Universidad Interamericana de Puerto Rico
Recinto de Arecibo

Project Organizer

Un Sistema de Manejo de Proyectos

COMP3400: Software Engineering

Abimael Santa
Abielmelex Nieves
Ramón Valentín
Génesis M. Ojeda

Documentación del Código

Front-end:

App.tsx

- Base de la aplicación donde se muestra el Navbar como componente, se establece las rutas con react-router-dom y se llama cada elemento en las mismas, siendo los Containers correspondiente(Projects, Members, Tasks). A su este archivo realiza el fetch inicial de la api con la ruta Get_All_Projects

```
import 'bootstrap/dist/css/bootstrap.min.css';

import 'bootstrap/dist/js/bootstrap.bundle.min.js';

import '@fortawesome/fontawesome-free/css/all.min.css';

// Estas importaciones son referentes a Bootstrap y a una libreria de
iconos




import React, { useEffect, useState } from 'react';

// Se importan los Hooks necesarios para la funcionalidad de nuestra
aplicacion

// Con react estamos importando los hooks useEffect y useState

// 1- useState nos permite manejar los estados de las variables para ser
reasignados de forma dinamica



// 2- useEffect nos permite realizar varias acciones una vez se haya
montado el componente, en este archivo

// nos esto nos ayuda a llamar a la api en el momento indicado sin
obstruir el orden de renderizado de la pagina.
```

```
// *Esto nos permite establecer 'placeholders' de carga*


import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

// Para la navegacion de la pagina se implemento react-router-dom que nos facilita el manejo de las rutas,en una aplicacion SPA (Single Page Aplication)

import { Project } from './interfaces/Project';

// Interface del data object -Product



import ProjectsContainer from './components/Containers/ProjectContainer';

import MembersContainer from './components/Containers/MembersContainer';

import TasksContainers from './components/Containers/TasksContainers';

// Cada ruta posee un contenedor que manejara o los productos, members o tasks



import Navbar from './components/Navbar/Navbar';

import PlaceholderCard from './components/Cards/PlaceHolder';

import ErrorModal from './components/Modal/ErrorModal';

// Se importan componentes utilizados en esta pagina







function App() {

  const [projects, setProjects] = useState<Project[]>([]);
```

```
  const [selectedProject, setSelectedProject] = useState<Project |
null>(null);

  // Se inicializaron las variables, producst y selectedProduct para
definir los projectos

  // y manjear el proyecto seleccionado que unicamente se mostrara entre
las rutas de task/member/projects



  const [loading, setLoading] = useState<boolean>(true);

  // Se maneja un estado que indica que la data esta cargando



  const [isErrorModalOpen, setIsErrorModalOpen] = useState(false);

  const [errorMessage, setErrorMessage] = useState<string>('');

  const handleCloseErrorModal = () => setIsErrorModalOpen(false);

  // Se definieron varios estados y una funcion para manejar la logica

  // de aparecer un modal cuando ocurra un error al llamar a la api



  useEffect(() => {

    fetchProjects();

  }, []);

  // Este fecth llama la api de get_all_products, mientras se procesa la
solicitud carga una pantalla de cargar, de ocurrir un error lo

  // muestra en pantalla, de lo contrario asigna la data a la constante
products

  const fetchProjects = () => {

    setLoading(true);
```

```
    fetch('http://172.16.5.78:5000/api/get_all_projects')
      .then(response => {
        if (!response.ok) {
          setErrorMessage('Network response was not ok');
          setIsErrorModalOpen(true);
        }
        return response.json();
      }).then(data => {
        if (Array.isArray(data.projects)) {
          setProjects(data.projects);
          console.log('Projects Fetch: ', data.projects);
        } else {
          setErrorMessage('An error ocurred while trying to connect to the
Data Base');
          setIsErrorModalOpen(true);
        }
      })
      .catch(error => {
        console.error('There was a problem with the fetch operation:',
error);
        setErrorMessage('There was a problem connecting to the server');
        setIsErrorModalOpen(true);
        setProjects([]);
      }).finally(() => setLoading(false));
```

```
  };



  // Este return renderiza los compponentes base para la navegacion, estableciendo las posibles rutas y llamando los contenedores correspondientes

  // A su vez carga el componente Navbar que maneja el enrutado de la pagina como el filtrado individual de proyectos

  // Ademaas este return muestra placeholder de carga y un modal de ocurrir un error

  return (

    <Router>

      <Navbar projects={projects} setSelectedProject={setSelectedProject} fetchProjects={fetchProjects}></Navbar>

        {loading ? (

          <div>

            <PlaceholderCard />

            <PlaceholderCard />

            <PlaceholderCard />

            <PlaceholderCard />

            <PlaceholderCard />

          </div>

        ) : (

          <Routes>
```

```
            <Route path="/" element={<ProjectsContainer
selectedProject={selectedProject} projects={projects}
fetchProjects={fetchProjects}/>} />

            <Route path="/tasks" element={<TasksContainers
selectedProject={selectedProject}/>} />

            <Route path="/members" element={<MembersContainer
selectedProject={selectedProject} />} />

          </Routes>

        )}

        <ErrorModal errorMessage={errorMessage} isOpen={isErrorModalOpen}
onClose={handleCloseErrorModal} />

    </Router>

  );

}



export default App;
```

Navbar.tsx

Tiene como funcionalidad presentar las opciones de navegación, además el botón de Add Project que hace visible el formulario de Add Project. A su vez muestra una opción de filtrar los productos y mostrar un producto seleccionado a través de todas las paginas.

```
import React, { useState } from "react";
import { Link } from "react-router-dom";
// Se importan los hooks necesarios, Link es un componente similar a la
etiqueta ancord pero optimizado
// Al utilizar Link y navegamos mediante las rutas de react-router-dom nos
permite renderizar  solo
```

```
// los elementos de cada ruta. (No el Navbar! ) Manteniendo el estado del
proyecto seleccionado.

import "../Cards/css/style.css";
// Estilos Para el componente Navbar

import { Project} from "../../interfaces/Project"
// Data Object of Product

import Modal from "../Modal/modal";
import ProjectForm from "../../forms/projects/ProjectForm";
// Componente Modal que se le pasa un Formulario como hijo

interface NavbarPropos {
    projects: Project[];
    setSelectedProject: (project: Project | null) => void;
    fetchProjects: () => void;
}
// Se le pasan los props de Project y SelectedProject ya que desde el
navbar el usuario va a poder filtrar por un proyecto individual

const Navbar: React.FC<NavbarPropos> = ({ projects, setSelectedProject,
fetchProjects }) => {
    const [isModalOpen, setModalOpen] = useState(false);
    const openModal = () => setModalOpen(true);
    const closeModal = () => setModalOpen(false);
    // Estado y logica para manejar la visibilidad del modal(Crear
Producto)


    // Se define una funcion que espera al ChangeEvent del Selection Task
    const handleProjectSelect = (event:
React.ChangeEvent<HTMLSelectElement>) => {
        const selectedId = event.target.value;
        //En el selection tag se muestra las opciones de All y el project
name pero el valor esta atado al project_id de ese project Name
        // Se le asigna el valor del selection a esta variable para
manejar la logica

        if (selectedId === "all") {
```

```
            setSelectedProject(null);
        } else {
            const selectedProject = projects.find(project =>
project.project_id?.toString() === selectedId);
            if (selectedProject) {
                setSelectedProject(selectedProject);
            }
        }

        //Si el valor es All, no reasigna el estado de SelectedProject en
el archivo App.tsx
        //De seleccionar un projecto y este coincida con un project_id
existente en la lista de projectos obtenida de la
        // Base de datos, asigna el estado de la variable selectedProject
al projecto Seleccionado.
    };


    return (
        <>
            <nav className="navbar navbar-expand-lg bg-body-tertiary
custom-navbar" >
                <div className="container-fluid">
                    <Link className="navbar-brand text-color
navbar-brand-bold" to="/">Project Management</Link>
                    <button className="navbar-toggler" type="button"
data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
                        <span className="navbar-toggler-icon"></span>
                    </button>
                    <div className="collapse navbar-collapse"
id="navbarSupportedContent">
                        <ul className="navbar-nav me-auto mb-2 mb-lg-0">
                            <li className="nav-item">
                                <Link className="nav-link text-color"
aria-current="page" to="/">Projects</Link>
                            </li>
                            <li className="nav-item">
```

```
                                <Link className="nav-link text-color"
aria-current="page" to="/members">Members</Link>
                            </li>
                            <li className="nav-item">
                                <Link className="nav-link text-color"
aria-current="page" to="/tasks">Tasks</Link>
                            </li>
                            <li className="nav-item">
                                <select className="form-select"
onChange={handleProjectSelect} aria-label="Select project">
                                    <option value="all">All
Projects</option>
                                    {projects.map((project) => (
                                        <option key={project.project_id}
value={project.project_id}>{project.project_name}</option>
                                    ))}
                                </select>
                            </li>
                        </ul>

                        <button className="btn add-project-btn"
onClick={openModal}>Add Project</button>
                        {/* //Si se clickea el boton de Add Project el
modal se hace visible y muestra el formulario de Projectos */}
                        <Modal isOpen={isModalOpen} onClose={closeModal}>
                            <ProjectForm isEditing={false}
defaultValues={{ project_name: '', description: '', status: '' }}
onSubmitSuccess={closeModal} fetchProjects={fetchProjects}/>
                            {/* Este Formulario espera varios props ya que
puede manejar la logica de editar y crear proyecto, en este caso
                                ambas props estan desactivadas (de manera
logica), Tiene un props la funcion de onSubmitSuccess que cierra el
                                modal una vez completado exitosamente el
formulario
                            */}
                        </Modal>
                    </div>
                </div>
            </nav>
        </>
```

```
    )
}


export default Navbar
```

