

Introducción al desarrollo Web con Ant Design
Update

Para este ejemplo, vamos a volver cada una de las celdas de la tabla editables para poder actualizar los campos de cada producto desde la celda seleccionada dentro de la tabla, para separar las responsabilidades, crearemos el archivo **utils/editProduct.jsx**
En este archivo vamos a crear las funciones de **EditableCell**, **isEditing**, **edit**, **cancel** y **save**, todas estas la reutilice del componente Table de Antd con filas editables.

```
1 // src/utils/editProduct.js
2 import { message, Input, InputNumber, Form } from 'antd';
3
4 export const EditableCell = ({ editing, dataIndex, title, inputType, record, index, children, ...restProps }) => {
5   const inputNode = inputType === 'number' ? <InputNumber /> : <Input />;
6   return (
7     <td {...restProps}>
8       {editing ? (
9         <Form.Item
10           name={dataIndex}
11           style={{ margin: 0 }}
12           rules={[{ required: true, message: `Por favor ingrese ${title}!` }]}
13         >
14           {inputNode}
15         </Form.Item>
16       ) : (
17         children
18       )}
19     </td>
20   );
21 };
22
23 export const isEditing = (editingKey, record) => record.key === editingKey;
24
25 export const edit = (selectedRowKeys, setEditingKey, products, form) => {
26   if (selectedRowKeys.length === 1) {
27     const record = products.find(item => item.key === selectedRowKeys[0]);
28     form.setFieldsValue({ name: '', price: '', category: '', ...record });
29     setEditingKey(record.key);
30   } else {
31     message.error('Seleccione un producto para editar.');
```

Ahora en el archivo `pages/Products.js` vamos a importar todas las funciones del archivo anterior **`import { EditableCell, isEditing, edit, cancel, save } from '../utils/editProduct'`**; y los componentes de Antd necesarios **`import { Divider, Table, message, Form, Button, Popconfirm } from 'antd'`**;

Dentro del componente extraemos form de `useForm` **`const [form] = Form.useForm()`**; y creamos el estado de que nos servira para editar las celdas **`const [editingKey, setEditingKey] = useState('')`**;

Vamos a modificar el arreglo inicial `columns` por `baseColumns`

```
const baseColumns = [
  {
    title: 'Nombre',
    dataIndex: 'name',
    editable: true,
  },
  {
    title: 'Precio',
    dataIndex: 'price',
    editable: true,
  },
  {
    title: 'Categoría',
    dataIndex: 'category',
    editable: true,
  },
  {
    title: 'Imagen',
    dataIndex: 'imgURL',
    render: (url) => <img src={url} alt="product" style={{ width: 100 }} />,
  },
];
```

Agrega la función `operationColumn` que mostrará la columna para el campo editable aunque el botón que active esta opción será el que está en `CrudBotton`. Podemos observar que esta columna tiene los botones Guardar y Cancelar con sus correspondientes funcionalidades, esta función se trajo directamente de los componentes de Antd.

```
56
57
58 const operationColumn = {
59   title: 'Operación',
60   dataIndex: 'operation',
61   render: (_, record) => {
62     const editable = isEditing(editingKey, record);
63     return editable ? (
64       <span>
65         <Button
66           onClick={() => save(record.key, form, products, setProducts, setEditingKey, setSelectedRowKeys)}
67           style={{ marginRight: 8 }}
68         >
69           Guardar
70         </Button>
71         <Popconfirm title="¿Seguro que quieres cancelar?" onConfirm={() => cancel(setEditingKey)}>
72           <Button>Cancelar</Button>
73         </Popconfirm>
74       </span>
75     ) : null;
76   },
77   };
78
```

Seguido de la función anterior agrega las siguientes funciones que manejan los cambios dentro del arreglo `columns`.

```
const columns = editingKey ? [...baseColumns, operationColumn] : baseColumns;

const mergedColumns = columns.map((col) => {
  if (!col.editable) {
    return col;
  }
  return {
    ...col,
    onCell: (record) => ({
      record,
      inputType: col.dataIndex === 'price' ? 'number' : 'text',
      dataIndex: col.dataIndex,
      title: col.title,
      editing: isEditing(editingKey, record),
    }),
  };
});
```

