

Technologies des services du Web

Jérôme DEGROOTE

Société générale



Persistence



Utilisation des BDDs

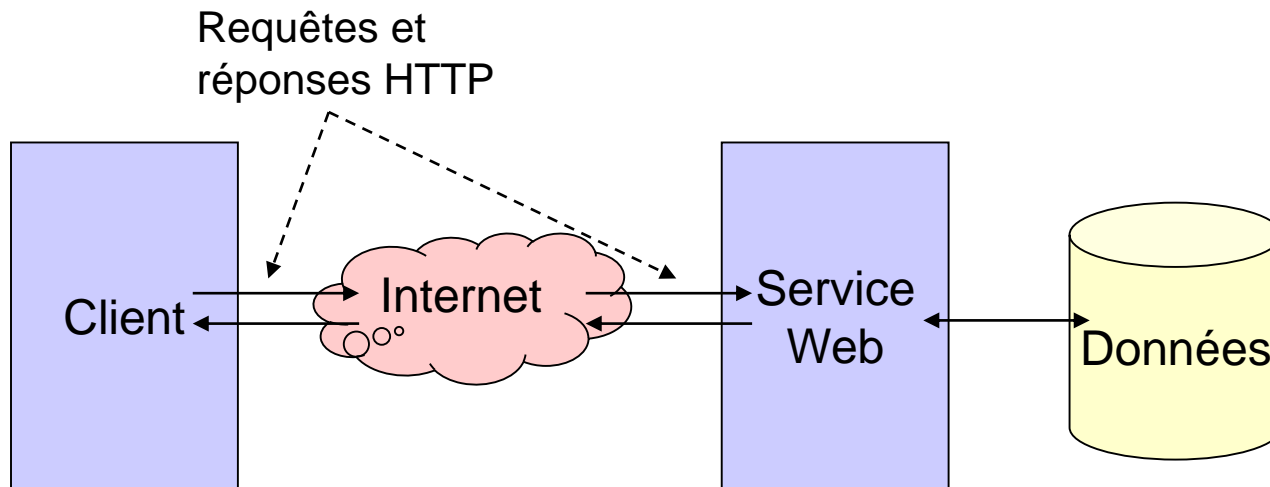
- Incontournable

- Toute application doit disposer d'un moyen d'accès aux données
 - Fiable
 - Persistant
 - Rapide

- Du SI d'entreprise...

...à votre téléphone

Exemple pour un site Web



JPA (1)

- Java Persistence API
- Atouts
 - "Léger"
 - Orienté POJO
- Spécification pour les EJB 3.0
 - 1 document pour l'introduction de nouvelles fonctionnalités dans EJB2.1
 - 1 document pour les beans sessions et message-driven
 - 1 document pour la persistance -> JPA

JPA (2)

- Le modèle se veut
 - Simple
 - Puissant
 - Flexible
- L'utilisation des POJO
 - Un objet applicatif peut être rendu persistant avec +/- 1 ligne de code !
 - Persistance définie uniquement par métadonnées
 - Annotations, XML

JPA (3)

- **Persistance non-intrusive**
 - L'API est une couche externe aux objets de persistance
 - Appel de l'API dans la logique métier
 - Objets à persister
 - Instructions à réaliser
- **Requêtes objets**
 - Langage JP-QL
 - Requêtes vers les objets et leurs relations sans forcément utiliser clés externes ou colonnes de la BDD

JPA (4)

- Entités mobiles
 - Architectures clients/serveurs
 - Un objet doit pouvoir "bouger" d'une machine virtuelle à l'autre
- Objets détachés/attachés
- Configuration simple
- Existe avec ou sans serveur d'application

JPA (5)

- Version actuelle

- JPA 2.2
- La version JPA 1 est encore employée et reste proche de la dernière version

- Principaux fournisseurs

- Oracle, EclipseLink, Hibernate, OpenJPA,...

Persitence.xml (v2)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="EmployeeService"
    transaction-type="RESOURCE_LOCAL">
    <class>org.istv.jpa.Employee</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="org.hsqldb.jdbcDriver" />
      <property name="javax.persistence.jdbc.url"
        value="jdbc:hsqldb:hsqldb://localhost;/create=true" />
      <property name="javax.persistence.jdbc.user" value="SA" />
      <property name="javax.persistence.jdbc.password" value="" />
    </properties>
  </persistence-unit>
</persistence>
```

D'un POJO...