ProjectContainer.tsx

Componente contenedor de las tarjetas de proyectos para la ruta ‘/projects’

```
import React, { useEffect, useState } from 'react';
import ProjectCard from '../Cards/proyecto/ProjectCard';
import { Project } from '../../interfaces/Project';

interface ProjectsContainerProps {
    selectedProject: Project | null;
    projects: Project[];
    fetchProjects: () => void;
}
// Este componente manejara los props de projects y selectedProject que se
le pasaran desde
// el App.tsx
const ProjectsContainer: React.FC<ProjectsContainerProps> =
({selectedProject,  projects, fetchProjects}) => {

    // Devuelve la lista de Projectos
    return (
        <div className='container-fluid'>
            {projects.filter(project =>
                selectedProject === null || project.project_id ===
selectedProject.project_id
            ).map(filteredProject => (
                <ProjectCard key={filteredProject.project_id}
project={filteredProject} fetchProjects={fetchProjects}/>
            ))}
        </div>
    );
};
```

```
export default ProjectsContainer;
```

MembersContainers.py

Componente contenedor para las tarjetas MemberTeam referentes a los equipos de miembros de los proyectos. Este archivo tienen como responsabilidad el llamado a la api “get_all_members”, ademas como de elementos de carga ‘placeHolderCard’ y modal de error.

```
import React, { useEffect, useState } from 'react';

import { Project } from '../../interfaces/Project';
import { MemberTeam } from '../../interfaces/Member';
// Interfaces de los data object de: Project / MemberTeam

import ProjectTeam from '../Cards/member/ProjectTeam';
import PlaceholderCard from '../Cards/PlaceHolder';
import ErrorModal from '../Modal/ErrorModal';
// Tarjeta que representa el equipo de miembros por Projecto

interface MembersContainerProps {
    selectedProject: Project | null;
}
// Para manejar la logica de filtrado por producto individual, se pasa
este prop desde App hasta el renderizado de este componente


const MembersContainer: React.FC<MembersContainerProps> = ({
selectedProject }) => {
    const [members, setMembers] = useState<MemberTeam[]>([]);
    // Constante que maneja el estado y valor del array members

    const [loading, setLoading] = useState<boolean>(true);
    // Se maneja un estado que indica que la data esta cargando

    const [isErrorModalOpen, setIsErrorModalOpen] = useState(false);
    const [errorMessage, setErrorMessage] = useState<string>('');
    const handleCloseErrorModal = () => setIsErrorModalOpen(false);
    // Se definieron varios estados y una funcion para manejar la logica
```

```
    // de aparecer un modal cuando ocurra un error al llamar a la api


    //useEffect solicita a la api la data de get all members, hasta que se
complete mantiene el estado de loading true, de ocurri un
    // error se muestra el modal, de lo contrario se asigna la data a la
constante members
    useEffect(() => {
        fetch('http://172.16.5.78:5000/api/members')
            .then(response => {
                if (!response.ok) {
                    setErrorMessage('Network response was not ok');
                    setIsErrorModalOpen(true);
                }
                return response.json();
            })
            .then(data => {
                if (Array.isArray(data.members)) {
                    setMembers(data.members);
                    console.log('Members Fetch: ', data.members);
                } else {
                    setErrorMessage('An error ocurred while trying to
connect to the Data Base');
                    setIsErrorModalOpen(true);
                }
            })
            .catch(error => {
                console.error('There was a problem with the fetch
operation:', error);
                setErrorMessage('There was a problem connecting to the
server');
                setIsErrorModalOpen(true);
                setMembers([]);
            }).finally(() => setLoading(false));
    }, []);


    // El return devuelve un listado de los Project Teams cards segun la
data que se haya obtenido de la solicitud
```

```
    // En el proceso de cargar muestra unos placeholders, de ocurrir un
error lo muestra en pantalla.
    return (
        <>
            {loading ? (
                <div>
                    <PlaceholderCard />
                    <PlaceholderCard />
                    <PlaceholderCard />
                    <PlaceholderCard />
                    <PlaceholderCard />
                </div>
            ) : (
                members.filter(member =>
                    selectedProject === null || member.project_id ===
selectedProject.project_id
                ).map(filteredMember => (
                    <ProjectTeam key={filteredMember.project_id}
projectTeam={filteredMember} />
                ))
            )}
            <ErrorModal errorMessage={errorMessage}
isOpen={isErrorModalOpen} onClose={handleCloseErrorModal} />
        </>
    );
};

export default MembersContainer;
```

TaskContainer.tsx

Componente contenedor para las tarjetas TaskProject referentes a las tareas de los equipos de miembros de los proyectos. Este archivo tienen como responsabilidad el llamado a la api “get_all_tasks”, ademas como de elementos de carga ‘placeHolderCard’ y modal de error.

```
import React, { useEffect, useState } from 'react';
import { Task, ProjectTasks } from '../../interfaces/Task';
import TasksProject from '../Cards/tasks/TaskProject';
```

```
import { Project } from '../../interfaces/Project';
import ErrorModal from '../Modal/ErrorModal';
import PlaceholderCard from '../Cards/PlaceHolder';

interface TasksContainerProps {
    selectedProject: Project | null;
}

const TasksContainers: React.FC<TasksContainerProps> = ({ selectedProject
}) => {
    const [projectTasks, setProjectTasks] = useState<ProjectTasks[]>([]);


    const [loading, setLoading] = useState<boolean>(true);
    // Se maneja un estado que indica que la data esta cargando

    const [isErrorModalOpen, setIsErrorModalOpen] = useState(false);
    const [errorMessage, setErrorMessage] = useState<string>('');
    const handleCloseErrorModal = () => setIsErrorModalOpen(false);
    // Se definieron varios estados y una funcion para manejar la logica
    // de aparecer un modal cuando ocurra un error al llamar a la api


    useEffect(() => {
        fetch('http://172.16.5.78:5000/api/get_all_tasks')
            .then(response => {
                if (!response.ok) {
                    setErrorMessage('Network response was not ok');
                    setIsErrorModalOpen(true);
                }
                return response.json();
            })
            .then(data => {
                if (Array.isArray(data.data)) {
                    setProjectTasks(data.data);
                    console.log('Task Fetch: ', data.data);
                } else {
                    setErrorMessage('An error ocurred while trying to
connect to the Data Base');
                    setIsErrorModalOpen(true);
```

```
                }
            })
            .catch(error => {
                console.error('There was a problem with the fetch
operation:', error);
                setErrorMessage('There was a problem connecting to the
server');
                setIsErrorModalOpen(true);
                setProjectTasks([]);
            }).finally(() => setLoading(false));
    }, []);

    return (
        <>
            {loading ? (
                <div>
                    <PlaceholderCard />
                    <PlaceholderCard />
                    <PlaceholderCard />
                    <PlaceholderCard />
                    <PlaceholderCard />
                </div>
            ) : (projectTasks.filter(task =>
                selectedProject === null || task.project_id ===
selectedProject.project_id
            ).map(filteredTask => (
                <TasksProject key={filteredTask.project_id}
projectTask={filteredTask} />
            ))
            )}
            <ErrorModal errorMessage={errorMessage}
isOpen={isErrorModalOpen} onClose={handleCloseErrorModal} />
        </>
    );

};

export default TasksContainers;
```

ProjectCard.tsx

Representa un proyecto individual con la capacidad de ver su información, editar o eliminar el mismo. Posee elementos de carga, modales para formularios y manejo de errores.

