setHeaters(List<SmartDevice> heaters) {

        this.heaters = heaters;

    }


    @OneToMany(mappedBy="home",cascade=CascadeType.PERSIST)

    public List<SmartDevice> getElectronicDevices() {

        return electronicDevices;

    }

    public void setElectronicDevices(List<SmartDevice> electronicDevices) {

        this.electronicDevices = electronicDevices;

    }


    public void addDevice(SmartDevice device){

        //Test s'il s'agit d'un chauffage

```



```

        if (device instanceof Heater){

            heaters.add(device);

        }else if (device instanceof ElectronicDevice){

            electronicDevices.add(device);

        }

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    @JsonIgnore

    @XmlTransient

    @ManyToMany(mappedBy="homes")

    public List<Person> getPeople() {

        return people;

    }

    public void setPeople(List<Person> people) {

        this.people = people;

    }

    @Override

    public String toString() {

        return "Home [id=" + id + ", name=" + name + "];"

    }

```

```
}
```

Question 3 : Peuplement de la base et requête

Nos classes sont contenues dans le package *jpa*.

Nous utilisons une classe *EntityManagerHelper* pour gérer notre *EntityManager* avec un thread local. Nous utilisons un pattern *singleton* pour garantir l'unicité de l'*EntityManager*.

Nous effectuons le peuplement de la base avec la classe *JpaTest*.

Class *EntityManagerHelper*

```
package jpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EntityManagerHelper {

    private static final EntityManagerFactory emf;
    private static final ThreadLocal<EntityManager> threadLocal;

    static {
        emf = Persistence.createEntityManagerFactory("dev");
        threadLocal = new ThreadLocal<EntityManager>();
    }

    public static EntityManager getEntityManager() {
        EntityManager em = threadLocal.get();

        if (em == null) {
```

```
        em = emf.createEntityManager();

        threadLocal.set(em);
    }

    return em;
}
```

```
public static EntityManagerFactory getEmf() {

    return emf;

}
```

```
    public static void closeEntityManager() {

        EntityManager em = threadLocal.get();

        if (em != null) {

            em.close();

            threadLocal.set(null);

        }

    }
```

```
public static void closeEntityManagerFactory() {

    emf.close();

}
```

```
public static void beginTransaction() {

    getEntityManager().getTransaction().begin();

}
```

```
public static void rollback() {

    getEntityManager().getTransaction().rollback();

}
```

```
public static void commit() {  
    getEntityManager().getTransaction().commit();  
}  
}
```

Class JpaTest

```
package jpa;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.EntityTransaction;  
import javax.persistence.Persistence;  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
  
import domain.ElectronicDevice;  
import domain.Heater;  
import domain.Home;  
import domain.Person;  
import domain.SmartDevice;  
  
public class JpaTest {
```

```
public JpaTest(EntityManager entityManager) {  
}
```

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

```
    EntityManagerHelper.getEntityManager().getTransaction();
```

```
    tx.begin();
```

```
    try {
```

```
        //Peuplement de la base
```

```
        createPerson();
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    tx.commit();
```

```
    //Requete sur la base
```

```
    listPerson();
```

```
    EntityManagerHelper.getEntityManager().close();
```

```
    EntityManagerHelper.getEmf().close();
```

```
}
```

```
private static void createPerson() {
```

```
    int numOfPeople = EntityManagerHelper.getEntityManager().createQuery("Select p From  
Person p", Person.class).getResultList().size();
```

```
    if (numOfPeople == 0) {
```

```
        //Creation des maisons
```

```
        Home home0 = new Home("résidence principale",90, 6, null,null,null);
```

```
        List<Home> homes = new ArrayList<Home>();
```

```
homes.add(home0);

//Creation des chauffages

Heater heater0=new Heater("Chauffage Salon", 10, home0);

Heater heater1=new Heater("Chauffage Salle à manger", 10, home0);

Heater heater2=new Heater("Chauffage Salle de bain", 10, home0);

Heater heater3=new Heater("Chauffage Chambre 1", 10, home0);

Heater heater4=new Heater("Chauffage Chambre 2", 10, home0);

Heater heater5=new Heater("Chauffage Chambre 3", 10, home0);

//Creation d'un tableau de chauffages

List<SmartDevice> heaters = new ArrayList<SmartDevice>();

heaters.add(heater0);

heaters.add(heater1);

heaters.add(heater2);

heaters.add(heater3);

heaters.add(heater4);

heaters.add(heater5);

