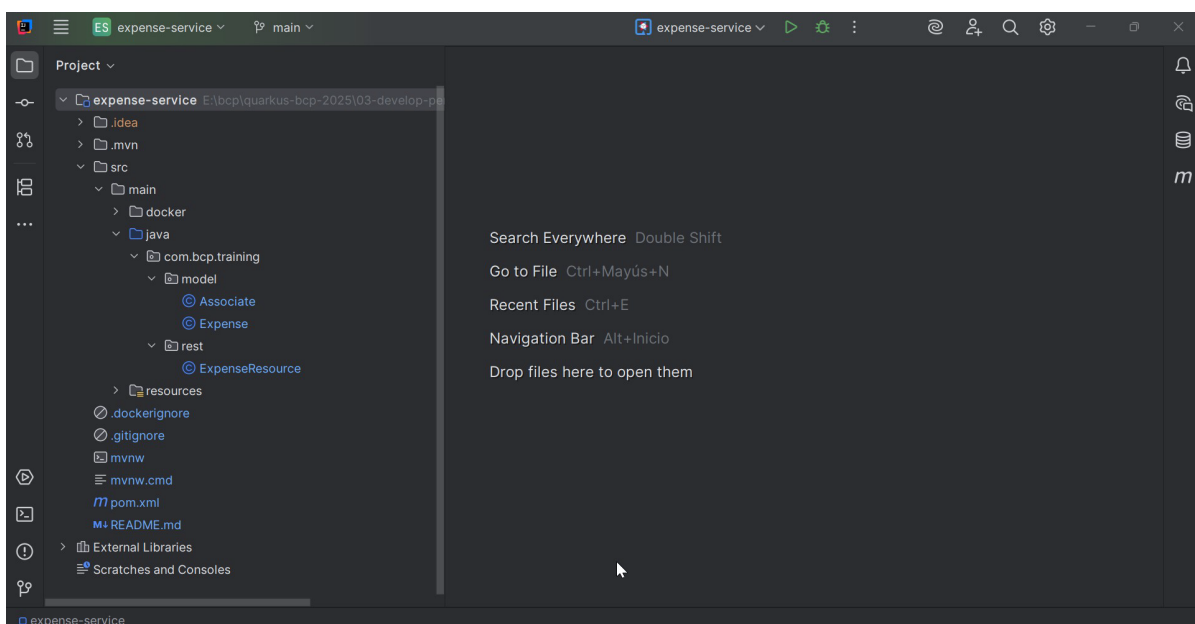


LAB 4: QUARKUS PERSISTENT

Autor: José Díaz

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

1. Abre el proyecto **expense-service**.

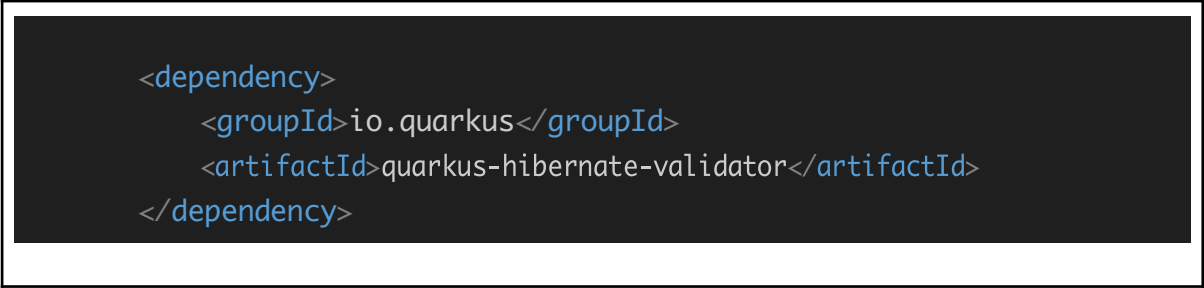


2. Agrega Panache y PostgreSQL JDBC a tu proyecto:

- 2.1 Agregar hibernate-orm-panache y jdbc-postgresql al proyecto.

```
mvn quarkus:add-extension -Dextensions="hibernate-orm-panache, jdbc-postgresql"
```

- 2.2 También vamos a necesitar quarkus-hibernate-validator:



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

- #### 4.1 Comienza la aplicación en modo desarrollo: `mvn quarkus:dev`

#### 4.2 Agrega la anotación @Entity a la clase Expense y Associate:

```
public class Expense {

@Entity

public class Associate {
```

