LAB 4: QUARKUS PERSISTENT

Autor: José Díaz
Apoyo: Juan Ramirez

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

1. Abre el proyecto **expense-service**.
2. Colocar las dependencias

```
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-hibernate-orm</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-hibernate-orm-panache</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-jdbc-postgresql</artifactId>
        </dependency>
        <dependency>
            <groupId>io.quarkus</groupId>
            <artifactId>quarkus-hibernate-validator</artifactId>
        </dependency>
```

3. Decorar las entidades

```
@Entity

public class Expense {
```

```
@Entity
public class Associate {
```

4. Heredar de PanacheEntity

```
@Entity
public class Expense extends PanacheEntity {
```

```
@Entity
public class Associate extends PanacheEntity {
```

5. Agregar las relaciones en Expense.java

```
    // TODO: Add many-to-one relationship between expense and associate
    @JsonbTransient
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "associate_id", insertable = false, updatable = false)
    public Associate associate;
```

```
    // TODO: Annotate the associateId with @Column
    @Column(name = "associate_id")
    public Long associateId;


    // TODO: Add a no-argument constructor
    public Expense() {
    }
```

6. Agregar en el constructor de Expense.java associateId:

```
public Expense(UUID uuid, String name,
LocalDateTime creationDate,
                PaymentMethod paymentMethod, String
amount, Associate associate) {
    this.uuid = uuid;
    this.name = name;
    this.creationDate = creationDate;
    this.paymentMethod = paymentMethod;
    this.amount = new BigDecimal(amount);
    this.associate = associate;
```

```
    // TODO: Add associateId association
    this.associateId = associate.id;
}
```

7. Modificar en Expense.java el método of:

```
@JsonbCreator
public static Expense of(String name,
PaymentMethod paymentMethod, String amount, Long
associateId) {


    // TODO: Update regarding the new relationship
    return
Associate.<Associate>findByIdOptional(associateId)
            .map(associate -> new Expense(name,
paymentMethod, amount, associate))
            .orElseThrow(RuntimeException::new);
}
```

8. Agregar el método update():

```
// TODO: Add update() method
public static void update(final Expense expense)
throws RuntimeException {
```

```
    Optional<Expense> previousExpense =
Expense.findByIdOptional(expense.id);


    previousExpense.ifPresentOrElse(updatedExpense
-> {
        updatedExpense.uuid = expense.uuid;
        updatedExpense.name = expense.name;
        updatedExpense.amount = expense.amount;
        updatedExpense.paymentMethod =
expense.paymentMethod;
        updatedExpense.persist();
    }, () -> {
        throw new
WebApplicationException(Response.Status.NOT_FOUND)
;
    });
}
```

9. En Associate.java agregar las relaciones:

```
@JsonbTransient
```

```
@OneToMany(mappedBy = "associate", cascade =
CascadeType.ALL, fetch = FetchType.LAZY)

public List<Expense> expenses = new ArrayList<>();
```

10. Agregar en Associate el constructor sin args:

```
// TODO: Add a no-argument constructor

public Associate() {

}
```

11. Vamos al ExpenseResource.java para implementar el método list();

```
@GET

// TODO 1: Implement with a call to "listAll()" of
Expense entity.

// TODO 2: Add pagination and sort by "amount" and
"associateId".

public List<Expense> list(@DefaultValue("5")
@QueryParam("pageSize") int pageSize,

                          @DefaultValue("1")
@QueryParam("pageNum") int pageNum) {

   PanacheQuery<Expense> expenseQuery =
Expense.findAll(

           Sort.by("amount").and("associateId"));
```

```
    return expenseQuery.page(Page.of(pageNum - 1,
pageSize)).list();

}
```

12. Agregar el persist() en el POST:

```
@POST

// TODO: Make the method transactional

@Transactional

public Expense create(final Expense expense) {

    Expense newExpense = Expense.of(expense.name,
expense.paymentMethod,

            expense.amount.toString(),
expense.associateId);

    // TODO: Use the "persist()" method of the
entity.

    newExpense.persist();


    return newExpense;

}
```

13. Hacer transaccional el método DELETE:

```
@DELETE
@Path("{uuid}")
// TODO: Make the method transactional
@Transactional
public void delete(@PathParam("uuid") final UUID
uuid) {
   // TODO: Use the "delete()" method of the
entity and list the expenses
   long numExpensesDeleted =
Expense.delete("uuid", uuid);

   if (numExpensesDeleted == 0) {
       throw new
WebApplicationException(Response.Status.NOT_FOUND)
;
   }
}
```

14. Implementar el PUT transaccional:

```java
    @PUT
    // TODO: Make the method transactional
    @Transactional
    public void update(final Expense expense) {
        // TODO: Use the "update()" method of the entity.
        try {
            Expense.update(expense);
        } catch (RuntimeException e) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
    }
```

15. Agregar propiedades en el application.properties:

```
# TODO: Add configuration
quarkus.datasource.devservices.image-name=postgres:14.1
quarkus.hibernate-orm.database.generation=drop-and-create
```

16. Ejecutar **mvn quarkus:dev**

Deberíamos observar que se genera el contenedor para postgresql creado por dev services.

17.

enjoy!

Jose