//Creation des appareils électriques

ElectronicDevice ed0=new ElectronicDevice("TV", 4, home0);

ElectronicDevice ed1=new ElectronicDevice("Lecteur Blue-Ray", 1, home0);

ElectronicDevice ed2=new ElectronicDevice("Téléphone", 1, home0);

//Creation d'un tableau d'appareils électriques

List<SmartDevice> electronicDevices = new ArrayList<SmartDevice>();

electronicDevices.add(ed0);

electronicDevices.add(ed1);

electronicDevices.add(ed2);

//Affectation des chauffages et des appareils electriques à la maison

home0.setElectronicDevices(electronicDevices);

home0.setHeaters(heaters);

//Création des personnes
```

```

        Person person1 = new Person(0,"Toto","Jean","jean.toto@tpsir.org",homes);

        Person person2 = new Person(0,"Titi","Jacques","jacques.titi@tpsir.org",homes);

        //Objets rendus persistants

        EntityManagerHelper.getEntityManager().persist(home0);

        EntityManagerHelper.getEntityManager().persist(ed0);

        EntityManagerHelper.getEntityManager().persist(ed1);

        EntityManagerHelper.getEntityManager().persist(ed2);

        EntityManagerHelper.getEntityManager().persist(heater0);

        EntityManagerHelper.getEntityManager().persist(heater1);

        EntityManagerHelper.getEntityManager().persist(heater2);

        EntityManagerHelper.getEntityManager().persist(heater3);

        EntityManagerHelper.getEntityManager().persist(heater4);

        EntityManagerHelper.getEntityManager().persist(heater5);

        EntityManagerHelper.getEntityManager().persist(person1);

        EntityManagerHelper.getEntityManager().persist(person2);

    }

}

```

```

private static void listPerson() {

    List<Person> resultList = EntityManagerHelper.getEntityManager().createQuery("Select p
From Person p", Person.class).getResultList();

    System.out.println("Liste des personnes");

    System.out.println("num of person:" + resultList.size());

    for (Person next : resultList) {

        System.out.println("next person: " + next);

    }

    System.out.println("-----");

    List<SmartDevice> resultList2 =
EntityManagerHelper.getEntityManager().createQuery("Select s From SmartDevice s",
SmartDevice.class).getResultList();

```

```

System.out.println("Liste des smart devices");

System.out.println("num of smart devices:" + resultList2.size());

for (SmartDevice next : resultList2) {

    System.out.println("next smart device : " + next);

}

System.out.println("-----");

CriteriaBuilder cb = EntityManagerHelper.getEntityManager().getCriteriaBuilder();

CriteriaQuery query = cb.createQuery(Person.class);

Root from = query.from(Person.class);

query.select(from).where(cb.equal(from.get("name"), "Toto"));

List result = EntityManagerHelper.getEntityManager().createQuery(query).getResultList();

for (Object enregistrement : result) {

    System.out.println("enregistrement : " + enregistrement);

}

System.out.println("-----");

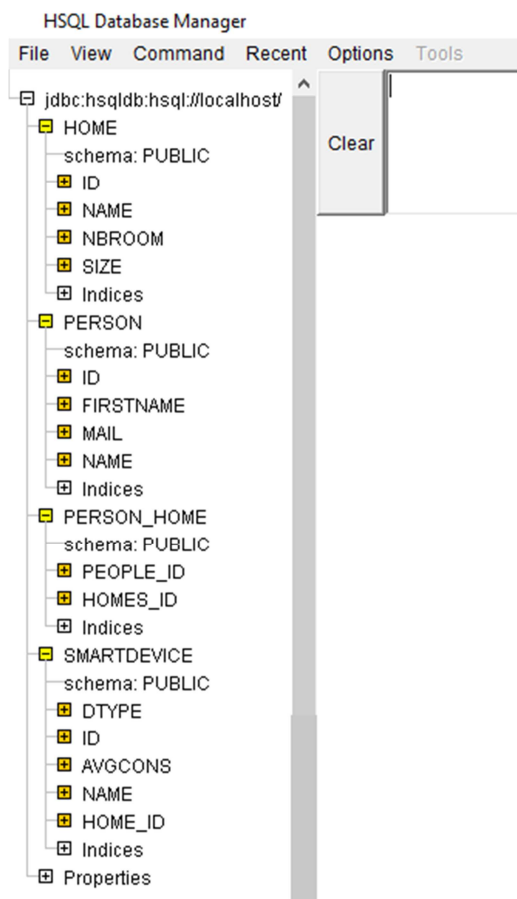
}

}

```

Affichage des résultats

Nous obtenons le résultat ci-dessous avec HSQL Database Manager :



Résultat des requêtes de la classe JpaTest

Les requêtes de test fonctionnent :