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
//Hooks utilizados

import '../css/style.css';

import Modal from '../../Modal/modal';
import ProjectForm from '../../../forms/projects/ProjectForm';
import ErrorModal from '../../Modal/ErrorModal';
// Componentes Utilizados

import { Project } from '../../../interfaces/Project';
interface ProjectProps {
  project: Project;
  fetchProjects: () => void;
}
// Interface de data objects y funciones utilizados en el componente

const ProjectCard: React.FC<ProjectProps> = ({ project, fetchProjects }) 
=> {
  const navigate = useNavigate()
  // Navigate Hook inicializado para poder utilizarlo

  // Edit Modal Visibility Logic
  const [isEditModalOpen, setEditModalOpen] = useState(false);
  const openEditModal = () => setEditModalOpen(true);
  const handleCloseEditModal = () => setEditModalOpen(false);

  // Error Modal Visibility Logic
  const [errorMessage, setErrorMessage] = useState<string>('');
  const [isErrorModalOpen, setIsErrorModalOpen] = useState(false);
  const handleCloseErrorModal = () => setIsErrorModalOpen(false);

  // Delete Modal Visibility Logic
  const [isDeleteModalOpen, setDeleteModalOpen] = useState(false);
  const openDeleteModal = () => setDeleteModalOpen(true);
```

```
  const handleCloseDeleteModal = () => setDeleteModalOpen(false);


  // Handle Delete triggered by the Delete Modal
  const handleDelete = async () => {
    const url =
`http://127.0.0.1:5000/api/delete_project/${project.project_id}`
    setDeleteModalOpen(false)

    try {
      const response = await fetch(url, {
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
        }
      })

      const data = await response.json();

      if (response.ok) {
        console.log('Success deleting Project')
        fetchProjects()
        navigate('/blank');
        navigate(-1)
      } else {
        console.log('Error deleting Project')
        setErrorMessage(data.error);
        setIsErrorModalOpen(true);
      }

    } catch (error) {
      console.error('Failed to delete project:', error);

      if (error instanceof Error) {
        setErrorMessage(error.message);
      } else {
        setErrorMessage('An unexpected error occurred');
      }
      setIsErrorModalOpen(true);
    }
```

```
  }



  return (

    <>
      <div className="card mb-4">
        <div className="card-body">
          <div className="row">
            <div className="col">
              <div className="d-flex justify-content-between
align-items-center">
                <h5 className="card-title
mb-0">{project.project_name}</h5>
                <div className="d-none d-md-block">
                  <button className="btn edit-button me-2"
onClick={openEditModal}><i className="fas fa-edit"></i> Edit</button>
                  <button className="btn btn-danger"
onClick={openDeleteModal}><i className="fas fa-trash"></i> Delete</button>
                </div>
              </div>
            </div>
          </div>
          <div className="row d-md-none">
            <div className="col mt-3">
              <button className="btn edit-button mb-2 w-100"
onClick={openEditModal}><i className="fas fa-edit"></i> Edit</button>
              <button className="btn btn-danger w-100"
onClick={openDeleteModal}><i className="fas fa-trash"></i> Delete</button>
            </div>
          </div>
          <div className="row">
            <div className="col">
              <hr className="d-none d-md-block" />
              <p className="card-text mt-3">Description:
{project.description}</p>
              <p className="card-text">Status: {project.status}</p>
            </div>
          </div>
```

```
        </div>
      </div>

    {/*
      Edit Modal With the asign default Values
    */}
      <Modal isOpen={isEditModalOpen} onClose={handleCloseEditModal} >
        <ProjectForm
          isEditing={true}
          defaultValues={
            {
              project_name: project.project_name,
              description: project.description,
              status: project.status,
              project_id: project.project_id
            }}
          onSubmitSuccess={handleCloseDeleteModal}
          handleCloseEditModal={handleCloseEditModal}
          fetchProjects={fetchProjects}
        />
      </Modal>

      {/*
      Delete Modal
       */}
      <Modal isOpen={isDeleteModalOpen} onClose={handleCloseDeleteModal}>
        <div className="d-flex justify-content-center">
          <div style={{ backgroundColor: 'white', padding: '20px' }}>
            <div>
              <h1>Delete Project</h1>
              <p>Are you sure to delete the project
{project.project_name}?</p>
              <button className="btn btn-danger me-2"
onClick={handleDelete} style={{ padding: '5px 10px', fontSize: '1.2rem'
}}>
                <i className="fas fa-trash me-1"></i>Delete
              </button>
              <button className="btn btn-secondary "
onClick={handleCloseDeleteModal} style={{ padding: '5px 10px', fontSize:
'1.2rem' }}>
```

```
                Cancel
              </button>
            </div>
          </div>
        </div>
      </Modal>
       {/*
       Error Modal
       */}
      <ErrorModal errorMessage={errorMessage} isOpen={isErrorModalOpen}
onClose={handleCloseErrorModal} />
    </>
  );
};

export default ProjectCard;
```

ProjectTeam.tsx

Representa al equipo de miembros, con opciones de agregar los mismos. A su vez renderiza la lista de miembros a través del member Card.

```
import React, { useEffect, useState } from "react";
import '../css/style.css';

import MemberCard from "./MemberCard";
import Modal from "../../Modal/modal";
import MemberForm from "../../../forms/members/MemberForm";
// Componentes Utilizados En este archivo, Modal and MemberForm to handle
adding member
// MemberCard to render each Members

import { Member, MemberTeam } from "../../../interfaces/Member";
interface ProjectTeamProps {
  projectTeam: MemberTeam;
}
// Interfaces de los data Objects a utilizar
```

```
const ProjectTeam: React.FC<ProjectTeamProps> = ({ projectTeam }) => {
  const [members, setMembers] = useState<Member[]>([])
  // Constante que maneja el array de members

  const [isModalOpen, setModalOpen] = useState(false);
  const openModal = () => setModalOpen(true);
  const closeModal = () => setModalOpen(false);
  // Logica para manejar el formulario/Modal


  useEffect(() => {
    setMembers(projectTeam.members || []);
  }, [projectTeam.members]);
  // Al cargarse el componente se le asigna los miembros del projectTeam
dta Object

  return (
    <div className="members">
    <div className="card mb-4">
      <div className="card-body">
        <div className="d-flex justify-content-between
align-items-center">
          <h5 className="card-title mb-0">{projectTeam.project_name}</h5>
          <div>
            <div className="d-lg-none">
              <button className="btn add-buttons" onClick={openModal}> Add
Member </button>
            </div>
            <div className="d-none d-lg-block">
              <button className="btn add-buttons px-5" onClick={openModal}
style={{ width: '190px' }}> Add Member </button>
            </div>
          </div>
        </div>
        <hr />
        <br></br>
        <Modal isOpen={isModalOpen} onClose={closeModal}>
          <MemberForm isEditing={false}
defaultValues={{project_id:projectTeam.project_id}}
```

```
onSubmitSuccess={closeModal}
handleCloseEditModal={closeModal}></MemberForm>
        </Modal>
        {/*
          Modal Para agregar miembro asociado con el project_id Correcto
        */}

        <div>
          {members.map(member => (
            <MemberCard key={member.member_id} member={member}
project_id={projectTeam.project_id}/>
          ))}
          {/*
            Renders memberCards for each members inside the members
          */}
        </div>
      </div>
    </div>
  </div>

  );
};

