Introducción al desarrollo Web con Ant Design

Modulo de usuarios

Lo primero será modificar nuestras rutas y los Tabs del Nav, quedará como Inicio, Productos, Usuarios, donde añadimos un condicional usando el contexto y validando que exista una sesión para hacer visible el Tab de **Usuarios**

```
const { user } = useAuth();
const tabNames = ["Inicio", "Productos"];
if (user) {
    tabNames.push("Usuarios");
}
```

En el archivo services/users.js crea el endpoint para traer todos los usuarios

```
const getAllUsers = async (token) => {
    try {
        const url = `${ENV.API_URL}/${ENV.ENDPOINTS.USERS}`;
        const response = await authFetch(url, {
            method: 'GET',
            headers: {
                'Authorization': `Bearer ${token}`
            }
        });

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        return await response.json();
    } catch (error) {
        console.error('Error al traer los usuarios', error);
        throw error;
    }
}
```

Crea el componente Users en pages/Users/index.jsx este tendrá una estructura similar a la vista de Productos, con sus respectivos cambios, estos son los imports que tengo en mi componente, repito, casi la misma estructura que Productos

src > pages > Users > index.jsx > Users

Haga clic para contraer el intervalo.

Buscar

```
1  import React, { useEffect, useState } from 'react';
2  import { Divider, Table, message, Form, Button, Popconfirm, Tag } from 'antd';
3  import Nav from '../../components/Nav';
4  import CrudButtonUser from '../../components/CrudButtonUser';
5  import { storageController } from '../../services/token';
6  import { usersService } from '../../services/users';
7  import { useAuth } from '../../hooks/useAuth';
8  import { EditableCell, isEditing, edit, cancel, save } from '../../utils/editProduct';
9  import './Users.css';
10
```

Los estados de selectedRowKeys y editingKey permanecen en este componente, al igual que las siguientes funciones que te puedes traer de Productos

```
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>
        ) : (
            <Popconfirm
                title="¿Seguro que quieres eliminar?"
                onConfirm={() => handleDelete(record.key)}
            >
                <Button type="danger">Eliminar</Button>
            </Popconfirm>
        );
    },
};

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

const mergedColumns = columns.map((col) => {
    if (!col.editable) {
        return col;
    }
    return {
        ...col,
        onCell: (record) => ({
            record,
            dataIndex: col.dataIndex,
            title: col.title,
            editing: isEditing(editingKey, record),
        }),
    };
});

const rowSelection = {
    selectedRowKeys,
    onChange: (selectedRowKeys) => {
        setSelectedRowKeys(selectedRowKeys);
    },
};
```

Las modificaciones vienen en ***baseColumns*** donde se añaden los campos según la solicitud, observa en en la columna Rol se usan las funciones **getColor** y **getRoleName** que crearemos más adelante.

```
const baseColumns = [
    {
        title: 'Nombre',
        dataIndex: 'username',
        editable: true,

    },
    {
        title: 'Email',
        dataIndex: 'email',
        editable: true,
    },
    {
        title: 'Rol',
        dataIndex: 'roles',
        with: '5%',
        editable: true,
        render: (roles, record) => (
            <span>
                {roles.map(role => (
                    <Tag color={getColor(role._id)} key={role._id}>
                        {getRoleName(role._id)}
                    </Tag>
                ))}
            </span>
        ),
    },
];
```

getColor y **getRoleName** determinará el color y el nombre en la etiqueta Tag según el rol del usuario, aquí dependerá del id de la BD en tu colección de roles, pero te doy un ejemplo.

```
// Función para obtener el nombre del rol basado en su ID
const getRoleName = (roleId) => {
    switch (roleId) {
        case '65e6453e7fec1bd764b997a2':
            return 'User';
        case '65e6453e7fec1bd764b997a3':
            return 'Moderador';
        case '65e6453e7fec1bd764b997a4':
            return 'Administrador';
        default:
            return 'Desconocido';
    }
};
// Función para modificar el color dependiendo del nombre nombre del rol basado en su ID

const getColor = (roleId) => {
    switch (roleId) {
        case '65e6453e7fec1bd764b997a2':
            return 'blue';
        case '65e6453e7fec1bd764b997a3':
            return 'yellow';
        case '65e6453e7fec1bd764b997a4':
            return 'green';
        default:
            return 'default';
    }
};
```