```
Liste des personnes
num of person:2
next person: Person [id=1, name=Toto]
next person: Person [id=2, name=Titi]
-----

Liste des smart devices
num of smart devices:9
next smart device : Electronic device : [id=1, name=TV]
next smart device : Electronic device : [id=2, name=Lecteur Blue-Ray]
next smart device : Electronic device : [id=3, name=Téléphone]
next smart device : Heater [id=4, name=Chauffage Salon]
next smart device : Heater [id=5, name=Chauffage Salle à manger]
next smart device : Heater [id=6, name=Chauffage Salle de bain]
next smart device : Heater [id=7, name=Chauffage Chambre 1]
next smart device : Heater [id=8, name=Chauffage Chambre 2]
next smart device : Heater [id=9, name=Chauffage Chambre 3]
-----

enregistrement : Person [id=1, name=Toto]
-----
```

Question 4 : Connexion à une base MySQL

Nous avons ajouté les paramètres de connexion à notre base MySQL dans le fichier *Persistence.xml*.

Nous avons modifié dans notre *EntityManagerFactory* défini dans la classe *EntityManagerHelper* pour se connecter à notre base MySQL. On indique en paramètre le nom de la connexion saisie dans le fichier *persistence.xml* : `emf = Persistence.createEntityManagerFactory("mysql");`

Ajout dans le fichier Persistence.xml

```
<persistence-unit name="mysql">
  <properties>
    <!--
      <property name="hibernate.ejb.cfgfile" value="/hibernate.cfg.xml"/>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
    -->
    <property name="hibernate.hbm2ddl.auto" value="validate"/>
    <property name="hibernate.archive.autodetection" value="class,
hbm"/>
    <property name="hibernate.show_sql" value="true"/>
    <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
    <property name="hibernate.connection.password" value="*****"/>
    <property name="hibernate.connection.url"
value="jdbc:mysql://anteros.istic.univ-rennes1.fr/base_21000155"/>
    <property name="hibernate.connection.username"
value="user_21000155"/>
    <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
    <property name="hibernate.c3p0.min_size" value="5"/>
    <property name="hibernate.c3p0.max_size" value="20"/>
    <property name="hibernate.c3p0.timeout" value="300"/>
    <property name="hibernate.c3p0.max_statements" value="50"/>
    <property name="hibernate.c3p0.idle_test_period" value="3000"/>
  </properties>
</persistence-unit>
```

Question 5 : Héritage

Nous décidons de faire hériter les classes *Heater* et *ElectronicDevice* d'une classe *SmartDevice*.

On crée une classe *Smart Device*. On modifie les classes *Heater* et *ElectronicDevice* pour les faire hériter de la classe *SmartDevice*. On saisit l'annotation `@DiscriminatorValue("Heater")` pour permettre d'identifier le type de smart device dans l'entité persistante (*Heater* ou *Electronic device*).

On ajoute l'annotation `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)` dans la classe *SmartDevice* car on souhaite n'avoir qu'une table qui contiendra tous nos smart device, c'est dire à la fois les *Heater* et les *ElectronicDevice*.

Class SmartDevice

```
package domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.ManyToOne;
import javax.xml.bind.annotation.XmlTransient;

import org.codehaus.jackson.annotate.JsonIgnore;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class SmartDevice {

    private long id;

    protected String name;

    private float avgCons;

    @JsonIgnore
    @XmlTransient
    private Home home;

    public String getName() {

        return name;

    }
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
public float getAvgCons() {  
    return avgCons;  
}
```

```
public void setAvgCons(float avgCons) {  
    this.avgCons = avgCons;  
}
```

```
@ManyToOne
```

```
@JsonIgnore
```

```
@XmlTransient
```

```
public Home getHome() {  
    return home;  
}
```

```
public SmartDevice(String name, float avgCons, Home home) {  
    super();  
    this.name = name;  
    this.avgCons = avgCons;  
    this.home = home;  
}
```

```
public SmartDevice() {  
}
```

```

        public void setHome(Home home) {

            this.home = home;

        }


        @Id

        @GeneratedValue

        public long getId() {

            return id;

        }


        public void setId(long id) {

            this.id = id;

