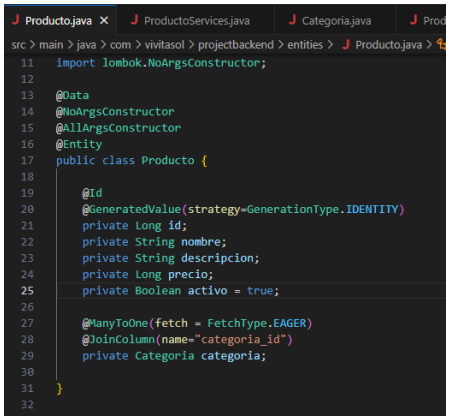
REACT - SPRINGBOOT

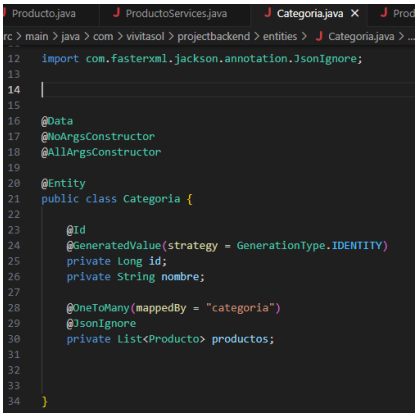
Optimizando Backend

Package Entities

Entities Producto:

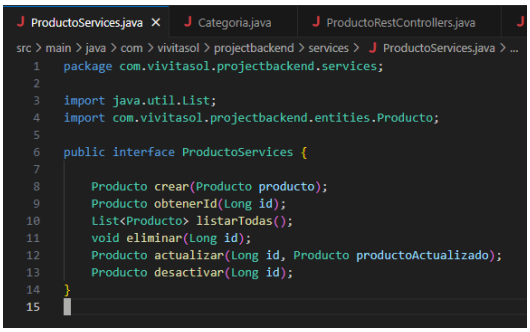
- Añadiremos atributo activo de tipo Boolean, aplicado para dar de baja un producto sin necesidad de eliminar físicamente el producto.
- Aplicaremos @JsonIgnore en la clase Categoría para el despliegue de productos integrando la categoría.





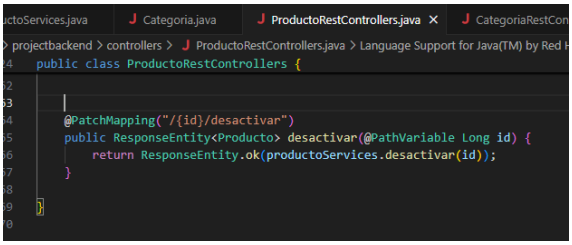
Package Services:

- Definimos método desactivar, implementado en la clase del servicio.



Package RestControllers:

- Implementamos petición patch() para modificar el estado del producto a través del servicio.



En la base de datos cargamos una nueva data: **base.sql**

Optimizando FrontEnd

Componente App.jsx: agregamos id en el path de editar.

```jsx
return (
  <Router>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/contacto" element={<Contacto />} />
      <Route path="/inventario" element={<Inventario />} />
      <Route path="/productos" element={<Productos />} />
      <Route path="/crear-producto" element={<CrearProducto />} />
      <Route path="/editar-producto/:id" element={<EditarProd />} />
    </Routes>
  </Router>
)
```

Componente productos.jsx:

- Mostramos los productos del backend. Al desactivar o dar de baja un producto, el despliegue se actualiza dentro del mismo componente. Se modifico el componente agregando el método cargarProductos(), el cual es llamado a través de useEffect().

- Se ha creado el método handleDesactivar(), el cual modifica el estado del producto y actualizando el componente.

Productos.jsx

```jsx
import './Productos.css';

export function Productos() {

    const [productos, setProductos] = useState([]);

    const cargarProductos = async () => {
        try {
            const response = await fetch('http://localhost:8080/api/productos');
            if (!response.ok) {
                throw new Error('Error en la respuesta del servidor');
            }

            const data = await response.json();
            console.log("Respuesta del backend:", data);

            const productosNormalizados = data.map(p => ({
                ...p,
                activo: p.activo === true || p.activo === 'true',
            }));

            setProductos(productosNormalizados);
        } catch (error) {
            console.error('Error al obtener los productos:', error);
        }
    };

    useEffect(() => {
        cargarProductos();
    }, []);

    const handleDesactivar = (id, nombre) => {
```

```jsx
useEffect(() => {
    cargarProductos();
}, []);

const handleDesactivar = (id, nombre) => {
    if (window.confirm(`¿Estás seguro de desactivar el producto "${nombre}"?`)) {
        fetch(`http://localhost:8080/api/productos/${id}/desactivar`, {
            method: 'PATCH'
        })
            .then(response => {
                if (!response.ok) {
                    throw new Error('Error al desactivar el producto');
                }
                return response.json();
            })
            .then(data => {
                console.log('Producto desactivado:', data);
                alert('Producto desactivado exitosamente');
                cargarProductos();
            })
            .catch(error => {
                console.error('Error al desactivar:', error);
                alert('Error al desactivar el producto');
            });
    }
};

return (
    <>
```

- Modificamos el estilo de los productos que estén desactivados.
- Los productos que estén activos, podrán ser editados, direccionando al componente y enviando el id.