export default ProjectTeam
```

MemberCard.tsx

Representa a cada miembro individual con opciones de editar/ eliminar al miembro.

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
// Se importan los hooks que se utilizaran en este componente

import { Member } from "../../../interfaces/Member";
// Interfaz del data object Member

import Modal from "../../Modal/modal";
import MemberForm from "../../../forms/members/MemberForm";
import ErrorModal from "../../Modal/ErrorModal";
// Componentes a utilizar
```

```tsx
interface MembersProps {
  member: Member;
  project_id?: number | undefined;
}
// Se definen los props a utilizar


const MemberCard: React.FC<MembersProps> = ({ member, project_id }) => {
  const [errorMessage, setErrorMessage] = useState<string>('');
  // const para manejar y establecer el mensaje de error

  const [isErrorModalOpen, setIsErrorModalOpen] = useState(false);
  const handleCloseErrorModal = () => setIsErrorModalOpen(false);
  //Error Visibility Modal Logic

  const [isEditModalOpen, setEditModalOpen] = useState(false);
  const openEditModal = () => setEditModalOpen(true);
  const handleCloseEditModal = () => setEditModalOpen(false);
  // Edit Visibility Modal Logic

  const [isDeleteModalOpen, setDeleteModalOpen] = useState(false);
  const openDeleteModal = () => setDeleteModalOpen(true);
  const handleCloseDeleteModal = () => setDeleteModalOpen(false);
  // Delete Visibility Modal Logic


  const navigate = useNavigate()
  // Inicializando el hook en la const navigate para utilizarlo


  // Delete Fetch Triggered by the Modal
  const handleDelete = async () => {
    const url =
`http://127.0.0.1:5000/api/delete_member/${member.member_id}`
    setDeleteModalOpen(false)
    // Se Oculta el Delete Modal

    try {
      const response = await fetch(url, {
```

```
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
        }
      })
      // Se realiza el fetch con todos los parapretos requeridos y se
guarda su respuesta en el const response

      const data = await response.json();
      // Se asigma la data del response en formato json

      // If / else manejan el estado de la respuesta Succes/Failed
      if (response.ok) {
        console.log('Success deleting Member')
        navigate('/blank');
        navigate(-1)
        // Este patron simula una recarga de la pagina, navegando en a una
pagina innexistente y volviendo a la actual

      } else {
        console.log('Error deleting Member')
        setErrorMessage(data.error);
        setIsErrorModalOpen(true);
        //De el response resultar falso se le asigna el error al const
errorMessage y se hace visible el modal
      }

    } catch (error) {
      console.error('Failed to delete member:', error);
      if (error instanceof Error) {
        setErrorMessage(error.message);
      } else {
        setErrorMessage('An unexpected error occurred');
      }
      setIsErrorModalOpen(true);
    }
  }

  return (<>
    <div className="card-body">
```

```
      <div className="row mb-4">
        <div className="col-md-3">
          <p className="card-text"><strong>Name:</strong>
{member.member_name}</p>
        </div>
        <div className="col-md-3">
          <p className="card-text"><strong>Id:</strong>
{member.member_id}</p>
        </div>
        <div className="col-md-3 mb-2">
          <p className="card-text"><strong>Role:</strong>
{member.role}</p>
        </div>
        <div className="col-md-3">
          <div className="d-grid gap-2 d-md-flex justify-content-md-end">
            <button className="btn edit-button me-md-2 mb-2 mb-md-0 "
onClick={openEditModal}>
              <i className="fas fa-edit"></i> Edit
            </button>
            <button className="btn btn-danger " onClick={openDeleteModal}>
              <i className="fas fa-trash"></i> Delete
            </button>
          </div>
        </div>
      </div>
    </div>
    <hr />

    {/*
      Edit Modal With the asign default Values
    */}
    <Modal isOpen={isEditModalOpen} onClose={handleCloseEditModal}>
      <div className="d-flex flex-column align-items-center w-100">
        <MemberForm
          isEditing={true}
          defaultValues={{
            member: {
              member_id: member.member_id,
              member_name: member.member_name,
              role: member.role
```

```
            },
            project_id: project_id
          }}
          onSubmitSuccess={handleCloseEditModal}
          handleCloseEditModal={handleCloseEditModal}
        />
      </div>
    </Modal>

    {/* Delete Modal */}
    <Modal isOpen={isDeleteModalOpen} onClose={handleCloseDeleteModal}>
      <div className="d-flex justify-content-center">
        <div style={{ backgroundColor: 'white', padding: '20px' }}>
          <div>
            <h1>Delete Member</h1>
            <p>Are you sure to delete the member {member.member_name}?</p>
            <button className="btn btn-danger btn-lg me-2"
onClick={handleDelete} style={{ padding: '5px 10px', fontSize: '1.2rem'
}}>
              <i className="fas fa-trash me-1"></i>Delete
            </button>
            <button className="btn btn-secondary btn-lg"
onClick={handleCloseDeleteModal} style={{ padding: '5px 10px', fontSize:
'1.2rem' }}>
              Cancel
            </button>
          </div>
        </div>
      </div>
    </Modal>

    {/* Error Modal */}
    <ErrorModal errorMessage={errorMessage} isOpen={isErrorModalOpen}
onClose={handleCloseErrorModal} />

  </>
  );
};

export default MemberCard
```

TaskProject.tsx

Representa el componente de tareas referente a un proyecto. Renderiza una lista de tareas, además de dar la opción de crear una tarea.

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

import '../css/style.css';

import TaskCard from "./TaskCard";
import Modal from "../../Modal/modal";
import TaskForm from "../../../forms/tasks/TaskForm";
//COmponentes Utilizados

import { ProjectTasks, Task } from "../../../interfaces/Task";
//Data object Interfaces
interface ProjectTasksProps {
  projectTask: ProjectTasks;
}


const TasksProject: React.FC<ProjectTasksProps> = ({ projectTask }) => {
  const [tasks, setTasks] = useState<Task[]>([]);
  // const Representativo de las tareas en cada proyecto

  // Modal for create task Logic
  const [isModalOpen, setModalOpen] = useState(false);
  const openModal = () => setModalOpen(true);
  const closeModal = () => setModalOpen(false);

  //Al montar el Componente se asignan las tareas al const tasks
  useEffect(() => {
    setTasks(projectTask.tasks);
  }, [projectTask.tasks]);


  return (
    <>
      <div className="tasks">
        <div className="card mb-4">
```

```
          <div className="card-body">
            <div className="d-flex justify-content-between
align-items-center">
              <h5 className="card-title
mb-0">{projectTask.project_name}</h5>
              <div>
                <div className="d-lg-none">
                  <button className="btn add-buttons" onClick={openModal}>
Add Task </button>
                </div>
                <div className="d-none d-lg-block">
                  <button className="btn add-buttons px-5"
onClick={openModal} style={{ width: '190px' }}> Add Task </button>
                </div>
              </div>
            </div>
            <hr />
            <br></br>
            <div>
              {/* Se renderizan todas las tareas del projecto */}
              {tasks.map(task => (
                <TaskCard key={task.task_id} task={task}
project_id={projectTask.project_id} />
              ))}
            </div>
          </div>
        </div>
      </div>

      {/* Modal for creating a task */}
      <Modal isOpen={isModalOpen} onClose={closeModal}>
        <TaskForm isEditing={false} defaultValues={{ project_id:
projectTask.project_id }} onSubmitSuccess={closeModal}
handleCloseEditModal={closeModal}></TaskForm>
      </Modal>
    </>
  );
};

export default TasksProject
```

TaskCard.tsx

Representa una tarea en específico, con la posibilidad de editar o eliminar la misma.

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

import '../css/style.css';

import TaskCard from "./TaskCard";
import Modal from "../../Modal/modal";
import TaskForm from "../../../forms/tasks/TaskForm";
//COmponentes Utilizados

import { ProjectTasks, Task } from "../../../interfaces/Task";
//Data object Interfaces
interface ProjectTasksProps {
  projectTask: ProjectTasks;
}


const TasksProject: React.FC<ProjectTasksProps> = ({ projectTask }) => {
  const [tasks, setTasks] = useState<Task[]>([]);
  // const Representativo de las tareas en cada proyecto

  // Modal for create task Logic
  const [isModalOpen, setModalOpen] = useState(false);
  const openModal = () => setModalOpen(true);
  const closeModal = () => setModalOpen(false);

  //Al montar el Componente se asignan las tareas al const tasks
  useEffect(() => {
    setTasks(projectTask.tasks);
  }, [projectTask.tasks]);


  return (
    <>
      <div className="tasks">
        <div className="card mb-4">
          <div className="card-body">
```

```
            <div className="d-flex justify-content-between
align-items-center">
              <h5 className="card-title
mb-0">{projectTask.project_name}</h5>
              <div>
                <div className="d-lg-none">
                  <button className="btn add-buttons" onClick={openModal}>
Add Task </button>
                </div>
                <div className="d-none d-lg-block">
                  <button className="btn add-buttons px-5"
onClick={openModal} style={{ width: '190px' }}> Add Task </button>
                </div>
              </div>
            </div>
            <hr />
            <br></br>
            <div>
              {/* Se renderizan todas las tareas del projecto */}
              {tasks.map(task => (
                <TaskCard key={task.task_id} task={task}
project_id={projectTask.project_id} />
              ))}
            </div>
          </div>
        </div>
      </div>

      {/* Modal for creating a task */}
      <Modal isOpen={isModalOpen} onClose={closeModal}>
        <TaskForm isEditing={false} defaultValues={{ project_id:
projectTask.project_id }} onSubmitSuccess={closeModal}
handleCloseEditModal={closeModal}></TaskForm>
      </Modal>
    </>
  );
};

export default TasksProject
```

PlaceholderCard.tsx

Componente que se muestra para simular la carga de elementos

```
const PlaceholderCard = () => {
  return (
    <div style={{ border: '1px solid #ccc', margin: '10px', padding:
'20px', borderRadius: '5px', backgroundColor: '#eee' }}>
      <div style={{ height: '20px', backgroundColor: '#ddd', width: '80%',
marginBottom: '10px' }}></div>
      <div style={{ height: '15px', backgroundColor: '#ddd', width: '50%'
}}></div>
    </div>
  );
};

export default PlaceholderCard;
```