        }

    }

```

Class Heater

```

package domain;


import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;


@Entity

@DiscriminatorValue("Heater")

public class Heater extends SmartDevice{


    public Heater() {

        super();
    }

```

```
}
```

```
public Heater(String name, float avgCons, Home home) {
```

```
    super(name, avgCons, home);
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
    return "Heater [id=" + super.getId() + ", name=" + name + "];"
```

```
}
```

```
}
```

Class ElectronicDevice

```
package domain;
```

```
import javax.persistence.DiscriminatorValue;
```

```
import javax.persistence.Entity;
```

```
@Entity
```

```
@DiscriminatorValue("Electronic Device")
```

```
public class ElectronicDevice extends SmartDevice{
```

```
    public ElectronicDevice() {
```

```
        super();
```

```
}
```

```
    public ElectronicDevice(String name, float avgCons, Home home) {
```

```
        super(name, avgCons, home);
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
    return "Electronic device : [id=" + super.getId() + ", name=" + name + "];
```

```
}
```

```
}
```

Question 6 : Mise en évidence du problème de n+1 select

On peuple la base avec JpaTest. On obtient 5000 department avec 10000 employés (2 employés par department).

Les temps de réponse sont meilleurs avec Joinfetch (913 ms) car une seule requête est effectuée. Les temps de réponse sont moins bons avec N1Select (6681 ms) car 5000 requêtes sont effectuées (une requête par department).

N1Select

```
temps d'exec = 6681
```

```
.. done
```

JoinFetch

```
temps d'exec = 913
```

```
.. done
```

TP3 : Servlet

Question 1 : Configuration du POM.xml

Nous avons modifié le POM.xml comme indiqué.

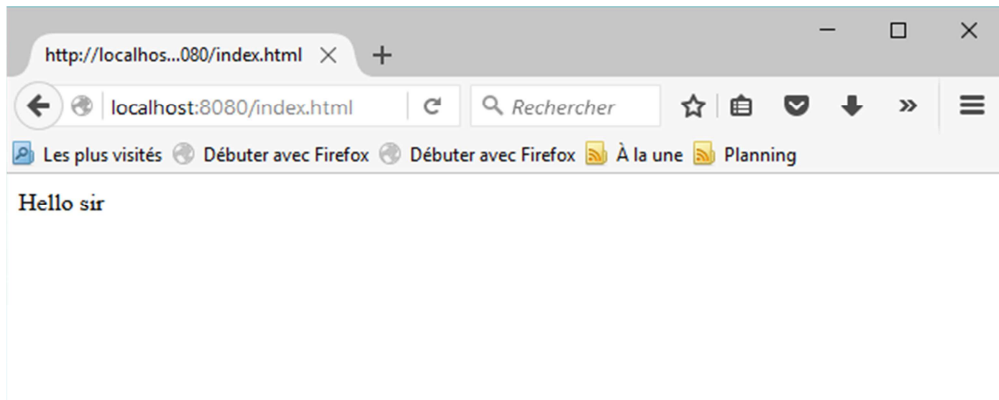
Le lancement de Tomcat réussi :

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building testjpa 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] >>> tomcat7-maven-plugin:2.2:run (default-cli) > process-classes @
testjpa >>>
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ testjpa ---
[INFO]
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ testjpa ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< tomcat7-maven-plugin:2.2:run (default-cli) < process-classes @
testjpa <<<
[INFO]
[INFO] --- tomcat7-maven-plugin:2.2:run (default-cli) @ testjpa ---
[INFO] Démarrage du war sur http://localhost:8080/
[INFO] Utilisation de la configuration existante du serveur Tomcat sur
C:\Users\user1\git\sirjpa\jpasample\target\tomcat
[INFO] create webapp with contextPath:
févr. 16, 2016 2:21:49 PM org.apache.coyote.AbstractProtocol init
INFOS: Initializing ProtocolHandler ["http-bio-8080"]
févr. 16, 2016 2:21:49 PM org.apache.catalina.core.StandardService startInternal
INFOS: Starting service Tomcat
févr. 16, 2016 2:21:49 PM org.apache.catalina.core.StandardEngine startInternal
INFOS: Starting Servlet Engine: Apache Tomcat/7.0.47
févr. 16, 2016 2:21:53 PM com.sun.jersey.api.core.PackagesResourceConfig init
INFOS: Scanning for root resource and provider classes in the packages:
    fr.istic.sir.rest
févr. 16, 2016 2:21:53 PM com.sun.jersey.api.core.ScanningResourceConfig
logClasses
INFOS: Root resource classes found:
    class fr.istic.sir.rest.SampleWebService
    class fr.istic.sir.rest.HomeService
févr. 16, 2016 2:21:53 PM com.sun.jersey.api.core.ScanningResourceConfig init
INFOS: No provider classes found.
févr. 16, 2016 2:21:53 PM
com.sun.jersey.server.impl.application.WebApplicationImpl _initiate
INFOS: Initiating Jersey application, version 'Jersey: 1.18.3 12/01/2014 08:23
AM'
févr. 16, 2016 2:21:54 PM org.apache.coyote.AbstractProtocol start
INFOS: Starting ProtocolHandler ["http-bio-8080"]
```


