ProjectManager | MERN

Context para proyectos

1. Crear el componente `<ProjectsProvider/>` en la carpeta `/context` con el siguiente código inicial

```
import React, { createContext, useState } from "react";

const ProjectsContext = createContext();

export const ProjectsProvider = ({ children }) => {
  const [projects, setProjects] = useState([]);
  return (
    <ProjectsContext.Provider
      value={{
        projects,
      }}
    >
      {children}
    </ProjectsContext.Provider>
  );
};

export default ProjectsContext;
```

2. Implementar el componente `<ProjectsProvider/>` en el componente `<App/>` de la siguiente manera:

```
<BrowserRouter>
    <AuthProvider>
      <ProjectsProvider>
      <Routes>
        {/* rutas públicas */}
        {/* rutas privadas */}
      </Routes>
     </ProjectsProvider>
    </AuthProvider>
</BrowserRouter>
```

3. Crear el hook `useProjects()` que consumirá los datos del `ProjectsContext`

```
import {useContext} from 'react';
import ProjectsContext from '../context/ProjectsProvider';
```

```
const useProjects = () => {
    return useContext(ProjectsContext)
}

export default useProjects
```

4. Crear el *state* `alert` y la función `showAlert()` en `<ProjectsProvider>` por medio de la cual setearemos dicho estado en caso de error. Se debe pasar el *state* en el `value` de `<ProjectsContext.Provider/>`

```
const [alert, setAlert] = useState({})

const showAlert = (msg, time = true) => {
  setAlert({
    msg,
  });

  if (time) {
    setTimeout(() => {
      setAlert({});
    }, 3000);
  }

};
```

MOSTRAR los proyectos o un proyecto en particular

1. Crear la función `getProjects()` en `<ProjectsProvider>` para traer todos los proyectos, y almancerlos en el *state* `projects`. Dicha función debe pasarse en el `value` de `<ProjectsContext.Provider/>`

```
const getProjects = async () => {

  setLoading(true);

  try {
    const token = sessionStorage.getItem("token");

    if (!token) return null;

    const { data } = await clientAxios.get("/projects", {
      headers: {
        "Content-Type": "application/json",
        Authorization: token,
      },
    });
```

```
      setProjects(data.projects);
      setAlert({})

    } catch (error) {
      console.error(error);
      showAlert(error.response ? error.response.data.msg : 'Upps.. hubo un error',
false)
    } finally {
      setLoading(false)
    }
};
```

2. Crear el state `loading` y setearlo en `true` como estado inicial. Este servirá para indicar cuando se está haciendo una petición a la API.

```
const [loading, setLoading] = useState(true);
```

3. Utilizando el hook `useProjects()` traer los siguintes *states* y *funciones* en el componente `<Projects/>`

```
const {loading, alert, projects, getProjects} = useProjects();
```

4. Utilizar un `useEffect()` en el componente `<Projects/>` para traer todos los proyectos.

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

5. Modificar la estructura del componente `<Projects/>` para que en el caso que haya una alerta activa la muestre, de contrario muestre los proyectos cuando se haya cargado los datos en el *state* `projects`

```
if(alert.msg){
    return <Alert {...alert}/>
  }

return (
  <>
  <h1
    className='text-4xl font-black'
  >
    Proyectos
  </h1>
```

```
    <div
      className='bg-white p-5 shadow mt-10 rounded-md'
    >
      {
        loading ?
        <p>Cargando...</p> :(
          projects.length
          ?
            projects.map(project => <ProjectPreview key={project._id}
{...project}/>)
          :
          <p>No hay proyectos agregados</p>
        )
      }
    </div>
    </>
  )
```

6. Modificar la estructura del componente `<ProjectPreview/>` para que muestre la información que recibe por *props*

```
<div className='border-b p-3 flex justify-between'>
    <p>
        {name}
        <span
        className='text-sm text-gray-500 uppercase'
        >
        {" | " + client }
        </span>
    </p>
    <Link
        to={`/projects/${_id}`}
        className="uppercase text-sm text-gray-400 hover:text-gray-800 font-bold"
    >
        Ver proyecto
    </Link>
</div>
```

7. Crear el estado `project` para guardar los datos del proyecto que esté en memoria.

```
  const [project, setProject] = useState({});
```

8. Crear la función `getProject()` en `<ProjectsProvider>` para traer el proyecto que machee con el ID que recibe por parámetro, y almancerlo en el *state* `project`. Dicha función debe pasarse en el `value` de `<ProjectsContext.Provider/>`

```
const getProject = async (id) => {
  setLoading(true);

  try {
    const token = sessionStorage.getItem("token");

    if (!token) return null;

    const config = {
      headers: {
        "Content-Type": "application/json",
        Authorization: token,
      },
    };

    const { data } = await clientAxios.get(`/projects/${id}`, config);

    setProject(data.project);
    setAlert({});
  } catch (error) {
    console.error(error);
    showAlert(
      error.response ? error.response.data.msg : "Upps.. hubo un error",
      false
    );
  } finally {
    setLoading(false);
  }
};
```

9. Implementar en el componente `<Project/>` el hook `useParams()` de *react-router-dom* para leer el ID de la URL.

```
  const {id} = useParams();
```

10. Utilizar un `useEffect()` en el componente `<Project/>` para traer los datos del proyecto.

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

11. Meditante el hook `useProjects()` traer el *state* `project` y demás datos del Context de proyectos.

```

```

```
const { loading, alert, getProject, project } = useProjects();
const { name, description, dateExpire, client } = project;
```