Construye la solicitud **fetchUsers** que consume la solicitud creada al principio

```
const fetchUsers = async () => {
    const token = storageController.getToken();
    try {
        const data = await usersService.getAllUsers(token);
        const usersWithKey = data.map(user => ({
            ...user,
            key: user._id,
            roles: user.roles.map(role => ({ _id: role, name: [role] }))

        }));
        setUsers(usersWithKey);

    } catch (error) {
        console.error('Error al obtener los usuarios', error);
    }
};

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

Por último lo que muestra el componente es lo siguiente, pueden reutilizar el componente **CrudButton** modificando y pasándole las propiedades, o crear **CrudButtonUser** con una estructura similar.

```
    return (
        <div>
            <Nav />
            <CrudButtonUser
                isSingleSelection={selectedRowKeys.length === 1}
                hasSelected={selectedRowKeys.length > 0}
                onDelete={handleDelete}
                onEdit={() => edit(selectedRowKeys, setEditingKey, users, form)}
                fetchUsers={fetchUsers} />



            <Divider />
            <div className="users-container">
                <h2>Usuarios</h2>
                <Form form={form} component={false}>
                    <Table
                        components={{
                            body: {
                                cell: EditableCell,
                            },
                        }}
                        bordered
                        dataSource={users}
                        columns={mergedColumns}
                        rowSelection={rowSelection}
                        rowClassName="editable-row"
                        pagination={{
                            onChange: () => cancel(setEditingKey),
                        }}
                        scroll={{ y: 400 }}
                    />
                </Form>
            </div>
        </div>
    );
};

export default Users;
```

Estas funciones por ahora solo muestran un mensaje en la consola.

```
    const handleDelete = async () => {
        console.log('Eliminar usuarios seleccionado');
    };
    const handleSave = async (key) => {
        console.log('Guardar usuario')
    };
```

Delete

Para eliminar un usuario solo tenemos que crear la solicitud en el archivo de services/users.js

```
const deleteUser = async (token, userId) => {
    try {
        if (!token) {
            throw new Error("Token no proporcionado");
        }
        const decoded = jwtDecode(token);
        const url = `${ENV.API_URL}/${ENV.ENDPOINTS.USERS}/${userId}`;
        console.log('url', url);
        const response = await authFetch(url, {
            method: 'DELETE',
            headers: {
                'Authorization': `Bearer ${token}`
            }
        });

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        return await response.json();
    } catch (error) {
        console.error('Error al eliminar el usuario', error);
        throw error;
    }
};
```

Y teniendo en cuenta que usamos tenemos una estructura en la tabla de productos y CrudButton parecida, la función handleDelete quedaría de la siguiente manera, considerando que para usar el endPoint deleteUser necesitamos ser administradores, por esa razón realizamos la condicional de que si obtenemos un error 403 de autorización nos mande el mensaje.

```
const handleDelete = async () => {
    if (selectedRowKeys.length !== 1) {
        message.error('Seleccione un solo usuario para eliminar');
        return;
    }
    try {
        const userId = selectedRowKeys[0];
        //console.log('userId a eliminar', userId);

        const token = storageController.getToken();
        if (!token) {
            throw new Error("Token no encontrado");
        }

        await usersService.deleteUser(token, userId);
        message.success('Usuario eliminado correctamente');
        setUsers(users.filter(user => user._id !== userId));
        setSelectedRowKeys([]);
        fetchUsers();
    } catch (error) {
        if (error.message === 'HTTP error! status: 403') {
            message.error('Necesitas ser administrador para realizar esta acción');
        } else {
            console.error('Error al eliminar el usuario', error);
            message.error('Error al eliminar el usuario');
        }

    }
};
```

Create

Para crear un nuevo usuario, reutiliza algunas de las funciones del CRUD de productos, como las funciones para el botón de agregar en el CrudButtonUser y crea el nuevo formulario con los campos requeridos para un nuevo usuario según tu Api.

Yo cree el componente ForNewUser que recibe el evento onSubmit del componente CrudButtonUser,

```
import React from 'react';
import { Form, Input, Button, Select } from 'antd';

const FormNewUser = ({ onSubmit }) => {

    const { Option } = Select;

    const onFinish = (values) => {
        onSubmit({
            ...values,
            roles: [values.roles]
        });
    };
```

Mi api recibe los campos de username, email, password y roles, por lo que estoy creando el contenido del formulario con esos campos, usando un select para los roles, recuerda que la propiedad name, se debe de especificar tal cual se reciben los datos en tu Api, así que cuida la nomenclatura en esta propiedad de acuerdo a la configuración de tus servicios.