Question 2. Insertion de ressources statiques

On crée le répertoire `src/main/webapp` et on ajoute le fichier `index.html`.

On affiche correctement notre page à l'adresse <http://localhost:8080/index.html> :



Question 3. Création de votre première Servlet

Nous avons créé une classe *MyServlet* qui étend *HttpServlet* dans la package *servlet*.

Classe *MyServlet*

```
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);

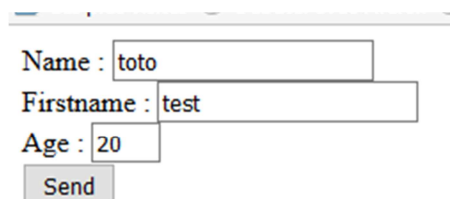
    }

}

```

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

Après avoir créé le fichier *myform.html* et la classe Java *UserInfo*, on teste notre première Servlet à l'adresse <http://localhost:8080/myform.html>



The screenshot shows a web browser window displaying a form. The form has three input fields: 'Name' with the value 'toto', 'Firstname' with the value 'test', and 'Age' with the value '20'. Below these fields is a 'Send' button.

Notre Servlet récupère bien les informations saisies :

Recapitulatif des informations

- Nom: toto
- Prenom: test
- Age: 20

Question 5. Retour sur OPOWER

Nous avons créé une page heater1.html. Cette page contient un premier formulaire qui permet d'ajouter un chauffage et un second qui affiche la liste des équipements électriques stockés en base. Chaque formulaire envoie une requête *HTTP* sur l'URI */HeaterInfo* avec la méthode *POST* pour l'ajout et la méthode *GET* pour l'affichage.

```
<html>
<body>
    <FORM Method="POST" Action="/HeaterInfo" Name="Form1">
        Name : <INPUT type=text size=20 name=name><BR> Average
        consomation : <INPUT type=float size=20 name=AvgCons><BR>
        Home : <INPUT type=text size=2 name=home><BR> <INPUT
                type=submit value=Send>
    </FORM>
    <FORM Method="GET" Action="/HeaterInfo" Name="Form2">
        <BR> <INPUT type=submit value="Afficher les chauffages">
    </FORM>
</body>
</html>
```

Nous créons ensuite une classe *HeaterInfo* qui hérite de la classe *HttpServlet*. Nous utilisons l'annotation *@webServlet* pour définir notre classe en tant que servlet et la faire correspondre à l'URI */HeaterInfo*. On définit les méthodes *doPost* et *doGet* qui seront appelés avec les requêtes *POST* et *GET* envoyés depuis la page web *Heater1.html*.

```
package servlet;
```

```
import java.io.IOException;
```

```
import java.io.PrintWriter;
```

```
import java.util.List;
```

```
import javax.persistence.EntityTransaction;
```

```
import javax.servlet.ServletException;
```

```

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


import domain.Heater;

import domain.Home;

import jpa.EntityManagerHelper;


@WebServlet(name="heaterInfo",urlPatterns={"/HeaterInfo"})

public class HeaterInfo 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>Consommation moyenne : "

            + request.getParameter("AvgCons") + "\n" +

            "  <LI>Résidence : "

            + request.getParameter("home") + "\n" +

            "</UL>\n");


        EntityTransaction tx = EntityManagerHelper.getEntityManager().getTransaction();

        tx.begin();

```

```

try {

    Home home0 = new Home("résidence principale",90, 6, null,null,null);

    Heater heater6=new Heater(request.getParameter("name"),
Integer.parseInt(request.getParameter("AvgCons")), home0);

    EntityManagerHelper.getEntityManager().persist(home0);

    EntityManagerHelper.getEntityManager().persist(heater6);

} catch (Exception e) {

    e.printStackTrace();

}

tx.commit();

out.println("Enregistrement effectué</BODY></HTML>");

}