4.3 Hacer que Expense y Associate hereden de `io.quarkus.hibernate.orm.panache.PanacheEntity` para obtener el soporte extendido ORM de Panache.

```
@Entity

public class Expense extends PanacheEntity {
```

```
@Entity

public class Associate extends PanacheEntity {
```

4.4 Agregar constructor sin argumentos a ambas clases:

```
public Associate() {
}
```

```
public Expense() {
}
```

4.5 Agregar un método static llamado update() en la clase **Expense** para modificar una entidad en la base de datos.

```
public static void update(final Expense expense) {
 Optional<Expense> previous = Expense.findByIdOptional(expense.id);
 previous.ifPresentOrElse(() -> {
 update.uuid = expense.uuid;
 update.name = expense.name;
 update.amount = expense.amount;
 update.paymentMethod = expense.paymentMethod;
 update.persist();
 }, () -> {
 throw new WebApplicationException(Response.Status.NOT_FOUND);
 });
}
```

5. Agregar una relación one-to-many entre las entidades Expense y Associate para crear las asociaciones relevantes.

5.1 Abrir la clase Associate y anotar la variable expenses list con la anotación @OneToMany. Agregar la anotación @JsonbTransient para hacer la variable expenses list transient para la serialización JSON.

El FetchType debería ser LAZY y el campo debería ser mapped by associate, que es el nombre del campo en la entidad Expense. La entidad Associate debería configurar cascade all todas las operaciones de la entidad Expense.

```
@Entity
public class Associate extends PanacheEntity {
 public String name;

 // TODO: Add one-to-many relationship between associate and
 // expenses
 @JsonbTransient
 @OneToMany(mappedBy = "associate", cascade = CascadeType.ALL,
 fetch = FetchType.LAZY)
 public List<Expense> expenses = new ArrayList<>();
}
```

5.2 Abre la clase Expense y configura las anotaciones para relación many-to-one. Agrega @ManyToOne sobre la variable associate. Agrega la anotación



@JsonbTransient para hacer la variable transient para JSON serialización y deserialización.

El FetchType debería ser LAZY. Agrega una anotación @JoinColumn con el nombre associate\_id, el cual define el nombre de columna creada en la tabla de base de datos para el associate ID. Y debido a que la columna associate\_id es un campo de referencia, marca este como insertable=false y updatable=False.

```
@JsonbTransient
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "associate_id", insertable = false, updatable = false)
public Associate associate;
```

5.3 Cómo has marcado la variable associate como transient para JSON serialización y deserialización, podemos usar un atributo separado para ver a que associate él expense pertenece. Anota el atributo associateId con @Column para hacer el associateId un campo de la entidad válido.

```
@Column(name = "associate_id")
public Long associateId;
```

Establece el associateId cada vez que el constructor es invocado:

```
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;
 this.associateId = associate.id;
}
```

5.4 Actualiza el método of() relacionado a la relación de Expense y Associate. En el nuevo método of() debería requerir un associate dado un associate ID y usar este para crear un nuevo expense. Si no hay associate para ese ID, entonces el método debería arrojar un RuntimeException.

```
@JsonbCreator
public static Expense of(String name, PaymentMethod paymentMethod,
String amount,
 Long associateId) {
 return Associate.<Associate>findByIdOptional(associateId)
 .map(associate -> new Expense(name, paymentMethod, amount,
associate))
 .orElseThrow(RuntimeException::new);
}
```

6. Modifica la implementación de la clase ExpenseResource para usar la capacidad de persistencia de la entidad Expense.

6.1 Examina el src/main/resources/import.sql. Quarkus ejecuta este archivo, después de inicializar la base de datos.

```
insert into Associate (id, name)
values (1, 'Jaime');
insert into Associate (id, name)
values (2, 'Pablo');

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Desk',
'0', '150.50', 1);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Online
Learning', '1', '75.00', 1);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Books',
'0', '50.00', 1);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Internet',
'1', '20.00', 1);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Phone',
'0', '15.00', 1);
```

```
insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Bookshelf',
'0','150.50', 1);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Printer
Cartridges', '1','15.00', 2);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Online
Learning', '0','50.00', 2);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Internet',
'1','20.00', 2);

insert into Expense (uuid, id, name, paymentmethod, amount,
associate_id)
values (gen_random_uuid(), nextval('Expense_SEQ'), 'Phone',
'0','15.00', 2);
```

