ProjectManager | MERN

EDITAR un proyecto existente

1. Implementar el hook `useParams()` para leer el id que viene por parámetro en el momento de cargar el formulario de edición de proyecto en el componente `<FormProject/>`

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

2. Con el hook `useRef()` hacer referencia a los inputs del formulario para cargar los datos del proyecto

```
const inputName = useRef(null)
const inputDescription = useRef(null)
const inputDateExpire = useRef(null)
const inputClient = useRef(null)
```

3. Vincular cada campo del formulario mediante el atributo `ref`. Ejemplo:

```
ref={inputName}
```

4. Utilizando el hook `useEffect()` cargar los datos en el formulario siempre y cuando exista un id en la URL

```
useEffect(() => {
  if (id) {
    const { name, description, dateExpire, client } = project;

    inputName.current.value = name;
    inputDescription.current.value = description;
    inputDateExpire.current.value = dateExpire.split('T')[0];
    inputClient.current.value = client;

    name = project.name;
    description = project.description;
    dateExpire = project.dateExpire;
    client = project.client;
  }
}, [id]);
```

5. Agregar el atributo id en el objeto que se envía como parametro de la función `storeProject()`

```
storeProject({
  id: id ? id : null,
  name,
  description,
  dateExpire,
  client,
});
```

6. Editar la función `storeProject()` en el componente `<ProjectsProvider/>` para que si en el objeto viene un id, haga la petición de actualización, de lo contrario de creación de un nuevo proyecto.

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

    const projectsUpdated = projects.map((projectState) => {
      if (projectState._id === data.project._id) {
        return data.project;
      }

      return projectState;
    });

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

  } else {
    const { data } = await clientAxios.post("/projects", project, config);
    setProjects([...projects, data.project]);
    Toast.fire({
      icon: "success",
      title: data.msg,
    });
  }
```

ELIMINAR un proyecto

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

```
const deleteProject = async (id) => {
    try {
      const token = sessionStorage.getItem("token");

      if (!token) return null;

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

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

      const projectsFiltered = projects.filter((project) => project._id !== id);

      setProjects(projectsFiltered);

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

      navigate("projects");

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

2. Utilizar el hook `useProjects()` para traer del *state* del Context de Proyectos `deleteProject` y `project`

```
  const {deleteProject,project} = useProjects();
```

3. Crear la función `handleDelete()` que ejecuta una alerta de confirmación con *SweetAlert2* y ejecutarla en el evento *onClick* del botón eliminar

```
  const handleDelete = () => {
    Swal.fire({
```

```
      title: '¿Estás seguro de eliminar el proyecto?',
      showCancelButton: true,
      confirmButtonColor : 'red',
      confirmButtonText: 'Confirmar',
    }).then((result) => {
      if (result.isConfirmed) {
        deleteProject(project._id)
      }
    })
  }
```