```

```

public void doGet(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String query = "select h from Heater as h";

    List result2 = EntityManagerHelper.getEntityManager().createQuery(query).getResultList();

    out.println("<HTML>\n<BODY>\n" +

        "<H1>Recapitulatif des informations</H1>\n" +

        "<UL>\n");

    for (Object enregistrement : result2) {

        out.println("<LI> enregistrement : " + enregistrement+"\n");

    }

    out.println("</UL>\n" +

        "</BODY></HTML>");

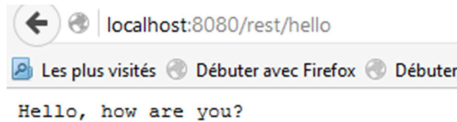
}

}

```

Question 6. En avant pour les architectures Rest.

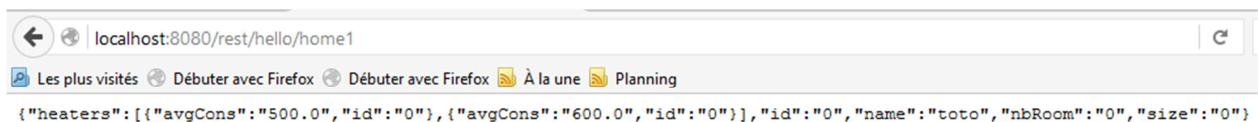
Nous avons modifié le fichier pom.xml pour ajouter la dépendance à Jersey et celle de jersey-json et créé le fichier web.xml. Nous avons ensuite créé le package Java `fr.istic.sir.rest` et la classe `SampleWebService`. Nous avons testé notre classe à l'adresse suivante : <http://localhost:8080/rest/hello> :



On ajoute la méthode ci-dessous à notre classe `SampleWebService`.

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

Nous obtenons bien notre objet au format JSON à l'adresse <http://localhost:8080/rest/hello/home>.



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

Nous avons créé une classe de service pour la gestion des maisons. Cette classe `HomeService` est contenu dans le package `fr.istic.sir.rest`. Cette classe permet d'effectuer les opérations de listing, ajout, modification et suppression sur l'entité `Home`.

Classe `HomeService`

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

```
import java.util.Collection;
```

```
import java.util.List;

import java.util.Map;
```

```
import javax.persistence.Query;

import javax.ws.rs.DELETE;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;
```

```
import domain.Heater;

import domain.Home;

import domain.Person;

import jpa.EntityManagerHelper;
```

```
@Path("/home")
```

```
public class HomeService {
```

```
    @GET
```

```
    @Produces(MediaType.APPLICATION_JSON)
```

```
    public Collection<Home> getHomes() {
```

```
        String query = "select h from Home as h";
```

```
        List result = EntityManagerHelper.getEntityManager().createQuery(query).getResultList();
```

```
        return result;
```

```
    }
```

```
    @POST
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
public Home create() {
```

```
    Home homeTestRest = new Home("résidence de test de Rest",90, 6, null,null,null);
```

```
    EntityManagerHelper.getEntityManager().getTransaction().begin();
```

```
    EntityManagerHelper.getEntityManager().persist(homeTestRest);
```

```
    EntityManagerHelper.getEntityManager().getTransaction().commit();
```

```
    return homeTestRest;
```

```
}
```

```
@DELETE
```

```
@Path("/{id}")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
public void delete(@PathParam("id") String arg0) {
```

```
    String query = "select h from Home as h where h.id="+Integer.parseInt(arg0);
```

```
    List<Home> result =
```

```
    EntityManagerHelper.getEntityManager().createQuery(query).getResultList();
```

```
    if (result.size()!=0){
```

```
        Home homeTest= result.get(0);
```

```
        EntityManagerHelper.getEntityManager().getTransaction().begin();
```

```
        homeTest = EntityManagerHelper.getEntityManager().merge(homeTest);
```

```
        EntityManagerHelper.getEntityManager().remove(homeTest);
```

```
        EntityManagerHelper.getEntityManager().getTransaction().commit();
```

```
    }
```

```
}
```

```
@GET
```

```
@Path("/{id}")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
public Home find(@PathParam("id") String arg0) {
```



```

        String query = "select h from Home as h where h.id="+Integer.parseInt(arg0);

        List<Home> result =
EntityManagerHelper.getEntityManager().createQuery(query).getResultList();

        if (result.size()==0){

            return null;

        }else{

            Home homeTest= result.get(0);

            return homeTest;

        }

    }
}

