UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO.

DGTIC

Diplomado Desarrollo de Sistemas con tecnología JAVA

Modulo 8: Avance de Proyecto

Sanvicente Enríquez Daniela Alejandra

Edición 14

Ciclo 2024

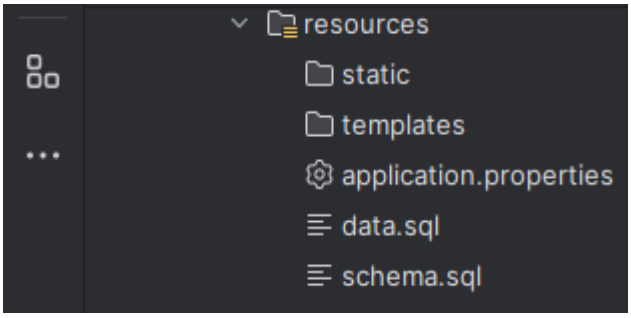
Objetivo:

Implementar lo aprendido con respecto a Spring Data en su proyecto final para el manejo de persistencia con Spring Data.

Instrucciones:

1. Tener al menos cinco tablas para el proyecto:
   i. Tablas de catalogo
   ii. Tablas de relación
   iii. Incluir en el pdf un Diagrama Entidad Relación
   iv. Incluir scripts sql para crear el esquema y el estado de la bd

En resources añadimos los archivos data.sql y schema.sql



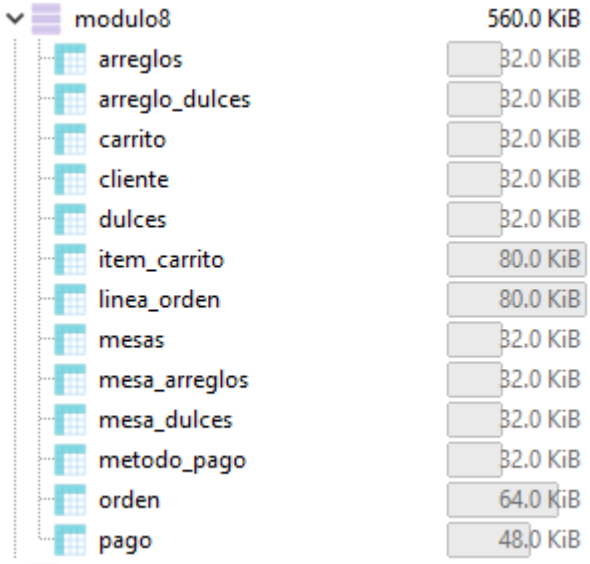
Se creo la clase de la aplicación con el bean del dataSource

```java
@SpringBootApplication
public class M8E1Application {

    public static void main(String[] args) { SpringApplication.run(M8E1Application.class, args); }


    @Bean
    public DataSource dataSource(){ //Establece la comunicación con la base de datos
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.mariadb.jdbc.Driver");
        dataSource.setUrl("jdbc:mariadb://localhost:3306/modulo8");
        dataSource.setUsername("root");
        dataSource.setPassword("Mar14D3");
        return dataSource;
    }
```

