

Introducción al desarrollo Web con Ant Design User Update


Para mostrar y editar la información del usuario, estoy utilizando un componente List de Antd, esta lista muestra título, descripción, icono y un botón para editar ese campo, el componente completo los puedes traer de Antd y solo modificar la data de acuerdo a tus necesidades, por ejemplo, yo estoy usando useAuth para mostrar los datos del usuario logueado.

Así quedaría tu componente List

```
1  ~/Desktop/Mayo-Agosto24/DW/intro-antd/src/components/CrudButton/index.jsx
2  import { Avatar, List } from 'antd';
3  import { useAuth } from '../../hooks/useAuth';
4  import { UserOutlined, MailOutlined } from '@ant-design/icons';
5  const ListDrawer = () => {
6    const { user, } = useAuth();
7
8    const data = [
9      {
10       title: 'Nombre',
11       description: user.username,
12       icon: <UserOutlined />,
13     },
14     {
15       title: 'Email',
16       description: user.email,
17       icon: <MailOutlined />,
18     },
19   ];
20
21   return (
22     <List
23       itemLayout="horizontal"
24       dataSource={data}
25       renderItem={(item, index) => (
26         <List.Item>
27           <List.Item.Meta
28             avatar=<Avatar icon={item.icon} />
29             title=<span style={{ fontSize: '18px' }}>{item.title}</span>
30             description={item.description}
31           />
32           <div>
33             <button onClick={() => alert(`Editar ${item.title}`)}>Editar</button>
34           </div>
35         </List.Item>
36       )}
37     />
38   );
39
40 };
41
42 export default ListDrawer;
43
```


Con una vista como la siguiente

× **Mi perfil**

 **Nombre**

Ana

Editar

 **Email**

ana@arteaga.com

Editar

Ahora el cambio que vamos a aplicar al componente List, es hacer esas celdas editables una vez que se accione el botón Editar, para poder editar sobre el mismo campo, añadiendo un botón para Guardar o cancelar los cambios.

Lo primero será agregar los estados para gestionar las celdas editables, agrega editingField y formData que recibe como estado inicial el objeto con los datos del usuario.

```
const { user } = useAuth();
const [editingField, setEditingField] = useState(null);
const [formData, setFormData] = useState({
  username: user.username,
  email: user.email,
});
```

Añade las cuatro funciones para controlar el estado, es decir el proceso de actualización, la primera, activa las celdas editables, la segunda escucha los cambios, la tercera guarda o guardará los nuevos cambios, y la cuarta cancela el proceso y retorna los mismos valores.

```
const handleEditClick = (field) => {
  setEditingField(field);
};

const handleInputChange = (e) => {
  const { name, value } = e.target;
  setFormData({
    ...formData,
    [name]: value,
  });
};

const handleSaveClick = () => {
  setEditingField(null);
};

const handleCancelClick = () => {
  setEditingField(null);
  setFormData({
    username: user.username,
    email: user.email,
  });
};
```


Ahora en tu constante data agrega el campo field con el identificador y en descripción pasale el valor de formData

```
const data = [
  {
    title: 'Nombre',
    field: 'username',
    description: formData.username,
    icon: <UserOutlined />,
  },
  {
    title: 'Email',
    field: 'email',
    description: formData.email,
    icon: <MailOutlined />,
  },
];
```

Ahora tu componente List en la propiedad **renderItem** solo recibe Item y el campo descripción lo modificamos por completo para añadirle la lógica para las celdas editables y los botones correspondientes, añadimos un div para que estos se muestren en la parte de abajo de cada elemento. Observa que se añade el componente Input y Button por lo que no te olvides de importarlos, y cada elemento utiliza las funciones creadas anteriormente.

```
<List
  itemLayout="horizontal"
  dataSource={data}
  renderItem={(item) => (
    <List.Item>
      <List.Item.Meta
        avatar={<Avatar icon={item.icon} />}
        title={<span style={{ fontSize: '18px' }}>{item.title}</span>}
        description={
          editingField === item.field ? (
            <div>
              <Input
                name={item.field}
                value={formData[item.field]}
                onChange={handleInputChange}
                style={{ marginBottom: 8 }}
              />
              <div>
                <Button onClick={handleSaveClick} type="primary" style={{ marginRight: 8 }}>
                  Guardar
                </Button>
                <Button onClick={handleCancelClick}>Cancelar</Button>
              </div>
            </div>
          ) : (
            item.description
          )
        }
      />
      {editingField !== item.field && (
        <Button onClick={() => handleEditClick(item.field)}>Editar</Button>
      )}
    </List.Item>
  )}
/>
```

Al presionar Editar, se mostrará algo así en tu vista.



Nombre

Guardar Cancelar

Finalmente para realizar la solicitud a la Api crea la solicitud en services/users.js

```
const updateUser = async (token, updateData) => {
  try {
    const decoded = jwtDecode(token);
    const userId = decoded.id;
    const url = `${ENV.API_URL}/${ENV.ENDPOINTS.USERS}/${userId}`;

    const response = await authFetch(url, {
      method: 'PUT',
      body: JSON.stringify(updateData),
    });

    return await response.json();
  } catch (error) {
    console.error('Error en updateUser', error);
    throw error;
  }
}
```

Importa esta función y getToken de storageController en el componente ListDrawer

```
import { useAuth } from '../hooks/useAuth';
import { usersService } from '../services/users';
import { storageController } from '../services/token';
```

Y extrae **updateUser** de useAuth

```
const { user, updateUser } = useAuth();
```

La función que realiza esta solicitud es **handleSaveClick** y queda de la siguiente manera, primero verifica la existencia del token al ser una solicitud protegida, realiza la solicitud a la Api por medio de updateUser enviando el token y formData, y actualiza updateUser del contexto.

No olvides mandar al usuario un mensaje de confirmación, y buen manejo del error.

```
const handleSaveClick = async () => {
  setEditingField(null);

  try {
    const token = await storageController.getToken();
    if (!token) throw new Error('Token no disponible');

    const updatedUser = await usersService.updateUser(token, formData);
    updateUser('username', updatedUser.username);
    updateUser('email', updatedUser.email);
    message.success('Usuario actualizado correctamente');
  } catch (error) {
    message.error('Error al actualizar usuario');
    console.error('Error al actualizar usuario:', error);
  }
};
```

NOTA: Realice una pequeña modificación en AuthFetch para mejor manejo del error y agregando Content-type

```
c > utils > JS authFetch.js > ...
1  import { storageController } from "../services/token";
2  import { tokenExpired } from './tokenExpired';
3
4  export const authFetch = async (url, params) => {
5      const token = await storageController.getToken();
6
7      const logout = () => {
8          storageController.removeToken();
9      };
10
11     if (!token) {
12         logout();
13         throw new Error('No token available');
14     }
15
16     if (tokenExpired(token)) {
17         logout();
18         throw new Error('Token expired');
19     }
20
21     const options = {
22         ...params,
23         headers: {
24             ...params?.headers,
25             Authorization: `Bearer ${token}`,
26             'Content-Type': 'application/json',
27         }
28     };
29
30     try {
31         const response = await fetch(url, options);
32         if (!response.ok) {
33             throw new Error(`HTTP error! status: ${response.status}`);
34         }
35         return response;
36     } catch (error) {
37         console.log('Error en authFetch', error);
38         throw error;
39     }
40
41
```