12. Utilizar el *state* `loading` para esperar que la API responda antes de mostrar los datos del proyecto

```
 {loading ? (
        <p>Cargando...</p>
      ) : (
        // estrutura del componente
      )}
```

13. Interpolar los datos del proyecto con la estructura del componente

```
<>
  <div className="flex justify-between">
    <h1 className="text-4xl uppercase font-bold">{name}</h1>
    <Link
      to={`/projects/edit-project/1`}
      className="flex justify-center items-center gap-2 text-gray-500 hover:text-
black uppercase font-bold"
    >
      <svg
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        strokeWidth={1.5}
        stroke="currentColor"
        className="w-6 h-6"
      >
        <path
          strokeLinecap="round"
          strokeLinejoin="round"
          d="M16.862 4.487l1.687-1.688a1.875 1.875 0 112.652 2.652L6.832 19.82a4.5
4.5 0 01-1.897 1.13l-2.685.8.8-2.685a4.5 4.5 0 011.13-1.897L16.863 4.487zm0 0L19.5
7.125"
        />
      </svg>
      <p>Editar</p>
    </Link>
  </div>
  <h2 className="text-2xl uppercase font-bold text-gray-600">{client}</h2>
  <hr className="border-b border-gray-600"/>
  <p>{description}</p>
  <div className="flex justify-between">
    <p className="font-bold text-3xl mt-10 mb-5">Tareas del proyecto</p>
    <div
      className="flex justify-center items-center gap-1 text-gray-500 hover:text-
```

```
black cursor-pointer"
      /*  onClick={handleModalForm} */
    >
      <svg
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        strokeWidth={1.5}
        stroke="currentColor"
        className="w-6 h-6"
      >
        <path
          strokeLinecap="round"
          strokeLinejoin="round"
          d="M12 4.5v15m7.5-7.5h-15"
        />
      </svg>
      <p>Nueva Tarea</p>
    </div>
  </div>
  {[1, 2].map((task) => (
    <Task />
  ))}
  <div className="flex items-center justify-between">
    <p className="font-bold text-3xl mt-10 mb-5">Colaboradores</p>

    <button
      className="flex justify-center items-center gap-1 text-gray-500 hover:text-
black cursor-pointer"
      /* onClick={handleModalAddCollaborator} */
    >
      <svg
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        strokeWidth={1.5}
        stroke="currentColor"
        className="w-6 h-6"
      >
        <path
          strokeLinecap="round"
          strokeLinejoin="round"
          d="M19 7.5v3m0 0v3m0-3h3m-3 0h-3m-2.25-4.125a3.375 3.375 0 11-6.75 0
3.375 3.375 0 016.75 0zM4 19.235v-.11a6.375 6.375 0 0112.75 0v.109A12.318 12.318 0
0110.374 21c-2.331 0-4.512-.645-6.374-1.766z"
        />
      </svg>

      <p>Agregar Colaborador</p>
    </button>
  </div>
  {[1, 2].map((collaborator) => (
    <Collaborator />
```

```
  ))}
</>
```

CREAR un nuevo proyecto

1. Instalar e utilizar *SweetAlert 2* para mostrar mensajes dinámicos

```
import Swal from "sweetalert2";

const Toast = Swal.mixin({
  toast: true,
  position: "top-end",
  showConfirmButton: false,
  timer: 3000,
  timerProgressBar: true,
  didOpen: (toast) => {
    toast.addEventListener("mouseenter", Swal.stopTimer);
    toast.addEventListener("mouseleave", Swal.resumeTimer);
  },
});
```

2. Importar e implementar el hook `useNavigate()` de *react router dom* para hacer redirecciones

```
  const navigate = useNavigate();
```

3. Crear la función `storeProject()` en `<ProjectsProvider>` para crear un nuevo proyecto y actualizar el *state* `projects`. Dicha función debe pasarse en el `value` de `<ProjectsContext.Provider/>`

```
const storeProject = async (project) => {

    const token = sessionStorage.getItem("token");

    if (!token) return null;

    try {
      const config = {
        headers: {
          "Content-Type": "application/json",
          Authorization: token,
        },
      };

      const { data } = await clientAxios.post("/projects", project, config);
```

```
        setProjects([...projects, data.project]);

        Toast.fire({
          icon: "success",
          title: data.msg,
        });

        navigate('projects')

      }catch{
        console.error(error);

        const { response } = error;
        if (response?.status === 401) {
          navigate("/");
        }else{
          showAlert(
            response ? response.data.msg : "Upps.. hubo un error",
            false
          );
        }
      }
  }
```

4. Implementar `useForm()` en el componente `<FormProject/>`

```
  const {formValues, handleInputChange, reset} = useForm({
    name : "",
    description : "",
    dateExpire : "",
    client : ""
  });

  let {name, description, dateExpire, client} = formValues;
```

5. Agregar los atributos que corresponden a cada elemento del formulario. Ejemplo:

```
value={name}
name="name"
onChange={handleInputChange}
```

6. Crear la función `handleSubmit()` e implementarla en el evento `onSubmit()` del formulario

```
  const handleSubmit = (e) => {
    e.preventDefault();
  };
```

7. Traer el *state* de alerta con el *hook* `useProject()`

```
const {showAlert, alert} = useProjects()
```

8. Implementar las validaciones correspondientes dentro de la función `handleSubmit()`

```
if ([name, description, dateExpire, client].includes("")) {
    showAlert("Todos los campos son obligatorios",true);
    return null
}
```

9. Una vez validados los campos, utilizar la función `storeProject()` para enviar la información del formulario al `<ProjectsProvider/>` y una vez guardado el proyecto resetear el formulario.

```
storeProject({
  name,
  description,
  dateExpire,
  client
});

reset()
```