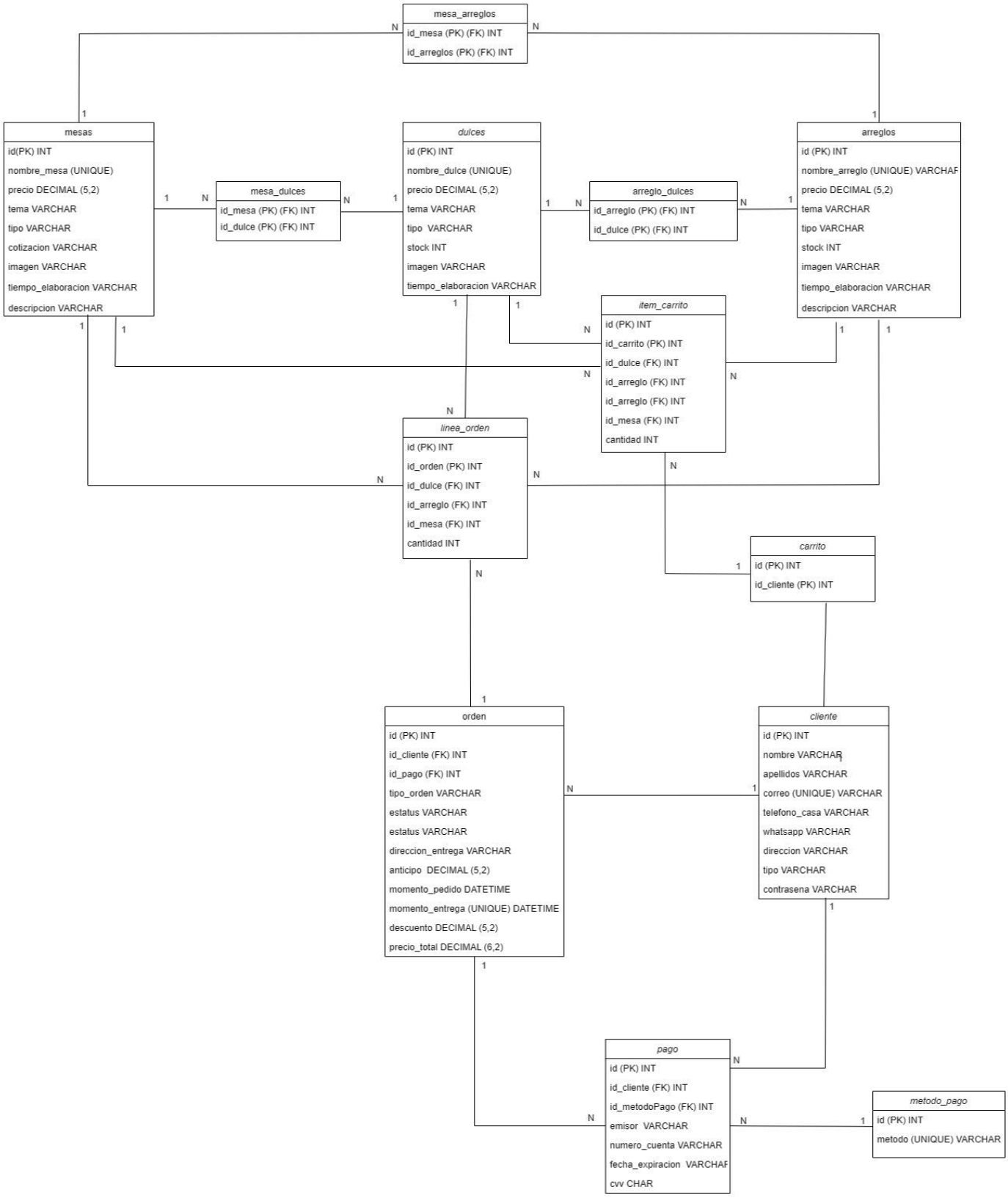
En cada clase con la anotación @SQL corremos los scripts data y schema antes de las pruebas unitarias

```java
@SpringBootTest //@SQL corre los scripts
@Sql({"/schema.sql", "/data.sql"})
//Crea el esquema para nuestro banco de datos modulo8 y añade datos a sus tablas
```

En HeidiSQL se visualizan las tablas creadas

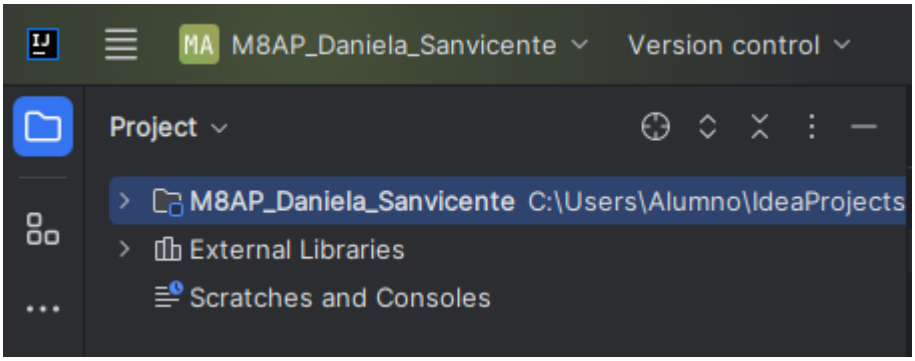


El Diagrama Entidad Relación es el siguiente:



2. Se debe crear un proyecto Spring nuevo con las siguientes características:
   i. Nombre del proyecto:
   ii. M8AP<PrimerNombre> _<PrimerApellido>
   iii. Ejemplo: M8AP_Omar_Mendoza

Creamos nuestro proyecto con el nombre M8AP_Daniela_Sanvicente.



En el pom.xml

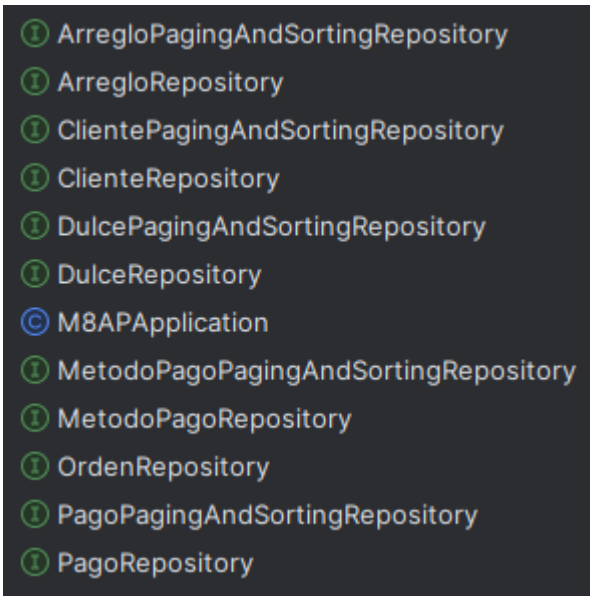
```
<groupId>mx.unam.dgtic.datos</groupId>
<artifactId>M8AP_Daniela_Sanvicente</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>M8AP</name>
<description>Avance de Proyecto para el Modulo 8 Spring Datos</description>
```

3. Con soporte JPA y MariaDb

En el pom.xml se ven las dependencias necesarias

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.mariadb.jdbc</groupId>
        <artifactId>mariadb-java-client</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

4. Debe implementar las clases e interfaces relacionadas con
   i. CrudRepository
   ii. PagingAndSortingRepository



5. Implementar relaciones entra las tablas

Se crean las clases tomando en cuenta las relaciones que tienen entre si, colocando anotaciones para las llaves primarias (@Id)e incrementales (@GeneratedValue), en caso de que el nombre sea diferente en SQL (@Column), si son unique (@Column (unique =true)) y si el nombre es diferente con llaves extranjeras (@JoinColumn).

i. ManyToOne

Para las relaciones ManyToOne como colocamos la anotación @ManyToOne

```java
package mx.unam.dgtic.datos.entidades;

import ...

@Entity
@Table(name = "orden")
public class Orden {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "id_cliente", referencedColumnName = "id", nullable = false)
    private Cliente cliente;

    @ManyToOne
    @JoinColumn(name = "id_pago", referencedColumnName = "id", nullable = false)
    private Pago pago;
```

ii. OneToMany

Para las relaciones OneToMany como colocamos la anotación @OneToMany (mappedBy), dónde mappedBy es para que Spring no cree una tabla para relacionar las dos entidades, diciendo que ya está relacionado por el lado dueño que es el lado con la llave foránea (que en el caso de la captura es la entidad(clase de java que sirven como mapeo para interactuar con la base de datos, desde un programa java) producto)

```java
package mx.unam.dgtic.datos.entidades;

import ...

@Entity
@Table(name = "metodo_pago")
public class MetodoPago {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @Column(nullable = false, unique = true)
    private String metodo;

    @OneToMany(mappedBy = "metodoPago", cascade = CascadeType.ALL)
    private List<Pago> pagos;

    public MetodoPago() {
    }
```

