Master 2 MIASHS
2019-2020
Jérôme DAVID

TP3 Java EE

JPA Basis

In this TP we will work with Java Persistence API. It is a framework that allows to to do relational-object mapping. It can be seen as a translator that transform objects into row in database and vice-versa.
It is based on annotations: classes, relations between classes and constraints are described directly in the entity code thanks to annotations.

In this TP, we will first makes the `SempicUser` class, a JPA entity. Then we will create classes that allows user to create groups of user and add users into theses groups.

1- Preliminaries

Edit `pom.xml` and set up the dependencies to look like:

```
<dependencies>
    <!-- for JPA -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>5.4.5.Final</version>
        <scope>runtime</scope>
    </dependency>

    <!-- jakartaee web API & JSF-->
    <dependency>
        <groupId>jakarta.platform</groupId>
        <artifactId>jakarta.jakartaee-web-api</artifactId>
        <version>8.0.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>jakarta.faces</groupId>
        <artifactId>jakarta.faces-api</artifactId>
        <version>2.3.2</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Create the class `ApplicationConfig` in the package `fr.uga.miashs.sempic`.

```
@DataSourceDefinition(
   name = ApplicationConfig.DATA_SOURCE,
   className = "org.hsqldb.jdbcDriver",
   url = "jdbc:hsqldb:file:sempicdb",
   databaseName = "SempicDB",
   user = "sempic",
```

```
    password = "sempic"
)

@FacesConfig(version = FacesConfig.Version.JSF_2_3)
@Singleton
@Named
@Startup
public class ApplicationConfig {
    public final static String DATA_SOURCE = "java:app/sempicdb";

    @PostConstruct
    public void init() {
    }
}
```

In the project, declare a persistence unit and associate it to the DB:

- File → new file → Persistence → Persistence Unit
- leave values with default, we will edit it after and replace everything.
- Edit the persistance.xml file (within directory META-INF) and put this:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="SempicPU" transaction-type="JTA">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <!--
provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider-->
    <jta-data-source>java:app/sempicdb</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <!-- for glassfish only -->
      <!--property name="hibernate.transaction.jta.platform"
value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/-->
      <property name="javax.persistence.schema-generation.database.action"
value="create"/> <!--drop-and-create-->
      <!-- for tomee only -->
      <property name="tomee.jpa.factory.lazy" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

2- SempicUser as JPA Entity

In order to enable object to relational mapping for a given java class, we have to do:

- Make the class a Java Bean (default no-arg constructor, private attributes, getters and setters).
- Annotate the class with `@javax.persistence.Entity` (or `@Entity` and `import javax.persistence.*`).
- Annotate the primary key field with `@javax.persistence.Id` (or `@Id` + import)

Annotate the class `SempicUser` with `@Entity.` Add an attribute (and a getter) `id` (type long) and annotate it with `@Id`, in order to make your `SempicUser` class a JPA Entity. Modify the equals and hashcode method in order to the equality between users rely on `id` attribute and not email anymore.

3- Abstract JPA service

For all entities, a basic service should be able to: create, read, update, delete and list all entities that are stored in the storage layer. JPA offers most of these services through the EntityManager API. All

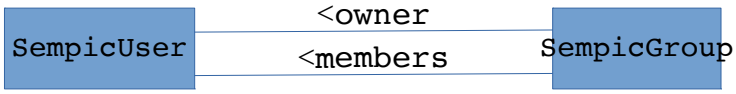
the code that deals with these basic operations will be the same for all entities. Then, it is good practice to factorise this code by introducing an abstract JPA Facade for these operations.
For this create in the package `fr.uga.miashs.sempic.dao` the class `AbstractJpaFacade` given in annex.
In the code, you can see that the persistence context is injected through the annotation `@PersistenceContext(unitName="SempicPU")`

4- SempicUserService

In the package `fr.uga.miashs.sempic.dao`, create the class `SempicUserJpaFacade` that extends `AbstractJpaFacade`.
Make this class a Stateless Session Bean using the annotation `@Stateless`
Modify the code of `CreateUser` in order to inject and use an instance of `SempicUserJpaFacade` instead of `SempicUserMap`.
Test your app, and verify that the user is now handled by the database:
Go to Services Tab (on the left frame of Netbeans) → Databases → JavaDB → SempicDB (connect) → Tables → SEMPICUSER → Right Click(View Data)

5- Adding Group Entity

We want users to create and be members of groups. We have then to create an entity group.
A group is represented by an id (type long), a name, an owner (a SempicUser), a set of members (a set of SempicUser).
There two relations between `SempicUser` and `SempicGroup` :



The relation owner from `SempicGroup` to `SempicUser` is a ManyToOne relation: A group has only one owner but a user can be owner of several groups. In this case, in JPA, the attribute `owner` of `SempicGroup` has to be annotated with `@ManyToOne`.
The relation member from `Group` to `SempicUser` is a ManyToMany relation: A group can have several members and a user can be member of several groups. In that case, the attribute `members` of `SempicGroup` has to annotated with `@ManyToMany`.
You can also add the reverse relations: in `SempicUser` you can add the set of group owned by the user (attribute `Set<SempicGroup> groups`) and the set of group he/she is member of (attribute `Set<SempicGroup> memberOf`). To that extent you have to add annotation `@OneToMany(mappedBy="owner")` on attribute `groups` and `@ManyToMany(mappedBy="members")` on `memberOf`. The parameter ownedBy means that the relation is handled (owned) by the reverse side, i.e. the class Group. If you omit this, the relation and its reverse will not be synchronized (they will be considered as different relations).

Implement the class `SempicGroup`, the reverse relations in `SempicUser` and create the corresponding service `SempicGroupJpaFacade`.
Implement a view (page+backing bean) for creating a group.
The id of the owner of the group will be a parameter of the view. To add a parameter, use the following tag:

```
<f:metadata>
    <f:viewParam name="userId" value="#{createGroup.ownerId}" />
</f:metadata>
```

Extras:

In order to see what the database contains, you can create a connection within Netbeans:
Create a connection to the database (the database will be created only after having executed at least once the project):
Go on the “service” tab (window → services). Open Databases, and right click on Databases → new connection. Choose HSQLDB, Next, and create the connection using the following parameters:
User Name: sempic
Password: sempic
jdbc:hsqldb:file:/CHEMIN_VERS_TOMEE/sempicdb