Modal.tsx

Componente reutilizable que representa un modal, se le pasa un formulario (Project/Member/Task) como componente hijo y lo muestra. Posee su propia logica para manejar su visibilidad.

```
import React from 'react';
import 'bootstrap-icons/font/bootstrap-icons.css';

interface ModalProps {
  isOpen?: boolean;
  onClose: () => void;
  children: React.ReactNode;
}
// Se definen los props a utilizar en el modal

const Modal: React.FC<ModalProps> = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;
  //Si el modal isOpen es Falso no muestra nada en pantalla
```

```
  // De lo contrario renderiza el Modal con la Informacion Correspondiente
(Formulario as Children)

  return (
    <>
<div className="modal" tabIndex={-1} style={{ display: 'block',
backgroundColor: 'rgba(0, 0, 0, 0.5)' }}>
  <div className="modal-dialog modal-lg" style={{ width: '80%', maxHeight:
'80%', position: 'fixed', top: '50%', left: '50%', transform:
'translate(-50%, -50%)' }}>
    <div className="modal-content" style={{ backgroundColor: 'white',
padding: '20px' }}>
      <div className="modal-body">
        {children}
      </div>
      <button onClick={onClose} type="button" className="btn-close"
aria-label="Close" style={{ position: 'absolute', top: '10px', right:
'10px' }}></button>
    </div>
  </div>
</div>
</>

  );
};

export default Modal;
```

ErrorModal.tsx

Componente reutilizable para presentar mensajes de error.

```
import React from 'react';

// Se le asignan los props que recibirá el componente
// errorMessage - El mensaje que mostrará el Modal
// isOpen and onClose manejan la lógica de la visibilidad del componente
```

```tsx
interface ErrorModalProps {
  errorMessage: string;
  isOpen: boolean;
  onClose: () => void;
}



const ErrorModal: React.FC<ErrorModalProps> = ({ errorMessage, isOpen,
onClose }) => {
  if (!isOpen) return null;
  //Si el modal isOpen es Falso no muestra nada en pantalla
  // De lo contrario renderiza el Modal con la Información Correspondiente

  return (
    <div style={{ position: 'fixed', top: '50%', left: '50%', transform:
'translate(-50%, -50%)', backgroundColor: 'white', padding: '20px',
zIndex: 1000 }}>
      <div style={{ backgroundColor: 'white', padding: '20px' }}>
        <div>
          <h4>{errorMessage}</h4>
          <br></br>
          <button onClick={onClose} className="btn btn-secondary" style={{
padding: '10px 20px', fontSize: '1.2rem' }}>Close</button>
        </div>
      </div>
    </div>

  );
};

export default ErrorModal;
```

ProjectForm.tsx

Presenta formulario para editar y crear proyectos.

```tsx
import React from 'react';
import { useForm, SubmitHandler } from 'react-hook-form';
```

```tsx
import { zodResolver } from '@hookform/resolvers/zod';
import { useNavigate} from 'react-router-dom';
// Hooks y modulos necesarios

import { projectSchema } from './SchemaValidation';
import {Project} from '../../interfaces/Project';
// Schema e Interfaz Utilizada

interface ProjectFormProps {
  defaultValues: Project;
  isEditing: boolean;
  onSubmitSuccess: () => void; // Call Back
  handleCloseEditModal?: () => void;
  fetchProjects: () => void;
}
// Props logicas para la funcionalidad del form


const ProjectForm: React.FC<ProjectFormProps> = ({ defaultValues,
isEditing , onSubmitSuccess, handleCloseEditModal, fetchProjects}) => {
  const navigate = useNavigate()


  const {
    register,
    handleSubmit,
    reset,
    formState: { errors }
  } = useForm<Project>({
    resolver: zodResolver(projectSchema),
    defaultValues
  });
  //Inicializacion de la funcionalidades de react-hook-form


  React.useEffect(() => {
    console.log('Resetting form with defaultValues:', defaultValues);
    reset(defaultValues);
  }, [defaultValues, reset]);
```

```
  // Al momneto de montar el componente se asignan los default values al
editar


  const onSubmit: SubmitHandler<Project> = async data => {
    console.log('Attempting to submit form', data);

    const url = isEditing ?
`http://172.16.5.78:5000/api/update_project/${defaultValues.project_id}` :
'http://172.16.5.78:5000/api/new_project';
    const method = isEditing ? 'PUT' : 'POST';

    console.log('Making API request to:', url);

    try {
        const response = await fetch(url, {
            method: method,
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(data)
        });

        if (!response.ok) throw new Error('Network response was not ok.');

        const api_response = await response.json();
        console.log('Success:', api_response);
        onSubmitSuccess();
        fetchProjects();
        navigate('/blank');
        navigate(-1);

    } catch (error) {
        console.error('Error:', error);
        alert('Hubo un error procesando tu solicitud.');
    }



  };
```

```
  console.log('Form errors:', errors);


  //Inputs que se asocial a la validacion de SchemaValidation de zod
resolver con react-hook-form
  return (
    <>
  <form onSubmit={handleSubmit(onSubmit)} className="mt-4">
  <h1 className="mb-4 text-center text-color-two">{isEditing ? 'Updating
Project' : 'Create Project'}</h1>
  <div className="mb-3">
    <label htmlFor="project_name" className="form-label
text-color-two">Project Name</label>
    <input {...register('project_name')} type="text"
className={`form-control ${errors.project_name ? 'is-invalid' : ''}`}
id="project_name" />
    {errors.project_name && <div
className="invalid-feedback">{errors.project_name.message}</div>}
  </div>

  <div className="mb-4">
    <label htmlFor="description" className="form-label
text-color-two">Description</label>
    <textarea {...register('description')} className={`form-control
${errors.description ? 'is-invalid' : ''}`} id="description"
rows={4}></textarea>
    {errors.description && <div
className="invalid-feedback">{errors.description.message}</div>}
  </div>

  <div className="mb-3">
    <label htmlFor="status" className="form-label
text-color-two">Status</label>
    <select {...register('status')} className={`form-select
${errors.status ? 'is-invalid' : ''}`} id="status">
      <option value="Completed">Completed</option>
      <option value="In progress">In progress</option>
      <option value="Not Started">Not Started</option>
    </select>
```

```
    {errors.status && <div
className="invalid-feedback">{errors.status.message}</div>}
  </div>

  <div className="mt-3">
    <input type="submit" className="btn btn-primary me-3 btn-lg"
onClick={() => console.log('Submit clicked')}></input>
    <button onClick={handleCloseEditModal} className="btn btn-secondary
me-3 btn-lg">Cancel</button>
  </div>
</form>

  </>
  );
};

export default ProjectForm;
```

ProjectSchemaValidation.tsx

Realiza las validaciones necesarias para los campos de proyectos.

```
import { z } from 'zod';

//Documento que representa las validaciones antes de enviar el formulario


const projectSchema = z.object({
  id: z.number().optional(),
  project_name: z.string().min(1, "Project name is required"),
  description: z.string().min(10, "Description should be at least 10
characters long"),
  status: z.enum(["Completed", "Progress", "Not Started"]),
});

export { projectSchema };
```

