## **Étape 1 : Stockage (DAO/JDBC/JPA)**

**Base de données**

Notre base de données comportera deux tables, une table PERSON permettant de décrire une personne et une table GROUP permettant de décrire un groupe. Une personne appartient à un et un seule groupe et un groupe peut appartenir à 0 ou plusieurs personne.

**Schéma Merise**

|  |
| --- |
| Person |
| * id * firstname * lastname * birthdate * email * website * password * group\_id |

Relation : Table Person (1,1) -----> Table Group (0,n)

|  |
| --- |
| Group\_ |
| * groupID * name |

**Diagramme de class**

Au niveau de la couche base de données, nous avons deux javaBeans, Person.java et Group.java qui représentent les deux tables (Person et Group\_). Les fichiers PersonDao.java et GroupDao servent à gérer la persistence des données avec la technologie OpenJPA.

**Entity**

@Entity(name = "Person")

|  |
| --- |
| **Person** |
| - id  - firstname  - lastname  - birthdate  - email  - website  - password  - group\_id |
| \* getteur et setteur |

@Entity(name = "Group\_")

|  |
| --- |
| **Group** |
| **-** groupID  - name |
| \* getteur et setteur |

**Data Access Object**

|  |
| --- |
| **IPersonDao** << Interface>> |
|  |
| + init(): void  + getPerson: Person  + getAllPersons(): List<Person>  + personIDExists(long id): boolean  + updatePerson(Person p): void  + close(): void |

|  |
| --- |
| **IGroupDao** << Interface>> |
|  |
| + init(): void  + getAllGroups():List<Group>  + getContentGroup(long id): Collections<Person>  + getGroup(long id): Group  + groupIDEexist(long id): boolean  + close(): void |

PersonDao et GroupDao implémentent respectivement les interfaces IPersonDao et IGroupDao.

**Étape 2 : La couche service**

La couche métier basée sur Spring. Cette couche prendre en charge l'accès et la modification des données métier.

Elle assure

* l'authentification des personnes,
* la modification des données personnelles,
* l'accès aux données.

|  |
| --- |
| **IPersonService** << Interface>> |
|  |
| + init(): void  + getPerson(): Person  + getAllPersons(): List<Person>  + personIDExists(long id): boolean  + updatePerson(Person p): void  + close(): void |

|  |
| --- |
| **IGroupService << Interface>>** |
|  |
| +init():void  +getAllGroups():Collections<Group>  +getContentGroup(long id):Collections<Person>  +getGroup(long id): Group  +groupIDEexist(long id):boolean  + close():void |

|  |
| --- |
| **PersonService** |
| - PersonDao: IPersonDao |
| + init(): void  + getPerson(): Person  + getAllPersons(): List<Person>  + personIDExists(long id): boolean  + updatePerson(Person p): void  + close(): void |

|  |
| --- |
| **GroupService** |
| - groupDao: IGroupDao |
| +init():void  +getAllGroups():Collections<Group>  +getContentGroup(long id):Collections<Person>  +getGroup(long id): Group  +groupIDEexist(long id):boolean  +close():void |

|  |
| --- |
| **IEmailService** |
|  |
| + sendEmail(String recipient, String subject, String content): void |

|  |
| --- |
| **EmailService** |
| - username: String // Adresse e-mail de l'expéditeur  - password: String // Mot de passe l'adresse e-mail de l'expéditeur  - prop: Properties // Config serveur smtp Gmail  - session: Session |
| + sendEmail(String recipient, String subject, String content):void |

|  |
| --- |
| User |
| - id  - password |
| \* getter et setter |

L'objet user représente l'utilisateur de l'application. Il permet l'authentification d'une personne.

|  |
| --- |
| **UserService** |
| - personDao: IPersonDao  - emailService: IEmailService |
| + connect(long login, String password):boolean  + sendUserPassword(long id): void //id de l'utilisateur |

|  |
| --- |
| **IUserService** |
|  |
| + connect(long login, Stringpassword):boolean  + sendUserPassword(long id):void //id de l'utilisateur |