

Desarrollo de micro servicios cloud-native con Quarkus

Persistencia de Datos con Panache

Configurando persistencia

```
<dependencies>
  <!-- Hibernate ORM dependency -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-hibernate-orm</artifactId>
  </dependency>
</dependencies>
```

```
<dependencies>
  <!-- Panache dependency -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-hibernate-orm-panache</artifactId>
  </dependency>
</dependencies>
```

Agregando el driver

```
<dependencies>
  <!-- JDBC driver dependency -->
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-jdbc-postgresql</artifactId>
  </dependency>
</dependencies>
```

Configurando la base de datos

```
# datasource configuration
quarkus.datasource.username = myusername 1
quarkus.datasource.password = mypassword 2
quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/mydatabase 3

# Optional configuration
quarkus.hibernate-orm.sql-load-script = META-INF/import-dev.sql 4
quarkus.hibernate-orm.database.generation = drop-and-create 5
```

Usando profiles: dev y prod

```
# production datasource configuration
%prod.quarkus.datasource.username = myusername
%prod.quarkus.datasource.password = mypassword
%prod.quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/mydatabase
%prod.quarkus.hibernate-orm.sql-load-script = no-file 1

# development datasource configuration
%dev.quarkus.hibernate-orm.sql-load-script = META-INF/import-dev.sql
%dev.quarkus.hibernate-orm.database.generation = drop-and-create
%dev.quarkus.datasource.devservices.image-name = quay.io/example/postgres:14.1 2
```


Persistencia con Hibernate ORM

```
@ApplicationScoped
public class EmployeeService {
    @Inject
    EntityManager em; ❶

    @Transactional ❷
    public void createEmployee( String name ) {
        Employee employee = new Employee();
        employee.setName( name );
        em.persist( employee ); ❸
    }
}
```

REQUIRES_NEW

MANDATORY

Persistencia con Hibernate ORM

```
@Transactional( TxType.REQUIRES_NEW )  
public void createEmployee( String name ) {  
    Employee employee = new Employee();  
    employee.setName( name );  
    em.persist( employee );  
}
```


Simplificando la persistencia con **Panache**

Patrón Repository

```
@Entity
public class Expense {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private BigDecimal amount;
    private String description;
    private LocalDateTime creationDate;

    // Getters and Setters
}
```

```
@ApplicationScoped
public class ExpenseRepository implements PanacheRepository<Expense> {
}
```

```
@Inject
ExpenseRepository expenseRepository;

Expense expense = new Expense();
expense.setName( "Hotel stay" );
expense.setAmount( 100 );
expense.setDescription( "Conference travel" );
expense.setCreationDate( LocalDateTime.now() );

expenseRepository.persist( expense );
```

Simplificando la persistencia con **Panache**

Active Record Pattern

```
@Entity
public class Expense extends PanacheEntity {
    public String name;
    public BigDecimal amount;
    public String description;
    public LocalDateTime creationDate;
}
```

```
Expense expense = new Expense();
expense.name = "Hotel stay";
expense.amount = 100;
expense.description "Conference travel";
expense.creationDate = LocalDateTime.now();

expense.persist();
```

```
@Entity
public class Expense extends PanacheEntity {
    public String name;
    public BigDecimal amount;
    public String description;
    public LocalDateTime creationDate;

    public static List<Expense> findCurrent(){
        return list( "creationDate", LocalDateTime.now() );
    }
}
```


Métodos provistos por Panache

```
// Persist the entity
instance.persist();

// check if the entity is persistent
instance.isPersistent()

// get a list of all entities
List<Entity> allEntitys = Entity.listAll();

// find a specific entity by ID
instance = Entity.findById( entityId );

// find a specific instance by ID via an Optional
Optional<Entity> optional = Entity.findByIdOptional( entityId );

// find all alive entities
List<Entity> aliveInstances = Entity.list( "alive", true );

// count all entities
long countAll = Entity.count();

// count all alive entities
long countAlive = Entity.count( "alive", true );

// delete all alive entities
Entity.delete( "alive", true );

// delete all entities
Entity.deleteAll();

// delete by id
boolean deleted = Entity.deleteById( entityId );

// set all alive entities as not alive
Entity.update( "alive = false where alive = ?1", true );
```

Paging y Sorting Query Results

Panache Paging Functions

Function	Usage
<code>page()</code>	Returns the current page.
<code>page(Page page)</code>	Sets and returns the specified page.
<code>page(int pageIndex, int pageSize)</code>	Shorthand for the previous method.
<code>pageCount()</code>	Returns the number of pages for the current page size.
<code>range(int startIndex, int lastIndex)</code>	Retrieves the results between <code>startIndex</code> and <code>lastIndex</code> .

```
@GET
public List<Example> list() {
    PanacheQuery<Example> exampleQuery = Example.findAll();
    return exampleQuery.page(Page.of(0, 10)).list();
}
```


Paging y Sorting Query Results

Panache Sorting Functions	
Function	Usage
by(String column)	Sorts by the given column in ascending order.
Function	Usage
by(String column, Direction direction)	Sorts by the column using the specified direction (Ascending or Descending).
by(String... columns)	Sorts by the given columns in the given order.
and(String column)	Adds an additional sorting column in ascending order.
and(String column, Direction direction)	Adds an additional sorting column with the given direction.
empty()	Returns an empty sorting instance.
ascending()	Sets the ascending order for the current sort columns.
ascending(String... columns)	Sets the columns to sort in ascending order.
descending()	Sets the descending order for the current sort columns.
descending(String... columns)	Sets the columns to sort in descending order.

```
@GET
public List<Example> list() {
    return Example.findAll( Sort.by("sample") ).list();
}
```

Demo