TaskForm.tsx

Presenta formulario para editar y crear tasks.

```
// src/components/ProjectForm.tsx
import React, { useEffect, useState } from 'react';
import { useForm, SubmitHandler } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { useNavigate } from 'react-router-dom';
// Hooks y modulos utilizados

import { taskSchema } from './SchemaValidation';
import {Task }from '../../interfaces/Task';
//Schema e interfaz representando los task

interface TaskFormProps {
  defaultValues: {
    task?: Task;
    project_id?: number;
  };
  isEditing: boolean;
  onSubmitSuccess: () => void;
  handleCloseEditModal: () => void;
}
//Props para la funcionalidad logica del form

interface MemberList {
  member_id: number;
  member_name: string;
}
// Interfaz para las lista de miembros de un proyecto, para asociarlo a un
task

const TaskForm: React.FC<TaskFormProps> = ({ defaultValues, isEditing,
onSubmitSuccess, handleCloseEditModal }) => {
  const [members, setMembers] = useState<MemberList[]>([]);
  const { task, project_id } = defaultValues;
  const navigate = useNavigate()
```

```
  React.useEffect(() => {
    if (project_id) {

fetch(`http://172.16.5.78:5000/api/get_members_by_project/${project_id}`)
        .then(response => response.json())
        .then(setMembers)
        .catch(console.error);
    }
  }, [project_id]);
  // Se llama a los members del proyecto para asociarlos con los ID


  //Formating Date
  const formatDate = (dateString?: string): string => {
    if (!dateString) return '';

    const date = new Date(dateString);
    const year = date.getFullYear();
    const month = (date.getMonth() + 1).toString().padStart(2, '0');
    const day = date.getDate().toString().padStart(2, '0');

    return `${year}-${month}-${day}`;
  };


  const formattedTaskValues: Task = {
    ...task,
    project_id: project_id,
    start_date: formatDate(task?.start_date),
    end_date: formatDate(task?.end_date),
  };



  // console.log("Initial default values:", defaultValues);


  interface FormValues extends Task {
    project_id?: number;
  }
```

```
  const {
    register,
    handleSubmit,
    reset,
    formState: { errors }
  } = useForm<FormValues>({
    resolver: zodResolver(taskSchema),
    defaultValues: {
      ...formattedTaskValues,
      project_id
    }
  });
  // Al momneto de montar el componente se asignan los default values al
editar



  React.useEffect(() => {
    console.log('Resetting form with defaultValues:', defaultValues);
    reset({
      ...formattedTaskValues,
      project_id
    });
  }, [defaultValues, reset]);
  // Al momneto de montar el componente se asignan los default values al
editar

  console.log("Initial default values for form:", {
    ...formattedTaskValues,
    project_id
  })



  // Submit Logic que depende la variable logica isEditing para editar o
crear
  const onSubmit: SubmitHandler<FormValues> = async (data: FormValues) =>
{
    console.log('Data for updating task is ', data);
    console.log('Member id is  ', data.member_id);
```

```
    const url = isEditing ?
`http://172.16.5.78:5000/api/update_task/${defaultValues.task?.task_id}/${
data.member_id}` :
`http://172.16.5.78:5000/api/new_task/${data.project_id}/${data.member_id}
`;
    const method = isEditing ? 'PUT' : 'POST';


    try {
        const response = await fetch(url, {
            method: method,
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(data)
        });

        if (!response.ok) throw new Error('Network response was not ok.');

        const api_response = await response.json();
        console.log('Success:', api_response);
        onSubmitSuccess();
        navigate('blank');
        navigate(-1)

    } catch (error) {
        console.error('Error:', error);
        alert('Hubo un error procesando tu solicitud.');
    }



  };

  console.log('Form errors:', errors);

  //Inputs que se asocial a la validacion de SchemaValidation de zod
resolver con react-hook-form
```

```
  return (
<form onSubmit={handleSubmit(onSubmit)} className="mt-4">
  <h1 className="mb-4 text-center text-color-two">{isEditing ? 'Edit Task'
: 'Create Task'}</h1>
  <div className="mb-3">
    <label htmlFor="task_name" className="form-label text-color-two">Task
Name</label>
    <input {...register('task_name')} type="text" className={`form-control
${errors.task_name ? 'is-invalid' : ''}`} id="task_name" />
    {errors.task_name && <div
className="invalid-feedback">{errors.task_name.message}</div>}
  </div>

  <div className="mb-3">
    <label htmlFor="start_date" className="form-label
text-color-two">Start Date</label>
    <input {...register('start_date')} type="date"
className={`form-control ${errors.start_date ? 'is-invalid' : ''}`} />
    {errors.start_date && <div
className="invalid-feedback">{errors.start_date.message}</div>}
  </div>

  <div className="mb-3">
    <label htmlFor="end_date" className="form-label text-color-two">End
Date</label>
    <input {...register('end_date')} type="date" className={`form-control
${errors.end_date ? 'is-invalid' : ''}`} id="end_date" />
    {errors.end_date && <div
className="invalid-feedback">{errors.end_date.message}</div>}
  </div>

  <div className="mb-3">
    <label htmlFor="member_id">Member</label>
    <select {...register('member_id')} className={`form-select
${errors.member_id ? 'is-invalid' : ''}`}>
      <option value="">Select a Member</option>
      {members.map(member => (
        <option key={member.member_id}
value={member.member_id}>{member.member_name}</option>
      ))}
```

```
    </select>
    {errors.member_id && <div
className="invalid-feedback">{errors.member_id.message}</div>}
  </div>
  <div className="mt-3">
    <input type="submit" className="btn btn-primary me-3 btn-lg"
onClick={() => console.log('Submit clicked')}></input>
    <button onClick={handleCloseEditModal} className="btn btn-secondary
me-3 btn-lg">Cancel</button>
  </div>
</form>



  );
};

export default TaskForm;
```

TaskSchemaValidation.tsx

Realiza las validaciones necesarias para los campos de Tasks.

```
import { z } from 'zod';

//Documento que representa las validaciones antes de enviar el formulario

const taskSchema = z.object({
  task_name: z.string().min(1, { message: "Task name is required." }),
  project_id: z.number().min(1, { message: "Task name is required." }),
  member_id: z.string().min(1, { message: "Member od is required." }),
  start_date: z.string()
    .optional(),
  end_date: z.string()
    .optional()
});
```

```
export { taskSchema };
```

MemberForm.tsx

Presenta formulario para editar y crear miembros.

```
// src/components/ProjectForm.tsx
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { useForm, SubmitHandler } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
// Hooks y modulos necesarios

import { memberSchema } from './SchemaValidation';
import { Member } from '../../interfaces/Member';
// Schema e Interfaz Utilizada


interface MemberFormProps {
  defaultValues:  {
    member?: Member;
    project_id?: number;
  };
  isEditing: boolean;
  onSubmitSuccess: () => void;
  handleCloseEditModal: () => void;
}
// Props logicas para la funcionalidad del form

const MemberForm: React.FC<MemberFormProps> = ({ defaultValues, isEditing
, onSubmitSuccess, handleCloseEditModal}) => {
  const navigate = useNavigate()


  interface FormValues extends Member{
    project_id?: number;
  }
  // Se extiende las propiedades de members pasandole el project_id
```

```
  const {
    register,
    handleSubmit,
    reset,
    formState: { errors }
  } = useForm<FormValues>({
    resolver: zodResolver(memberSchema),
    defaultValues: {
      member_name: defaultValues.member?.member_name,
      role: defaultValues.member?.role,
      project_id: defaultValues.project_id,
      member_id: defaultValues.member?.member_id
    }
  });
  //Inicializacion de la funcionalidades de react-hook-form


  React.useEffect(() => {
    console.log('Resetting form with defaultValues:', defaultValues);
    reset(defaultValues);
  }, [defaultValues, reset]);
  // Al momneto de montar el componente se asignan los default values al
editar


  // Submit Logic que depende la variable logica isEditing para editar o
crear
  const onSubmit: SubmitHandler<Member> = async data => {
    console.log('Form data', data);

    const url = isEditing ?
`http://172.16.5.78:5000/api/update_member/${defaultValues.project_id}/${d
efaultValues.member?.member_id}` :
`http://172.16.5.78:5000/api/add_members/${defaultValues.project_id}`;
    const method = isEditing ? 'PUT' : 'POST';


    try {
        const response = await fetch(url, {
            method: method,
```

```
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(data)
        });

        if (!response.ok) throw new Error('Network response was not ok.');

        const api_response = await response.json();
        console.log('Success:', api_response);
        onSubmitSuccess();
        navigate('/blank');
        navigate(-1);

    } catch (error) {
        console.error('Error:', error);
        alert('Hubo un error procesando tu solicitud.');
    }



  };

  console.log('Form errors:', errors);


  //Inputs que se asocial a la validacion de SchemaValidation de zod
resolver con react-hook-form
  return (
<form onSubmit={handleSubmit(onSubmit)} className="mt-4 w-100">
  <h1 className='text-center text-color-two'>{isEditing ? 'Edit Member' :
'Add Member'}</h1>

  <input type="hidden" {...register('project_id')} />

  <div className="mb-3">
    <label htmlFor="member_name" className="form-label
text-color-two">Member Name</label>
```

```
    <input {...register('member_name')} type="text"
className={`form-control ${errors.member_name ? 'is-invalid' : ''}`}
id="member_name" />
    {errors.member_name && <div
className="invalid-feedback">{errors.member_name.message}</div>}
  </div>
  <div className="mb-3">
    <label htmlFor="role" className="form-label
text-color-two">Role</label>
    <input {...register('role')} type="text" className={`form-control
${errors.role ? 'is-invalid' : ''}`} id="role" />
    {errors.role && <div
className="invalid-feedback">{errors.role.message}</div>}
  </div>

  <div className="mt-3">
    <input type="submit" className="btn btn-primary me-3 btn-lg"
onClick={() => console.log('Submit clicked')}></input>
    <button onClick={handleCloseEditModal} className="btn btn-secondary
me-3 btn-lg">Cancel</button>
  </div>
</form>



  );
};

export default MemberForm;
```

MemberSchemaValidation.tsx

Realiza las validaciones necesarias para los campos de miembros.

```
import { z } from 'zod';

//Documento que representa las validaciones antes de enviar el formulario
```

```
const memberSchema = z.object({
  project_id: z.number(),
  member_name: z.string().min(1, "Member name is required"),
  role: z.string().min(1, "Member role is required")
});

export { memberSchema };
```

Interfaces

Representan los data objects a utilizar, util para manejo de tipado y errores además de representar Data Transfer Objects.