Entonces lo que retorna tu componente Products sería lo que se muestra a continuación, en CrudButton, agrega **onEdit** y en Table los campos editables.

```
return (
  <div>
    <Nav />
    {user && (
      <CrudButton
        isSingleSelection={selectedRowKeys.length === 1}
        hasSelected={selectedRowKeys.length > 0}
        onDelete={handleDelete}
        onEdit={() => edit(selectedRowKeys, setEditingKey, products, form)}
      />
    )}
    <Divider />
    <div className="products-container">
      <h2>Productos</h2>
      <Form form={form} component={false}>
        <Table
          components={{
            body: {
              cell: EditableCell,
            },
          }}
          bordered
          dataSource={products}
          columns={mergedColumns}
          rowSelection={rowSelection}
          rowClassName="editable-row"
          pagination={{
            onChange: () => cancel(setEditingKey),
          }}
          scroll={{ y: 400 }}
        />
      </Form>
    </div>
  </div>
);
```

Finalmente en CrudButton pasa onEdit por props




```
const CrudButton = ({ isSingleSelection, onDelete, onEdit }) => {
  const [open, setOpen] = useState(false);
```

Y en el botón de editar en el evento onClick pasa esa función

```
<Button
  type="primary"
  icon={<EditOutlined />}
  disabled={!isSingleSelection}
  onClick={onEdit}
/>
```

Tus celdas editables quedarán de la siguiente manera. Verifica que la celda que selecciones sea editable y que los botones de guardar y cancelar sean funcionales, como puedes observar, los cambios en cada celda se muestran de manera local, si se recarga la página no persisten esos cambios.

Productos

Nombre	Precio	Categoría	Imagen	Operación
<input type="checkbox"/> Microsoft Surface Pro 9	82000	Tablet		
<input checked="" type="checkbox"/> Samsung Galaxy S23	28000	Smartphone		Guardar Cancelar
<input type="checkbox"/> MacBook Pro 16	75000	Laptop		

Es momento de integrar la solicitud a la Api para dejar de hacer los cambios en local e integrar estos cambios en la BD, recuerda que actualizar es una solicitud que requiere la existencia del token.

En el archivo **services/products.js** construye la solicitud para actualizar los productos.

```
36 export const updateProduct = async (productId, productData) => {
37   try {
38     const url = `${ENV.API_URL}/${ENV.ENDPOINTS.PRODUCTS}/${productId}`;
39     console.log(`URL para actualizar: ${url}`);
40     const token = storageController.getToken();
41     if (!token) {
42       throw new Error('No se encontró el token de autenticación');
43     }
44     const response = await axios.put(url, productData, {
45       headers: {
46         Authorization: `Bearer ${token}`,
47         'Content-Type': 'application/json'
48       }
49     });
50     return response.data;
51   } catch (error) {
52     console.error('Error al actualizar el producto', error);
53     throw error;
54   }
55 };
```

Importa y consume esta solicitud en el componente Productos y crea la función handleSave que usa actualiza el estado de productos y el estado de las celdas editables, al consumir la solicitud.

```
9   };
10  const handleSave = async (key) => {
11    try {
12      const row = await form.validateFields();
13      const newData = [...products];
14      const index = newData.findIndex((item) => key === item.key);
15      if (index > -1) {
16        const item = newData[index];
17        const updatedProduct = { ...item, ...row };
18
19        await updateProduct(item.key, updatedProduct);
20
21        newData.splice(index, 1, updatedProduct);
22        setProducts(newData);
23        setEditingKey('');
24        setSelectedRowKeys([]);
25        message.success('Producto actualizado exitosamente');
26      } else {
27        newData.push(row);
28        setProducts(newData);
29        setEditingKey('');
30      }
31    } catch (errInfo) {
32      console.error('Error al actualizar el producto:', errInfo);
33      message.error('Error al actualizar el producto');
34    }
35  };
36  };
```

Ahora solamente llama esta función en el botón Guardar de la celda editable.

```
const operationColumn = {
  title: 'Operación',
  dataIndex: 'operation',
  render: (_, record) => {
    const editable = isEditing(editingKey, record);
    return editable ? (
      <span>
        <Button
          onClick={() => handleSave(record.key)}
          style={{ marginRight: 8 }}
        >
          Guardar
        </Button>
        <Popconfirm title="¿Seguro que quieres cancelar?" onConfirm={() => cancel(setEditingKey)}>
          <Button>Cancelar</Button>
        </Popconfirm>
      </span>
    ) : null;
  },
};
```