DOCUMENTACION PROYECTO SPRING CV2

Mi proyecto esta hecho con el framework Spring, este consiste en una web que utilizan los supervisores de una empresa de supermercados, para controlar tanto el personal de la empresa como los productos y clientes que tiene. Cada campo va referenciado en una tabla(Supervisores, Empleados, etc…) con toda la información necesaria para cada situación, estas tablas estas relacionadas unas con otras de alguna forma. Para este proyecto ademas de el framework nos apoyaremos de un servidor en este caso local para establecer conexión con una BD en MySQL, el servidor sera Xampp y el programa MySQLWorkbench para realizar la creación de la BD y alguna edición ocasional dentro de ella, puesto que la mayor parte de la gestión de la BD la haremos dentro de nuestro proyecto Spring.

Para este proyecto sera necesaria las siguientes dependencias:

```
<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-thymeleaf</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>com.mysql</groupId>
                <artifactId>mysql-connector-j</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
```

Definimos nuestro applpicaction.properties para configurar la conexión a nuestra BD y algunas configuraciones adicionales como *spring.jpa.hibernate.ddl-auto=update*, que nos servirá para que sea nuestro programa el que cree las tablas dentro de la BD y las relaciones necesarias que configuremos

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/cv2
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql: true
```

A continuación se muestra una clase a modo de ejemplo usando diferentes anotaciones para la creación de una tabla:
@Entity -> Anotacion usada para especificar que la clase es una entidad(hace referencia a una tabla)
@Table -> Se utiliza para mapear una clase de entidad a una tabla en la BD
@Id-> Anotacion para especificar que el campo sera una clave primaria
@GeneratedValue-> Tipo de generación de la clave primaria
@Column-> Usamos esta anotacion para dar nombre dentro de nuestra tabla en caso que no querramos que sea igual a la variable a la que hace referencia, aunque esta anotacion tiene mas funciones
@ManyToOne-> Anotacion para especificar que el siguiente campo hace referencia a una clave primaria en otra tabla en relación n-1
@JoinColumn-> Anotacion con la que realizamos la union de dos tablas a traves de una clave primaria (id)
@OneToMany-> Anotacion para especificar que el siguiente campo hace referencia a una clave primaria en otra tabla en relación 1-n

```
@Entity
@Table(name = "empleado")
public class Empleado implements Serializable {

    private static final Long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_empleado", unique = true, nullable = false)
    private Long id_empleado;

    @Column(name = "nombre", nullable = false)
    private String nombre;

    @Column(name = "apellido", nullable = false)
    private String apellido;

    @Column(name = "telefono", nullable = false)
    private int telefono;

    //id_supervisor n-1
    @ManyToOne
    @JoinColumn(name = "id_supervisor")
    private Supervisor supervisor;

    //id_cliente 1-n
    @OneToMany(mappedBy = "empleado", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Cliente> clientes;

    //id_producto 1-n
    @OneToMany(mappedBy = "empleado", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Producto> productos;
```

Para acceder a los datos que tengamos en nuestras tablas sera necesario crear un repositorio con las consultas que vayamos a utilizar, para ello crearemos una interfaz que extienda de la clase CrudRepository la cual tiene predefinidas consultas básicas(CRUD) y a parte de ello nosotros crearemos a traves de JPQL usando la anotacion @Query

```
public interface EmpleadoRepository extends CrudRepository<Empleado, Long> {

    @Query("SELECT e FROM Empleado e WHERE e.id_empleado = :id")
    Empleado encontrarEmpleadoId(@Param("id") Long id);

    @Query("SELECT s FROM Supervisor s JOIN s.empleados e WHERE e.id_empleado = :empleadoId")
    Supervisor findSupervisorByEmpleadoId(@Param("empleadoId") Long empleadoId);

}
```

Para ver este contenido en nuestra web debemos antes crearnos un controlador que gestione las acciones que tomemos dentro de esta, para ello la definiremos por métodos, definiendo como @GetMapping aquellas acciones en las que el servidor envíe información que muestre al cliente(Pagina web) y @PostMapping las acciones en las que sea el servidor el que haga de receptor y el cliente nos envíe la información, Definimos métodos tanto para acceder a cada una de las tablas, como para interactuar con ellas, ya sea para ver mas a detalle un campo, crear dentro de una tabla un campo nuevo o eliminarlo