Members.ts

```
// Data Object que representa un Miembro
export interface Member {
    member_id?: number;
    member_name?: string;
    role?: string;
    project_id?: number;
}


//Data object que representa el equipo de trabajo de un proyecto
export interface MemberTeam {
    project_id?: number;
    project_name?: string;
    members?: Member[];
}
```

Tasks.ts

```
//Data object que representa una tarea, esta interfaz es
//utilizada tanto para crear/editar y plasmar la informacion de un task
export interface Task {
  task_id?: number;
  member_id?: number;
```

```
  member_name?: string;
  project_id?: number;
  task_name?: string;
  start_date?: string;
  end_date?: string;
}


// Data object que representa el conjunto de tareas de un proyecto
export interface ProjectTasks {
  project_id?: number;
  project_name?: string;
  tasks: Task[];
}
```

Project.ts

```
// Data Object que representa un proyecto
export interface Project {
    description?: string;
    project_id?: number;
    project_name?: string;
    status?: string;
  }
```

CSS

```
/*adds styles to the cards*/

/*adds styles to the cards*/
```

```
.card {

    margin: 15px 15px 15px 15px;

    border: 1px solid black;



    border: 1px solid black;



}



/*header styles of the card */

/*header styles of the card */

.card-body {

    border-radius: 5px;

    padding: 20px;

    /* background-color:  #486980;     */

    /* background-color:  #a2ceec; */

    background-color:  #dee1e4;

    color: #1c1c1d;

    /* background-color:  #486980;     */

    /* background-color:  #a2ceec; */

    background-color:  #dee1e4;

    color: #1c1c1d;

}
```

```
.card-title {
    font-size: 20px;
    margin-bottom: 10px;
}

.card-text {
    font-size: 16px;
    color: #b0b8be;
    color: #b0b8be;
}



/*adds color to the navbar*/
/*adds color to the navbar*/
.custom-navbar {
    background-color: #092866 !important;
    color: #f8f9fa;
    background-color: #092866 !important;
    color: #f8f9fa;
  }

  /*nav bar styles*/
  .nav-text-color {
```

```
  color: rgb(244, 246, 250) ;
}

.nav-text-color:hover {
  color: rgb(194, 194, 231);
}
/*nav bar styles*/
.nav-text-color {
  color: rgb(244, 246, 250) ;
}

.nav-text-color:hover {
  color: rgb(194, 194, 231);
}

.navbar-brand-bold {
  font-weight: bold;
}

/*hamburger color*/
.navbar-toggler.navbar-hamburger {
  background-color: #ffffff;
}
```

```
/*text styles*/


  .text-color-one {
    color: rgb(18, 18, 19);
  }


  .text-color-two {
    color: rgb(23, 23, 24);
  }


  .text-color:hover {
    color: rgb(228, 228, 230);
  }



  /*In large viewports, it applies this specific width*/
  @media (min-width: 992px) {
    .add-project-btn {
      width: 210px !important;
    }
  }
```

```
  /*button styles*/

  /*hamburger color*/

  .navbar-toggler.navbar-hamburger {

    background-color: #ffffff;

  }


/*text styles*/


  .text-color-one {

    color: rgb(18, 18, 19);

  }


  .text-color-two {

    color: rgb(23, 23, 24);

  }


  .text-color:hover {

    color: rgb(228, 228, 230);

  }



  /*In large viewports, it applies this specific width*/

  @media (min-width: 992px) {
```

```
  .add-project-btn {
    width: 210px !important;
  }
}

/*button styles*/
.add-project-btn {
  background-color:#1bb341;
  background-color:#1bb341;
  color: white;
}
.add-project-btn:hover {
  background-color: #158a32;
  color: white;

}
.add-project-btn:hover {
  background-color: #158a32;
  color: white;

}

.add-buttons{
```

```
  background-color: #1a8f8f;
  background-color: #1a8f8f;
  color: white;
}

.add-buttons:hover{
  background-color:  #116e6e;
  color: white;
}

.add-buttons:hover{
  background-color:  #116e6e;
  color: white;
}

.edit-button{
  background-color: #da7d40;
  background-color: #da7d40;
  color: white;
}
```

```
  .edit-button:hover{
    background-color: #b1632f;

  }

.btn {
  font-size: 14px;
  border-radius: 5px;
}

/*Adds color to the body*/
  body{
    background-color: #f8f9fc;
  }

/*Adds colors to the links*/

.custom-active-link {
  --bs-navbar-active-color: rgba(0, 0, 0, 0) !important; /* Color
transparente */
  color: rgb(240, 234, 234) !important; /* Color de texto
transparente */
}
```

```
.is-active {

  color: #50dcff !important; /* Cambia este valor al color deseado
*/

}
```

Base de Datos

/*

Tabla: projects

Propósito: Esta tabla almacena información sobre los proyectos.

Campos:

- project_id: Un identificador único para cada proyecto. Es un primary Key.

- project_name: El nombre del proyecto. No puede ser nulo.

- description: Una descripción detallada del proyecto. No puede ser nulo.

- status: El estado actual del proyecto. No puede ser nulo.

*/

-- Table 1: projects

```
CREATE TABLE IF NOT EXISTS projects (
        project_id INT AUTO_INCREMENT PRIMARY KEY,
        project_name VARCHAR(255) NOT NULL,
        description TEXT NOT NULL,
        status VARCHAR(50) NOT NULL
);
/*
Tabla: team_members
Propósito: Esta tabla almacena información sobre los miembros pertenecientes a un proyecto.

Campos:
- member_id: Un identificador único para cada miembro. Es un primary Key.
- member_name: El nombre del miembro. No puede ser nulo.
- role: Rol designado al miembro. No puede ser nulo.
- proyect_id: El miembro referencia a un proyecto en específico. No puede ser nulo. Es un foreign key.
*/

-- Table 2: team_members
CREATE TABLE IF NOT EXISTS team_members (
        member_id INT AUTO_INCREMENT PRIMARY KEY,
        member_name VARCHAR(255) NOT NULL,
        role VARCHAR(100) NOT NULL,
```

```
    project_id INT NOT NULL,
    FOREIGN KEY (project_id) REFERENCES projects(project_id)
);

/*
```

Tabla: tasks

Propósito: Esta tabla almacena información sobre los tasks de los miembros.

Campos:

- task_id: Un identificador único para cada task. Es un primary Key.

- task_name: El nombre de cada task. No puede ser nulo.

- start_date: La fecha de inicio del task. No puede ser nulo.

- end_date: La fecha de finalización del task. No puede ser nulo.

- member_id: El task referencia a un miembro en específico. No puede ser nulo. Es un foreign key.

- proyect_id: El task referencia a un proyecto en específico. No puede ser nulo. Es un foreign key.

```
*/
-- Table 3: tasks
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    task_name VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    project_id INT NOT NULL,
    member_id INT NOT NULL,
```

```
    FOREIGN KEY (project_id) REFERENCES projects(project_id),
    FOREIGN KEY (member_id) REFERENCES team_members(member_id)
);
```

Backend

App2.py

app2.py actúa como el servidor principal para una aplicación de gestión de proyectos desarrollada en Flask. Integra módulos para tareas, miembros y proyectos, proporcionando una API RESTful que permite crear, actualizar, eliminar y recuperar información sobre tareas, miembros y proyectos. Utiliza CORS para permitir peticiones seguras desde otros dominios y se configura para escuchar en todas las interfaces de red. Cada ruta está claramente definida para manejar operaciones específicas, asegurando que los datos puedan ser gestionados eficientemente a través de llamadas a funciones importadas desde otros scripts, lo que facilita la interacción con la base de datos y mejora la modularidad del sistema.

```python
#Abielmelex#
# Agrega miembros a un proyecto específico.
@app.route('/api/add_members/<int:project_id>', methods=['POST'])
def add_member_route(project_id):
    return post_members(project_id)

# Obtiene todos los miembros asociados a un proyecto específico.
@app.route('/api/get_members_by_project/<int:project_id>', methods=['GET'])
def get_members_by_project_route(project_id):
    # Llama a la función que contiene la lógica de la base de datos
    return get_members_by_project(project_id)

#----------------------------** RUTAS DE PROYECTOS **-----------------------------------
# Ruta para crear un nuevo proyecto
@app.route('/api/new_project', methods=['POST'])
def new_project_route():
    return new_project()

# Elimina un proyecto específico según su ID.
@app.route('/api/delete_project/<int:project_id>', methods=['DELETE'])
def delete_project_route(project_id):
    return delete_project(project_id)
```

```python
from flask import Flask, jsonify, request
import mariadb
import sys
from flask_cors import CORS
from config import DATABASE_CONFIG
from tasks import create_task, delete_task, get_all_tasks, update_task
from members import get_all_members, delete_member, update_member, post_members, get_members_by_project
from projects import new_project, delete_project, update_project, get_all_projects

app = Flask(__name__)
CORS(app)

#----------------------------** RUTAS DE TAREAS **-----------------------------------

# Crea una nueva tarea asignada a un miembro en un proyecto específico.
@app.route('/api/new_task/<int:project_id>/<int:member_id>', methods=['POST'])
def new_task_route(project_id, member_id):
    return create_task(project_id, member_id)

# Elimina una tarea específica según su ID.
@app.route('/api/delete_task/<int:task_id>', methods=['DELETE'])
def delete_task_route(task_id):
    return delete_task(task_id)

# Obtiene todas las tareas disponibles.
@app.route('/api/get_all_tasks', methods=['GET'])
def get_all_tasks_route():
    return get_all_tasks()

# Actualiza la asignación de un miembro para una tarea específica.
@app.route('/api/update_task/<int:task_id>/<int:member_id>', methods=['PUT'])
def update_task_route(task_id, member_id):
    return update_task(task_id, member_id)



#----------------------------** RUTAS DE MIEMBROS **-----------------------------------

# Obtiene todos los miembros disponibles.
@app.route('/api/get_all_members', methods=['GET'])
def get_all_members_route():
    return get_all_members()

# Elimina un miembro específico según su ID.
@app.route('/api/delete_member/<int:member_id>', methods=['DELETE'])
def delete_member_route(member_id):
    return delete_member(member_id)

# Actualiza la información de un miembro en un proyecto específico.
@app.route('/api/update_member/<int:project_id>/<int:member_id>', methods=['PUT'])
def update_member_route(project_id, member_id):
    return update_member(project_id, member_id)
```