```
public class Employee {  
  
    private int id;  
    private String name;  
    private long salary;  
  
    public Employee() {}  
  
    public Employee(int id) {this.id = id;}  
  
    public int getId() {return id;}  
  
    public void setId(int id) {this.id = id;}  
  
    public String getName() {return name;}  
  
    public void setName(String name) {this.name = name;}  
  
    public long getSalary() {return salary;}  
  
    public void setSalary(long salary) {this.salary = salary;}  
}
```

...vers un Entity

@Entity

```
public class Employee {  
    @Id  
    private int id;  
    private String name;  
    private long salary;  
  
    public Employee() {}  
  
    public Employee(int id) {this.id = id;}  
  
    public int getId() {return id;}  
  
    public void setId(int id) {this.id = id;}  
  
    public String getName() {return name;}  
  
    public void setName(String name) {this.name = name;}  
  
    public long getSalary() {return salary;}  
  
    public void setSalary(long salary) {this.salary = salary;}  
}
```

Ajout d'un élément persistant

```
em.getTransaction().begin();  
Employee emp = new Employee(id);  
emp.setName(name);  
emp.setSalary(salary);  
em.persist(emp);  
em.getTransaction().commit();
```

→ Penser au contexte transactionnel

Recherche d'un élément

```
em.find(Employee.class, id);
```

```
Query query =
```

```
em.createQuery("SELECT e FROM Employee e");
```

```
Collection<Employee> ce =
```

```
    (Collection<Employee>) query.getResultList();
```

Suppression

```
Employee emp =  
    em.find(Employee.class, id);  
  
if (emp != null) {  
    em.remove(emp);  
}
```

Accès aux données

```
Employee emp = em.find(Employee.class, id);
```

```
if (emp != null) {  
    emp.setSalary(emp.getSalary() + raise);  
}
```


Tester la persistance

```
public class Test {  
  
    public static void main(String[] args) {  
        EntityManagerFactory emf =  
            Persistence  
                .createEntityManagerFactory("EmployeeBDD");  
        EntityManager em = emf.createEntityManager();  
  
        em.close();  
        emf.close();  
    }  
}
```

Mapping Objet/Relationnel

A decorative graphic consisting of a vertical green line and a horizontal green line intersecting at a point. The vertical line is positioned to the left of the title, and the horizontal line extends across the width of the slide below the title.

Lien vers une table

- Annotation de la classe
- Attribut @Table

```
@Entity  
@Table(name="EMP")  
public class Employee { ... }
```

```
@Entity  
@Table(name="EMP", schema="HR")  
public class Employee { ... }
```

```
@Entity  
@Table(name="EMP", catalog="HR")  
public class Employee { ... }
```

Lien vers une colonne

- Annotation de l'attribut
- Attribut @Column

```
@Entity
public class Employee {
    @Id
    @Column(name="EMP_ID")
    private int id;
    private String name;
    @Column(name="SAL")
    private long salary;
    @Column(name="COMM")
    private String comments;
    // ...
}
```

Clé primaire

- Génération automatique
- Automatique

```
@Entity
public class Employee {
    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    // ...
}
```

- Par une table

```
@Id @GeneratedValue(strategy=GenerationType.TABLE)
private int id;
```

Relations

- Exemple Many to one



- Exemple Many to Many



Mappings possibles

- Many to One
- One to Many
- One to One
- Many to Many

One to Many unidirectionnelle

- Relation inverse de Many to One

```
@Entity
public class Department {
    @Id private int id;
    private String name;
    @OneToMany
    private Collection<Employee> employees;
    // ...
}
```


One to Many bidirectionnelle

■ Relation inverse de Many to One

```
@Entity
public class Department {
    @Id private int id;
    private String name;
    @OneToMany(mappedBy="department")
    private Collection<Employee> employees;
    // ...
}
```



Equivalent

```
@Entity
public class Department {
    @Id private int id;
    private String name;
    @OneToMany(targetEntity=Employee.class, mappedBy="department")
    private Collection employees;
    // ...
}
```

One To One (1)

```
@Entity
public class Employee {
    @Id private int id;
    private String name;
    @OneToOne
    @JoinColumn(name="PSPACE_ID")
    private ParkingSpace parkingSpace;
    // ...
}
```

One To One (2)

- Bi-directionnel