```
    <Form
        onFinish={onFinish}
        labelCol={{ span: 8 }}
        wrapperCol={{ span: 14 }}
        layout="horizontal"
        initialValues={{}}
        style={{ maxWidth: 800, marginTop: 40 }}
    >
        <Form.Item
            name="username"
            label="Nombre"
            rules={[{ required: true, message: 'Por favor ingrese el nombre del usuario' }]}
        >
            <Input />
        </Form.Item>
        <Form.Item
            name="email"
            label="Correo eléctronico"
            rules={[{ required: true, message: 'Por favor ingrese el correo del usuario' }]}
        >
            <Input />
        </Form.Item>
        <Form.Item
            name="password"
            label="Contraseña"
            rules={[{ required: true, message: 'Por favor ingrese la categoría del usuario' }]}
        >
            <Input.Password />
        </Form.Item>
        <Form.Item
            name="roles"
            label="Rol"
            rules={[
                {
                    required: true,
                },
            ]}
        >
            <Select
                placeholder="Selecciona el rol del usuario"
                // onChange={onGenderChange}
                allowClear
            >
                <Option value="65e6453e7fec1bd764b997a2">Usuario general</Option>
                <Option value="65e6453e7fec1bd764b997a3">Moderador</Option>
                <Option value="65e6453e7fec1bd764b997a4">Administrador</Option>
            </Select>
        </Form.Item>
        <Form.Item
            wrapperCol={{ offset: 8, span: 14 }}
            style={{ display: 'flex', justifyContent: 'center', }}
        >
            <Button type="primary" htmlType="submit" style={{ width: 160, fontSize: 16, height: 35 }}>
                Guardar
            </Button>
        </Form.Item>
    </Form>
  );
};
```

Una vez creado el formulario, regresa al componente CrudButtonUser y crea la función handleSubmit para enviar los datos desde el modal.

```
};
const handleSubmit = async (formData) => {
    console.log('Formulario', formData);
    const userData = {
        username: formData.username,
        email: formData.email,
        password: formData.password,
        roles: formData.roles
    };

    try {
        setConfirmLoading(true);
        const token = localStorage.getItem('token');
        await usersService.createUser(token, userData);
        setConfirmLoading(false);
        setOpenModal(false);
        message.success('Usuario creado correctamente');
        fetchUsers();
    } catch (error) {
        console.error('Error al crear el usuario', error);
        setConfirmLoading(false);
        message.error('Error al crear el usuario');
    }
};
```

No te olvides llamar tu nuevo formulario en el Modal

```
<Modal
    title="Crear usuario"
    open={openModal}
    confirmLoading={confirmLoading}
    onCancel={handleCancelModal}
    width={800}
    okText='Guardar'
    footer={null}

>
    <FormNewUser onSubmit={handleSubmit} />
</Modal>
```

Trabajo extra: Crear un usuario también debe de ser tarea exclusiva del administrador, realiza las validaciones tanto en tu Api (en caso de ser necesario) y en el front.

Update

Para el campo de editar vamos a utilizar exactamente el mismo proceso que en el recurso 11 update (products), crearemos el archivo **utils/editUser.jsx** en este archivo vamos a crear las funciones de EditableCell, isEditing, edit, cancel y save