iii. ManyToMany

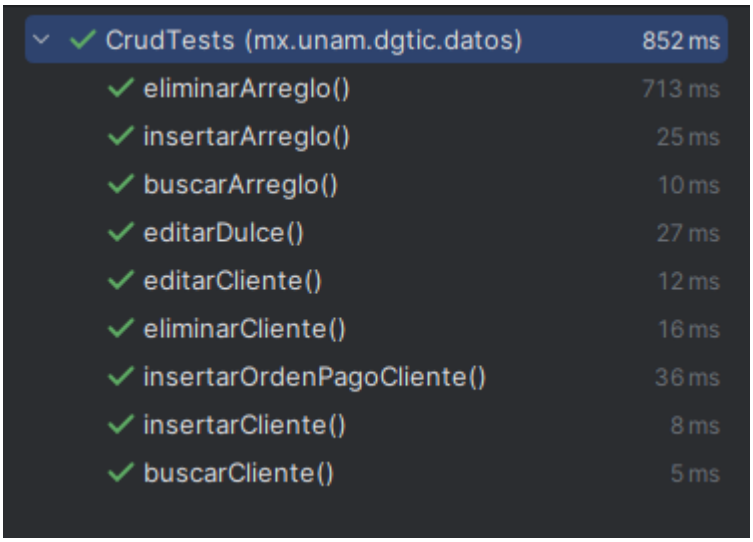
Para las relaciones ManyToMany como colocamos la anotación @ManyToMany. Las relaciones ManyToMany necesitan una tabla asociativa (en el caso de la captura es definida por la anotación @JoinTable con el valor venta_producto), con ambas las direcciones de la asociación de esta nueva tabla (definidas por joinColumns y inverseJoinColumns).

```
@Column(nullable = false, columnDefinition = "VARCHAR(300) DEFAULT 'N/A'")
private String descripcion;

@ManyToMany
@JoinTable(
        name = "arreglo_dulces",
        joinColumns = @JoinColumn(name = "id_arreglo"),
        inverseJoinColumns = @JoinColumn(name = "id_dulce")
)
private List<Dulce> dulces;

@ManyToMany(mappedBy = "arreglos")
private Set<Mesa> mesas;

public Arreglo() {
}
```

6. Implementar operaciones CRUD para diferentes tablas de catalogo y de relación

En CrudTests vemos los métodos para las operaciones CRUD de cliente y producto siguientes:

Create (insertar)
Read (buscar)
Update (editar)
Delete (eliminar)

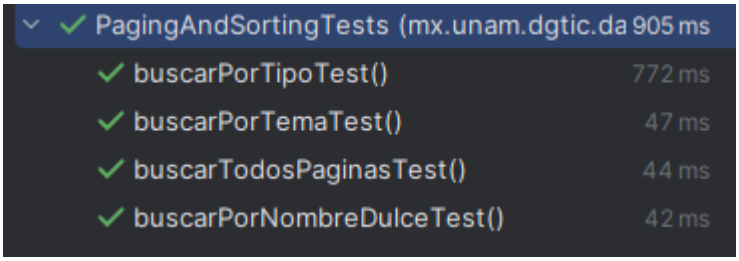


Para el para el equema y estado de las tablas se utilizan las anotaciones @SQL y @Test

```
package mx.unam.dgtic.datos;

import ...

@SpringBootTest
// prepara esquema de las tablas
@Sql({"/schema.sql", "/data.sql"})
public class CrudTests {
```

7. Implementar metodos de CrudRepository y PagingAndSortingRepository

Los test de PagingAndSorting de “buscarPor” dividen los dulces en páginas, en este caso de 3 en 3 o según lo queramos en el atributo PAGE_SIZE, y los ordenamos según uno de sus atributos, en este caso se probaron con los atributos nombreDulce, tema y tipo de la entidad Dulce. El test “buscarTodosPaginasTest” recoge todos los registros de dulces de nuestra tabla del banco de datos, y hace la paginación de acuerdo con el atributo static PAGE_SIZE definido anteriormente dentro de la página.