```java
@Controller
public class Controlador {

    @Autowired
    private ClienteRepository clienteRepository;
    @Autowired
    private EmpleadoRepository empleadoRepository;
    @Autowired
    private ProductoRepository productoRepository;
    @Autowired
    private SuperRepository superRepository;
    @Autowired
    private SupervisorRepository supervisorRepository;

    @GetMapping("/")
    public String inicio() {...3 lines }

    //METODOS PARA INGRESAR A CADA UNA DE LAS TABLAS
    @GetMapping("/Cliente")
    public String cliente(Model modelo) {...5 lines }

    @GetMapping("/Empleado")
    public String empleado(Model modelo) {...5 lines }

    @GetMapping("/Producto")
    public String producto(Model modelo) {...5 lines }

    @GetMapping("/Super")
    public String ubicacion(Model modelo) {...5 lines }

    @GetMapping("/Supervisor")
    public String supervisor(Model modelo) {...5 lines }

    //METODOS PARA VER MAS A DETALLE CADA UN CAMPO ESPECIFICO
    @GetMapping("/Ver/Cliente/{id_cliente}")
    public String verCliente(Cliente cliente, Empleado empleado, Model modelo) {...7 lines }

    @GetMapping("/Ver/Empleado/{id_empleado}")
    public String verEmpleado(Empleado empleado, Supervisor supervisor, Model modelo) {...7 lines }

    @GetMapping("/Ver/Producto/{id_producto}")
    public String verProducto(Producto producto, Empleado empleado, Model modelo) {...7 lines }

    @GetMapping("/Ver/Super/{id_super}")
    public String verSuper(Super supermercado, Supervisor supervisor, Model modelo) {...7 lines }
```

Y para finalizar con el proyecto debemos crear la vista que tendrá nuestro proyecto o sea el html que se mostrara en nuestra pagina web, usaremos la tecnología thmeleaf para comunicar nuestro código html con el del controlador para mostrar o recibir información, a continuación se muestra una de las paginas para ver su contenido:

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Clientes</title>
        <!-- Agregar Bootstrap CSS -->
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <!-- Agregar jQuery y Bootstrap JS -->
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
    </head>
    <body>
        <div class="container">
            <h1 class="text-center">Clientes</h1>
            <table class="table">
                <thead>
                    <tr>
                        <th>Nombre</th>
                        <th>Apellidos</th>
                        <th><a th:href="@{/AgregarC}">+</a></th>
                    </tr>
                </thead>
                <tbody>
                    <tr th:each="cliente : ${clientes}">
                        <td th:text="${cliente.nombre}"></td>
                        <td th:text="${cliente.apellido}"></td>
                        <td> <a th:href="@{/Ver/Cliente/} + ${cliente.id_cliente}">Ver Mas...</a></td>
                    </tr>
                </tbody>
            </table>
        </div>
    </body>
</html>
```

Ya configurada cada una de las clases html que necesitemos podemos iniciar nuestro proyecto y visualizarlo en nuestro navegador con la ruta por defecto localhost:8080(El puerto podemos cambiarlo en caso de conflicto dentro del application.properties).

Roy Denys Calderon Manrique

Clientes

Nombre	Apellidos	+
Daniel	Sánchez	Ver Mas...
Sergio	Martín	Ver Mas...
Lucía	Díaz	Ver Mas...
Adrián	Rodríguez	Ver Mas...
Sara	González	Ver Mas...
Javier	Fernández	Ver Mas...
Marta	Pérez	Ver Mas...
Pablo	Martínez	Ver Mas...
Julia	Sánchez	Ver Mas...
Hugo	Núñez	Ver Mas...
Clara	Martín	Ver Mas...
Alejandro	Díaz	Ver Mas...
Valeria	González	Ver Mas...
Rodrigo	Fernández	Ver Mas...
Alicia	Jiménez	Ver Mas...
Eric	Gómez	Ver Mas...
Castro	Fern	Ver Mas...