```jsx
import { message, Input, InputNumber, Form } from 'antd';

export const EditableCell = ({ editing, dataIndex, title, inputType, record, index, children, ...restProps }) => {
    const inputNode = inputType === 'number' ? <InputNumber /> : <Input />;
    return (
        <td {...restProps}>
            {editing ? (
                <Form.Item
                    name={dataIndex}
                    style={{ margin: 0 }}
                    rules={[{ required: true, message: `Por favor ingrese ${title}!` }]}
                >
                    {inputNode}
                </Form.Item>
            ) : (
                children
            )}
        </td>
    );
};

export const isEditing = (editingKey, record) => record.key === editingKey;

export const edit = (selectedRowKeys, setEditingKey, users, form) => {
    if (selectedRowKeys.length === 1) {
        const record = users.find(item => item.key === selectedRowKeys[0]);
        form.setFieldsValue({ username: '', email: '', roles: '', ...record });
        setEditingKey(record.key);
    } else {
        message.error('Seleccione un usuario para editar.');
    }
};

export const cancel = (setEditingKey) => {
    setEditingKey('');
};

export const save = async (key, form, users, setusers, setEditingKey, setSelectedRowKeys) => {
    try {
        const row = await form.validateFields();
        const newData = [...users];
        const index = newData.findIndex((item) => key === item.key);
        if (index > -1) {
            const item = newData[index];
            newData.splice(index, 1, { ...item, ...row });
            setusers(newData);
            setEditingKey('');
        } else {
            newData.push(row);
            setusers(newData);
            setEditingKey('');
        }
        setSelectedRowKeys([]);
    } catch (errInfo) {
        console.log('Error en la validación:', errInfo);
    }
};

```

Ahora en el index de pages/User vamos a importar todas las funciones del archivo anterior import { EditableCell, isEditing, edit, cancel, save } from '../../utils/editUser'; 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 servirá para editar las celdas const [editingKey, setEditingKey] = useState('');

baseColumns por ahora en la fila de Rol, marcamos editable como false, para solo poder editar Nombre y Email.

```
const baseColumns = [
    {
        title: 'Nombre',
        dataIndex: 'username',
        editable: true,
    },
    {
        title: 'Email',
        dataIndex: 'email',
        editable: true,
    },
    {
        title: 'Rol',
        dataIndex: 'roles',
        with: '5%',
        editable: false,
        render: (roles, record) => (
            <span>
                {roles.map(role => (
                    <Tag color={getColor(role._id)} key={role._id}>
                        {getRoleName(role._id)}
                    </Tag>
                ))}
            </span>
        ),
    },
];
```

Agrega la función **operationColumn, columns y mergedColumns**

No te olvides de llamar todas estas funciones en el componente Table y CrudButtonUser (sigue los pasos del recurso 11)

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

            </div>
        </div>
    );
};

export default Users;
```

Entonces solo nos quedan dos cosas crear la solicitud a la Api (será diferente porque involucra token) y **handleSave** que será la encargada de realizar esta solicitud dentro del botón Guardar.

En el archivo **src/services/users.js** importa storageController y axios y exporta la siguiente función en **usersService**

```
const updateUser2 = async (userId, userData) => {
    try {
        const url = `${ENV.API_URL}/${ENV.ENDPOINTS.USERS}/${userId}`;
        // console.log(`URL para actualizar: ${url}`);
        const token = storageController.getToken();
        if (!token) {
            throw new Error('No se encontró el token de autenticación');
        }
        const response = await axios.put(url, userData, {
            headers: {
                Authorization: `Bearer ${token}`,
                'Content-Type': 'application/json'
            }
        });
        return response.data;
    } catch (error) {
        console.error('Error al actualizar el usuario', error);
        throw error;
    }
};
```

Para finalizar construye **handleSave** no te olvides de importar los elementos necesarios.

```
const handleSave = async (key) => {
    try {
        const row = await form.validateFields();
        const newData = [...users];
        const index = newData.findIndex((item) => key === item.key);
        if (index > -1) {
            const item = newData[index];
            const updatedUser = { ...item, ...row };

            await usersService.updateUser2(item.key, updatedUser);

            newData.splice(index, 1, updatedUser);
            setUsers(newData);
            setEditingKey('');
            setSelectedRowKeys([]);
            message.success('Usuario actualizado exitosamente');
        } else {
            newData.push(row);
            setUsers(newData);
            setEditingKey('');
        }
    } catch (errInfo) {
        console.error('Error al actualizar el usuario:', errInfo);
        message.error('Error al actualizar el usuario');
    }
};
```