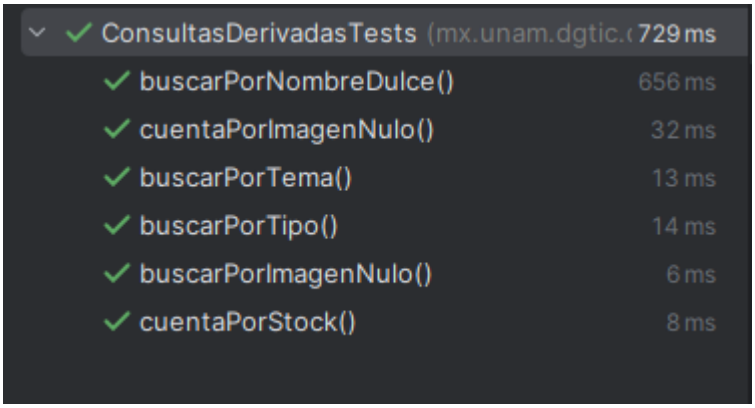


En la siguiente captura se puede observar como se implementó y utilizó CrudRepository

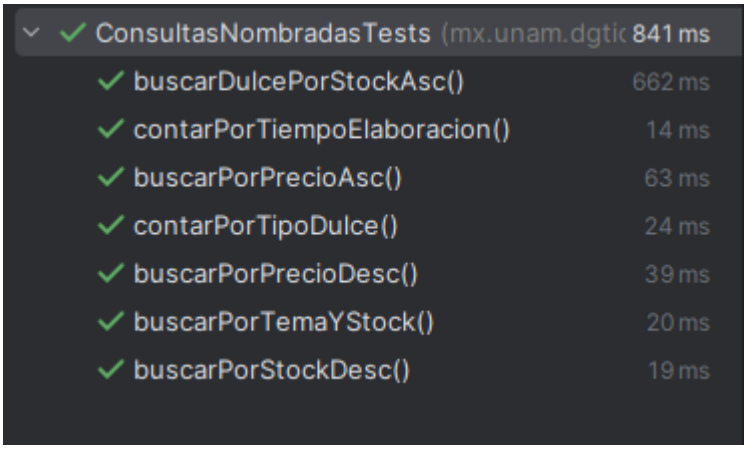
```java
@Repository
public interface DulceRepository extends CrudRepository<Dulce, Integer> {
    //------Consultas Derivadas----------
    List<Dulce> findByNombreDulce(String nombreDulce);
    List<Dulce> findByTema(String tema);
    List<Dulce> findByTipo(String tipo);

    List<Dulce> streamByImagenIsNull();

    Long countByStock(Integer stock);
    Long countByImagenIsNull();
```

8. Implementar consultas sobre las tablas

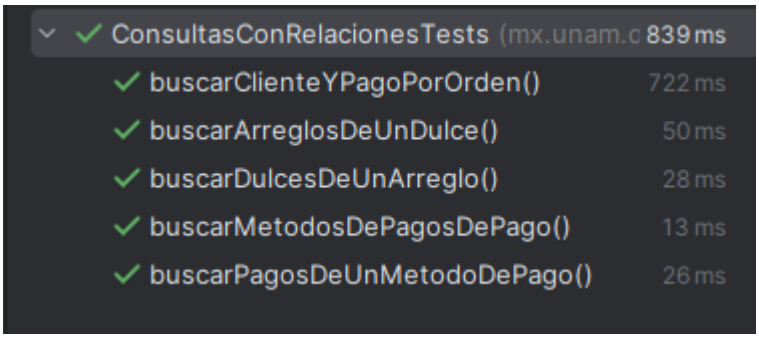
i. Consultas derivadas (al menos 5)



ii. Consultas nombradas (al menos 5)



iii. Consultas con relaciones (al menos 5)



9. Probar las implentaciones en pruebas unitarias organizadas por tema (diferentes clases de prueba para cada tema que incluyen varias pruebas unitarias).

En la siguiente captura se puede observar la organización de las pruebas.