```
@Entity
public class ParkingSpace {
    @Id private int id;
    private int lot;
    private String location;
    @OneToOne(mappedBy="parkingSpace")
    private Employee employee;
    // ...
}
```

Many to Many

```
@Entity
public class Employee {
    @Id private int id;
    private String name;
    @ManyToMany
    private Collection<Project> projects;
    // ...
}

@Entity
public class Project {
    @Id private int id;
    private String name;
    @ManyToMany(mappedBy="projects")
    private Collection<Employee> employees;
    // ...
}
```

Requêtes JP-QL



Sélection simple

- Ca ressemble à du SQL
 - Tant mieux !
- L'idée est de parler objet

```
SELECT e  
FROM Employee e
```

Sélection d'attributs

- Liste de noms d'employés

```
SELECT e.name  
FROM Employee e
```

- Liste d'attributs à travers une relation

```
SELECT e.department  
FROM Employee e
```

Requêtes paramétrables

- 2 types de déclarations supportées

```
SELECT e
FROM Employee e
WHERE e.department = ?1 AND
      e.salary > ?2
```

```
SELECT e
FROM Employee e
WHERE e.department = :dept AND
      e.salary > :base
```


Utilisation de requêtes statiques paramétrables

```
private static final String QUERY =  
    "SELECT e.salary " +  
    "FROM Employee e " +  
    "WHERE e.department.name = :deptName AND " +  
    "      e.name = :empName ";  
  
public long queryEmpSalary(String deptName, String empName) {  
    return (Long) em.createQuery(QUERY)  
        .setParameter("deptName", deptName)  
        .setParameter("empName", empName)  
        .getSingleResult();  
}
```

Requêtes nommées (1)

- Généralement définie dans le composant Entity le plus proche de la requête

```
@NamedQuery(name="findSalaryForNameAndDepartment",  
            query="SELECT e.salary " +  
                  "FROM Employee e " +  
                  "WHERE e.department.name = :deptName AND " +  
                  "      e.name = :empName")
```

Requêtes nommées (2)

- Définition de plusieurs requêtes

```
@NamedQueries({  
    @NamedQuery(name="Employee.findAll",  
        query="SELECT e FROM Employee e"),  
    @NamedQuery(name="Employee.findByPrimaryKey",  
        query="SELECT e FROM Employee e WHERE e.id = :id"),  
    @NamedQuery(name="Employee.findByName",  
        query="SELECT e FROM Employee e WHERE e.name = :name")  
})
```

Requêtes nommées (3)

■ Exécution