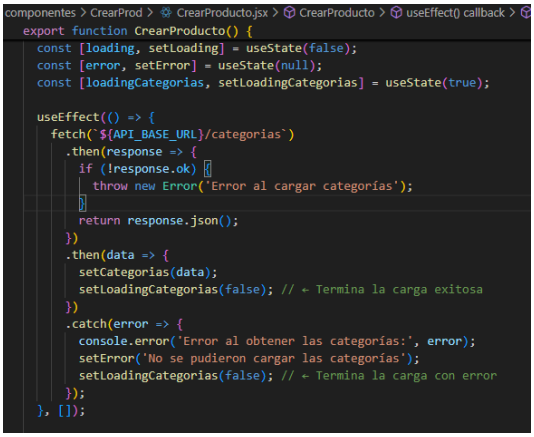
```jsx
<td>
    {prod.activo ? (
        <Link className="btn btn-outline-primary"
            style={{ fontSize: '13px' }}
            to={`/editar-producto/${prod.id}`}>
            Editar Producto
        </Link>
    ) : (
        <button className="btn btn-outline-secondary"
            style={{ cursor: 'not-allowed',
```

```jsx
        </tr>
    </thead>
    <tbody>
        {productos.map((prod) => (
            <tr key={prod.id}
                style={{ opacity: prod.activo ? 1 : 0.5,
                    backgroundColor: prod.activo ? 'white' : '#f8f8f8'}}>
                <td>{prod.id}</td>
                <td>{prod.nombre}</td>
                <td>{prod.descripcion}</td>
                <td>{prod.precio}</td>
                <td>
                    {prod.activo ? (
                        <Link className="btn btn-outline-primary"
                            style={{ fontSize: '13px' }}
                            to={`/editar-producto/${prod.id}`}>
                            Editar Producto
                        </Link>
                    ) : (
                        <button className="btn btn-outline-secondary"
                            style={{ cursor: 'not-allowed',
                                pointerEvents: 'none',}}
                            disabled>
                            Editar Producto
                        </button>
                    )}
                </td>
                <td>
                    {prod.activo ? (
                        <button
                            className="btn btn-sm btn-outline-danger"
```

Componente crearProducto:

En este componente debemos codificar el acceso a RestController de Categorías y de Productos.

- Se define const de tipo producto, por medio de useState(), la cual será modificada una vez que el objeto se crea en el formulario.
- Se implementa método que nos devuelve las categorías registradas en el backend.



- Método handleChange(): permite capturar los cambios generados en el formulario;

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setProducto(prev => ({
    ...prev,
    [name]: value
  }));
};
```

- e.targe: input que genera el evento.
- {name, value}: extrae el name y el value del input.
- setProducto(): actualiza el estado de un objeto, reemplaza el objeto con los valores ingresados en el formulario.

- Método handleSubmit(): se ejecuta cuando se envía el formulario.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError(null);
  setLoading(true);
```

- controla que la página no se recargue al enviar el formulario.
- Limpia cualquier error anterior y se activa el estado de "cargando datos".

```
try {
  const response = await fetch(`${API_BASE_URL}/productos`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(productoParaEnviar)
  });

  if (!response.ok) {
    const errorData = await response.json().catch(() => ({}));
    throw new Error(errorData.message || 'Error al crear el producto');
  }

  navigate('/inventario', {
    state: { message: 'Producto creado exitosamente' }
  });

} catch (err) {
  setError(err.message || 'No se pudo crear el producto');
} finally {
  setLoading(false);
}
};
```

- Petición POST.
- Control de excepciones.
- Se redirecciona a '/inventario' por medio de navigate.

Posteriormente se implementa return() con la creación del formulario.

Implementando return():

```
EditarProd.jsx   CrearProducto.jsx ×
src > componentes > CrearProd > CrearProducto.jsx > CrearProducto
  8   export function CrearProducto() {
 93
 94   return (
 95     <>
 96       <Navbar />
 97
 98       <div className="crear-producto-container">
 99         <div className="form-card">
100           <h2>Crear Producto</h2>
101             {error && (
102               <div className="error-message">
103                 {error}
104                 <button onClick={() => setError(null)} className="error-close">×</button>
105               </div>
106             )}
107           {loadingCategorias ? (
108             <div className="loading">Cargando categorías...</div>
109           ) : (
110
111             <form onSubmit={handleSubmit}>
112               <div className="form-row">
113                 <div className="form-group">
114                   <label htmlFor="nombre">
115                     Nombre del producto <span className="required">*</span>
116                   </label>
117                   <input
```

- Botón close (x).
- onSubmit={handleSubmit}
- onChange = {handleChange}
- Cargar categorías: categorías.map
- Button type=”submit”
- {handleCancel}