```

```

@POST

@Path("/{id}")

@Produces(MediaType.APPLICATION_JSON)

public Home update(@PathParam("id") String arg0) {

    String query = "select h from Home as h where h.id="+Integer.parseInt(arg0);

    List<Home> result =
EntityManagerHelper.getEntityManager().createQuery(query).getResultList();

    if (result.size()==0){

        return null;

    }else{

        Home homeTest= result.get(0);

        homeTest.setName("Test update REST");

        EntityManagerHelper.getEntityManager().getTransaction().begin();

        homeTest = EntityManagerHelper.getEntityManager().merge(homeTest);

        EntityManagerHelper.getEntityManager().getTransaction().commit();

        return homeTest;

    }

}
}

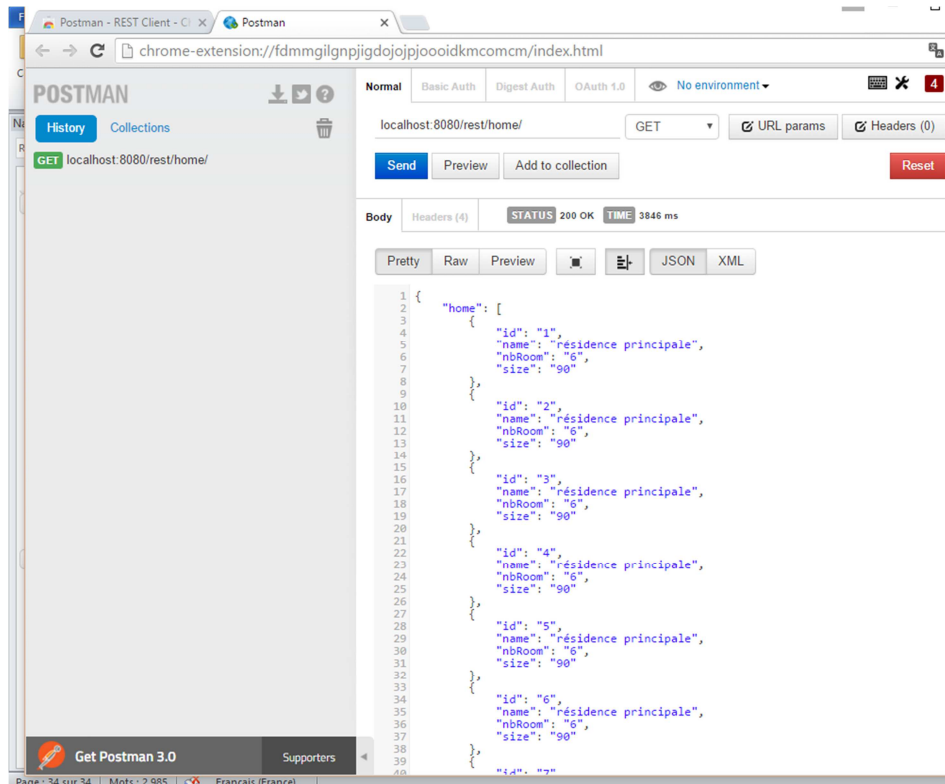
```

}

Résultat avec Postman

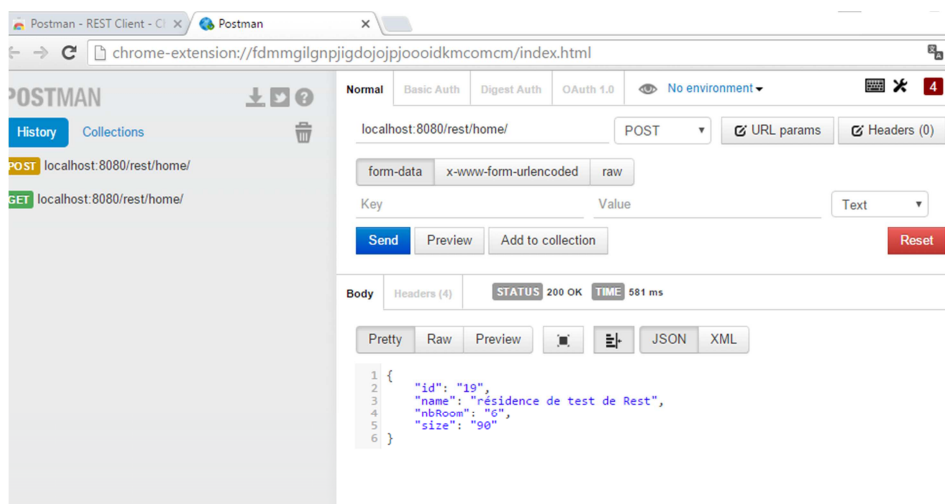
- Liste des résidences :