```
public Employee findEmployeeByName(String name) {  
    return (Employee) em.createNamedQuery("Employee.findByName")  
        .setParameter("name", name)  
        .getSingleResult();  
}
```

En pratique



Environnement

- HSQLDB 2.4.0
- Hibernate 5.2.12.Final
- Télécharger les .zip et et les décompresser



HSQLDB - 100% Java Database

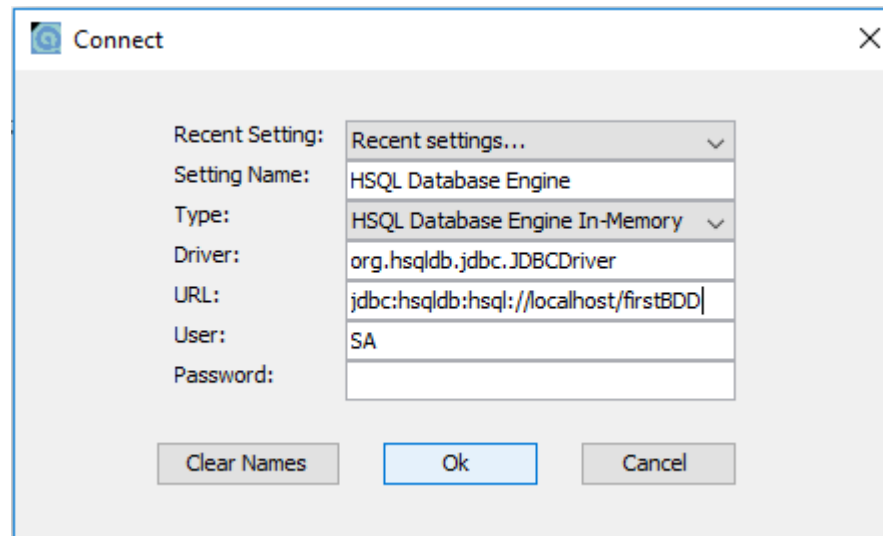


HIBERNATE

Lancer HSQLDB

- Se placer dans le dossier contenant hsqldb.jar
- Démarrer HSQLDB :
 - `java -cp hsqldb.jar org.hsqldb.server.Server --database.0 file:mydb --dbname.0 firstBDD`
- Démarrer « Database Manager »
 - `java -cp hsqldb.jar org.hsqldb.util.DatabaseManagerSwing`

Database Manager



The image shows a 'Connect' dialog box with a title bar containing a blue icon and a close button. The dialog contains several labeled fields and three buttons at the bottom.

Recent Setting:	Recent settings... ▾
Setting Name:	HSQL Database Engine
Type:	HSQL Database Engine In-Memory ▾
Driver:	org.hsqldb.jdbc.JDBCDriver
URL:	jdbc:hsqldb:hsq://localhost/firstBDD
User:	SA
Password:	

Buttons: Clear Names, Ok, Cancel

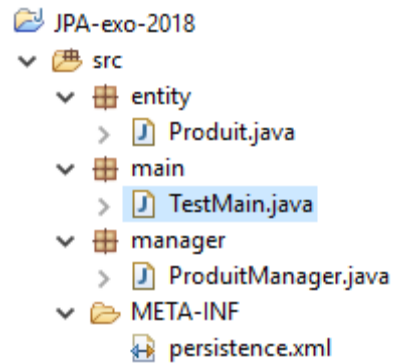
Comment démarrer ?

- Dans eclipse :
 - Créer un java project
 - Mettre dans le build path :
 - Le .jar hsqldb
 - Les .jar hibernate du dossier « required »
 - Ne pas oublier le persistence.xml

Exercice

1. Créer une entité « Produit » avec un id, un nom et un prix
2. Créer un manager permettant de créer nouveau produit en BDD
3. Créer un main() pour tester la création d'un nouveau produit
4. Pour approfondir :
 1. Enrichir le manager pour gérer le CRUD sur le Produit
 2. Créer une deuxième entité avec un lien OneToMany

Arborescence



Produit.java

@Entity

```
public class Produit {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private int id;  
    private String nom;  
    private float prix;  
  
    public Produit() {  
        super();  
    }  
  
    public Produit(String nom, float prix) {  
        super();  
        this.nom = nom;  
        this.prix = prix;  
    }  
  
    // Getters, Setters ...  
}
```

ProduitManager.java

```
public class ProduitManager {  
    private EntityManager em;  
  
    public ProduitManager(EntityManager em) {  
        this.em = em;  
    }  
  
    public Produit createProduit(String nom, float prix) {  
        Produit p = new Produit(nom, prix);  
        em.persist(p);  
        return p;  
    }  
}
```

TestMain.java

```
public class TestMain {  
    public static void main(String[] args) {  
        EntityManagerFactory emf =  
            Persistence.createEntityManagerFactory("firstBDD");  
        EntityManager em = emf.createEntityManager();  
  
        ProduitManager pm = new ProduitManager(em);  
        em.getTransaction().begin();  
        Produit p = pm.createProduit("nom", 10.5f);  
        em.getTransaction().commit();  
  
        System.out.println("Produit créé : " + p.getId());  
  
        em.close();  
        emf.close();  
    }  
}
```

Persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
version="2.0">
  <persistence-unit name="firstBDD">
    <description>Fichier de conf de la BDD</description>

    <class>entity.Produit</class>

    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.hsqldb.jdbc.JDBCdriver" />
      <property name="javax.persistence.jdbc.url"
value="jdbc:hsqldb:hsqldb://localhost/firstBDD;create=true" />
      <property name="javax.persistence.jdbc.user" value="sa" />
      <property name="javax.persistence.jdbc.password" value="" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
      <!-- <property name="hibernate.hbm2ddl.auto" value="update" /> -->
    </properties>
  </persistence-unit>
</persistence>
```

La persistance et spring boot

Spring data

- Couche d'abstraction aux sources de données
- Implémente automatiquement les opérations de bases
- Plus de `persistence.xml`
- Configuration directement dans le `application.properties`