6.2 Abre la clase ExpenseResource y actualiza el método list() con una llamada a Expense.listAll().

```
@GET
public List<Expense> list() {
 return Expense.listAll();
}
```

6.3 Agrega la anotación @Transactional a los métodos que modifican entidades: create, delete y update.

```
@POST
@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.

 return newExpense;
}
```

```
@DELETE
@Path("/{uuid}")
@Transactional
public void delete(@PathParam("uuid") final UUID uuid) {
 // TODO: Use the "delete()" method of the entity and list the
 expenses
}

@PUT
@Transactional
public void update(final Expense expense) {
 // TODO: Use the "update()" method of the entity.
}
```

6.4 Agregar la llamada `newExpense.persist()` en el método `create()`.

```
@POST
@Transactional
public Expense create(final Expense expense) {
 Expense newExpense = Expense.of(expense.name,
 expense.paymentMethod,
 expense.amount.toString(), expense.associateId);
 newExpense.persist();

 return newExpense;
}
```

6.5 Agrega las llamadas `Expense.delete()` y `Expense.listAll()` en el método `delete()`.

```
@DELETE
@Path("/{uuid}")
@Transactional
public void delete(@PathParam("uuid") final UUID uuid) {
 long numExpensesDeleted = Expense.delete("uuid", uuid);
 if (numExpensesDeleted == 0) {
 throw new WebApplicationException(Response.Status.NOT_FOUND);
 }
}
```

6.6 Agrega la llamada a `Expense.update(expense)` en el método `update()`.

```
@PUT
@Transactional
public void update(final Expense expense) {
 try {
 Expense.update(expense);
 } catch (RuntimeException e) {
 throw new WebApplicationException(Response.Status.NOT_FOUND);
 }
}
```

## 7. Probar el servicio Expense.

7.1 Abre el navegador web y navega a el endpoint de Swagger UI de la aplicación en <http://localhost:8080/q/swagger-ui>.

7.2 Clic a la línea POST y luego clic en Try it out.

7.3 Reemplaza el ejemplo del request body con el siguiente JSON y clic en Execute.

```
{
 "amount": 100,
 "name": "Chair",
 "paymentMethod": "CASH",
 "associateId": 2
}
```

Verifica que la respuesta del servidor corresponde al HTTP response code 200.

7.4 Oculta el bloque POST y haz clic en la línea GET para expandir su correspondiente bloque.

7.5 Clic en Try it out y luego clic en Execute.

7.6 Verifica que la respuesta incluye la nueva data.

## 8. Agrega paging y sorting al listado de data Expense. Configura el tamaño de página por defecto a 5. También el ordenamiento por los atributos amount y associateId.

8.1 Elimina la línea return Expense.listAll() del método list() y agrega Expense.findAll() para usar y habilitar la paginación.

```
@GET
public List<Expense> list() {
 PanacheQuery<Expense> expenseQuery = Expense.findAll();
 return Expense.listAll();
}
```

8.2 Agrega los parámetros de paginación como query parameters y úsalos en la llamada a `expenseQuery.page()`. Hay dos parámetros en el siguiente ejemplo, page number y page size.

```
@GET
public List<Expense> list(@DefaultValue("5") @QueryParam("pageSize")
int pageSize,
 @DefaultValue("1") @QueryParam("pageNum")
int pageNum) {
 PanacheQuery<Expense> expenseQuery = Expense.findAll();
 return expenseQuery.page(Page.of(pageNum - 1, pageSize)).list();
}
```

8.3 Agrega el ordenamiento para amount y associateId.

```
@GET
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();
}
```

8.4 En tu terminal windows, ejecuta el siguiente comando para ver la lista de expense paginados.

```
[student@workstation ~]$ curl \
 'http://localhost:8080/expenses?pageNum=1&pageSize=5' | jq
[
 {
 "id": 5,
 "amount": 15.00,
 "associateId": 1,
 "name": "Phone",
 "paymentMethod": "CASH"
 },
 ...output omitted...
 {
 "id": 9,
 "amount": 20.00,
 "associateId": 2,
 "name": "Internet",
 "paymentMethod": "CREDIT_CARD"
 }
]
```

Linux o MacOSX:

```
curl 'http://localhost:8080/expenses?pageNum=1&pageSize=5' | jq
```

PowerShell:

```
Invoke-RestMethod 'http://localhost:8080/expenses?pageNum=1&pageSize=5' |
ConvertTo-Json -Depth 10
```

Listo. Hemos terminado.

Enjoy!

Jose