Requête GET à l'adresse <http://localhost:8080/rest/home>



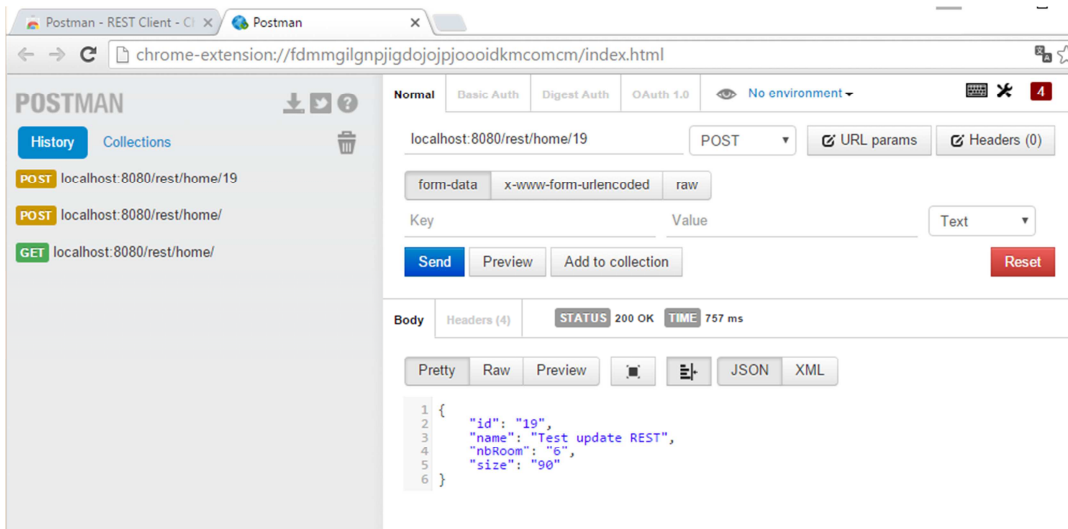
- Ajout d'une résidence :

Requête POST à l'adresse <http://localhost:8080/rest/home>



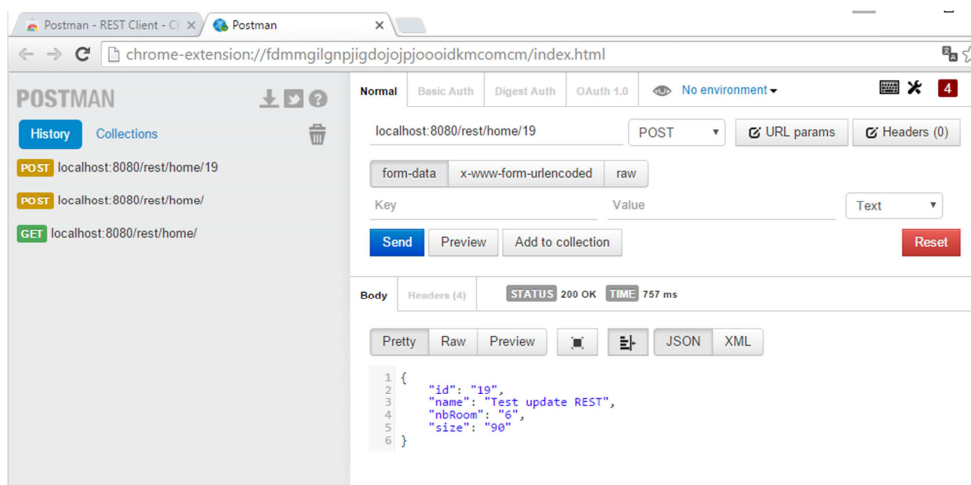
- Modification d'un élément (id 19) de l'entité Home

Requête POST à l'adresse <http://localhost:8080/rest/home/19>



- Affichage d'un élément (id 19) de l'entité Home

Requête GET à l'adresse <http://localhost:8080/rest/home/19>



- Suppression d'un élément (id 19) de l'entité Home

Requête DELETE à l'adresse <http://localhost:8080/rest/home/19>