Spring data

■ Dépendance maven

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

■ Application.properties

```
spring.datasource.driver-class-name=org.hsqldb.jdbcDriver  
spring.datasource.url=jdbc:hsqldb:hsql://localhost/firstBDD  
spring.datasource.username=SA  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update
```

Les repositories

- Interfaces héritant de Repository
- Permet de fournir les méthodes de base d'accès aux données
- Permet de déclarer de nouvelles méthodes d'accès aux données qui seront implémentées automatiquement par Spring

CrudRepository

- Fournit les méthodes CRUD pour l'accès à la source de données :
 - save, findOne, findAll, delete...
- 2 types paramétrés sur l'interface :
 - L'entité manipulée
 - Le type de l'identifiant de l'entité

```
public interface PersonneRepository extends  
CrudRepository<Personne, Long> {}
```

CrudRepository

- Instanciée automatiquement grâce à l'annotation `@Autowired`

`@Autowired`

```
private PersonRepository personneRepository;
```

- Utilisation

```
personneRepository.findAll();
```

```
personneRepository.save(p);
```

```
personneRepository.deleteById(id);
```

Méthodes requêtes

- Spring data permet d'écrire des requêtes automatiquement à partir des noms de méthodes
- Il suffit de mettre les bons mots clés dans le nom de la méthode
 - And, Or, OrderBy...
- <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details>

Méthodes requêtes

```
public interface PersonRepository extends CrudRepository<Personne, Long> {  
    List<Personne> findByAge(int age);  
    List<Personne> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);  
    List<Personne> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);  
    List<Personne> findByLastnameIgnoreCase(String lastname);  
    List<Personne> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);  
    List<Personne> findByLastnameOrderByFirstnameAsc(String lastname);  
    List<Personne> findByLastnameOrderByFirstnameDesc(String lastname);  
}
```

Définir ses propres requêtes

- Pour définir sa propre requête, il suffit d'utiliser `@Query`
- Requêtes écrites en JPQL

```
@Query("select p from Personne p where p.prenom = :prenom or  
p.age = :age")
```

```
List<Personne> trouverPersonneParAgeOuPrenom(@Param("prenom")  
String prenom, @Param("age") int age);
```


Définir ses propres requêtes

- Pour les requêtes effectuant des modifications, il faut utiliser l'annotation `@Modifying`

```
@Query("update Personne p set p.nom = :nom  
where p.id = :id")
```

```
@Modifying
```

```
public int metAJourNom(@Param("nom")String  
nom, @Param("id") Long id);
```

Comment démarrer

- Mettre les dépendances suivantes dans le pom.xml :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.hsqldb</groupId>
  <artifactId>hsqldb</artifactId>
  <scope>runtime</scope>
</dependency>
```

- Configuration dans application.properties
- Créer une entité et une interface de repository
- Ne pas oublier de démarrer le serveur de BDD hsqldb 😊

application.properties

```
spring.datasource.driver-class-name=org.hsqldb.jdbcDriver  
spring.datasource.url=jdbc:hsqldb:hsql://localhost/firstBDD  
spring.datasource.username=SA  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update
```

Personne

@Entity

```
public class Personne implements Serializable {  
    private static final long serialVersionUID = -8537962680206576813L;  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
    private String prenom;  
    private int age;  
  
    public Personne() {  
        super();  
    }  
    public Personne(String prenom, int age) {  
        super();  
        this.prenom = prenom;  
        this.age = age;  
    }  
    // Getters, Setters ...  
}
```

PersonneRepository

```
public interface PersonneRepository extends CrudRepository<Personne,
Long> {
    List<Personne> findByAge(int age);
    List<Personne> findByPrenomAndAgeGreaterThanOrderByAgeDesc(String
prenom, int age);

    @Query("select p from Personne p where p.prenom = :prenom or p.age
= :age")
    List<Personne> trouverPersonneParAgeOuPrenom(@Param("prenom")
String prenom, @Param("age") int age);
}
```

PersonneResource

```
@Path("personnes")
```

```
public class PersonneResource {
```

```
    @Autowired
```

```
    private PersonneRepository personneRepository;
```

```
    @POST
```

```
    @Consumes(MediaType.APPLICATION_JSON)
```

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

```
    public Personne createPersonne(Personne p) {
```

```
        return personneRepository.save(p);
```

```
    }
```

```
}
```