Config.py

config.py juega un papel fundamental en la estructura de tu aplicación Flask al almacenar las configuraciones centrales de la base de datos en DATABASE_CONFIG. Este archivo define parámetros clave como el host, puerto, usuario, contraseña, y nombre de la base de datos necesarios para establecer la conexión con MariaDB. Al separar esta configuración en un archivo independiente, config.py mejora la seguridad y la mantenibilidad del código, permitiendo ajustes rápidos a la configuración de la base de datos sin alterar el código principal de la aplicación.

```
DATABASE_CONFIG = {
'host': '172.16.5.133',
'port': 3306,
'user': 'ducketeers',
'password': 'best_group',
'database': 'project_management'
}
@app.route('/api/update_project/<int:project_id>', methods=['PUT'])
def update_project_route(project_id):
    return update_project(project_id)

# Obtiene todos los proyectos disponibles.
@app.route('/api/get_all_projects', methods=['GET'])
def get_all_projects_route():
    return get_all_projects()


#-----------------------------------** RUTAS END **---------------------------------------


# Inicia el servidor Flask si el script se ejecuta directamente
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Task.py

task.py define una serie de funciones API para gestionar tareas en un sistema de gestión de proyectos, utilizando datos JSON. Cada función primero verifica la existencia de las tareas o los detalles relevantes antes de realizar operaciones como agregar, actualizar o eliminar tareas. Además, se manejan errores específicos de la base de datos y se devuelven respuestas en formato JSON. Esto permite una integración eficiente con otras partes del sistema o interfaces de usuario que dependan de estos servicios para la gestión de tareas dentro de proyectos.

```python
from flask import jsonify, request
import mariadb
import sys
from config import DATABASE_CONFIG

def get_db_connection():
    try:
        conn = mariadb.connect(**DATABASE_CONFIG)
        return conn
    except mariadb.Error as e:
        print(f"Error connecting to MariaDB: {e}")
        sys.exit(1)

# Crea una nueva tarea
def create_task(project_id, member_id):
    conn = get_db_connection()
    try:
        data = request.json
        task_name = data.get('task_name')
        start_date = data.get('start_date')
        end_date = data.get('end_date')

        cursor = conn.cursor()
        cursor.execute("INSERT INTO tasks (task_name, start_date, end_date, member_id, project_id) VALUES (?, ?, ?, ?, ?)",
                       (task_name, start_date, end_date, member_id, project_id))
        conn.commit()
        return jsonify({"message": "New task created successfully"}), 201
    except mariadb.Error as e:
        conn.rollback()
        return jsonify({"message": str(e)}), 500
    finally:
        cursor.close()
        conn.close()

#-----------------------------------** FUNCION DELETE TASK **-------------------------------------
# Elimina una tarea existente
def delete_task(task_id):
    conn = get_db_connection()
    try:
        cursor = conn.cursor()

        # Verifica si la tarea existe
        cursor.execute("SELECT * FROM tasks WHERE task_id = %s", (task_id,))
        task = cursor.fetchone()

        if task is None:
            return jsonify({"error": "Task does not exist."}), 404

        # Elimina la tarea
        cursor.execute("DELETE FROM tasks WHERE task_id = %s", (task_id,))
```

```
        # Llenar los proyectos con las tareas existentes, incluyendo el nombre del miembro
        cursor.execute("""
            SELECT t.task_id, t.task_name, t.start_date, t.end_date, t.project_id, t.member_id, m.member_name
            FROM tasks t
            JOIN team_members m ON t.member_id = m.member_id
        """)
        tasks = cursor.fetchall()
        for task in tasks:
            task_info = {
                "task_id": task[0],
                "task_name": task[1],
                "start_date": task[2],
                "end_date": task[3],
                "member_id": task[5],
                "member_name": task[6]  # Añadido el nombre del miembro
            }
            project_id = task[4]
            if project_id in projects_with_tasks:
                projects_with_tasks[project_id]["tasks"].append(task_info)

        # Construir la respuesta JSON
        response_data = [
            {
        conn.commit()

        return jsonify({"message": "Task deleted successfully."}), 200
    except Exception as e:
        conn.rollback()
        return jsonify({"error": str(e)}), 500
    finally:
        cursor.close()
        conn.close()

#-----------------------------------** FUNCION GET ALL TASKS **-----------------------------------
# Obtiene todas las tareas
def get_all_tasks():
    conn = get_db_connection()
    try:
        projects_with_tasks = {}

        # Inicializar la lista de proyectos con sus datos
        cursor = conn.cursor()
        cursor.execute("SELECT project_id, project_name FROM projects")
        all_projects = cursor.fetchall()
        for project in all_projects:
            project_id, project_name = project
            projects_with_tasks[project_id] = {"project_name": project_name, "tasks": []}
```

```
                "project_id": project_id,
                "project_name": project_data["project_name"],
                "tasks": project_data["tasks"]
            } for project_id, project_data in projects_with_tasks.items()
        ]
        return jsonify({"data": response_data}), 200

    except Exception as e:
        print(e)
        return jsonify({"error": str(e)}), 500

    finally:
        cursor.close()
        conn.close()

#-----------------------------------** FUNCION UPDATE TASK **---------------------------------------
# Actualiza una tarea existente
def update_task(task_id, member_id):
    conn = get_db_connection()
    try:
        data = request.json
        task_name = data.get('task_name')
        start_date = data.get('start_date')
        end_date = data.get('end_date')

        cursor = conn.cursor()
        cursor.execute("UPDATE tasks SET task_name = %s, start_date = %s, end_date = %s, member_id = %s WHERE task_id = %s",
                       (task_name, start_date, end_date, member_id, task_id))
        conn.commit()

        return jsonify({"message": "Record updated"}), 200
    except Exception as e:
        conn.rollback()
        return jsonify({"error": str(e)}), 500
    finally:
        cursor.close()
        conn.close()
```

Projects.py

projects.py implementa APIs para administrar proyectos en un sistema de gestión. Incluye operaciones como añadir, modificar, y eliminar proyectos, y recuperar información sobre los mismos. Las funciones manejan errores de base de datos y proporcionan respuestas JSON, asegurando una interacción eficiente y robusta con interfaces de usuario o sistemas dependientes.

```python
from flask import jsonify, request
import mariadb
import sys
from config import DATABASE_CONFIG

def get_db_connection():
    try:
        conn = mariadb.connect(**DATABASE_CONFIG)
        return conn
    except mariadb.Error as e:
        print(f"Error connecting to MariaDB: {e}")
        sys.exit(1)
#-----------------------------------** FUNCION NEW PROJECT **-------------------------------------
def new_project():
    conn = get_db_connection()
    try:
        data = request.json
        project_name = data.get('project_name')
        description = data.get('description')
        status = data.get('status')

        cursor = conn.cursor()
        cursor.execute("INSERT INTO projects (project_name, description, status) VALUES (?, ?, ?)", (project_name, description, status))
        conn.commit()
        return jsonify({"message": "Project inserted successfully"}), 201

    except mariadb.Error as e:
        conn.rollback()
        return jsonify({"message": str(e)}), 500

    finally:
        cursor.close()
        conn.close()

#-----------------------------------** FUNCION DELETE PROJECT **---------------------------------------
def delete_project(project_id):
    conn = get_db_connection()
    try:
        cursor = conn.cursor()

        # Check if project exists
        cursor.execute("SELECT * FROM projects WHERE project_id = %s", (project_id,))
        project = cursor.fetchone()

        if project is None:
            return jsonify({"error": "Project does not exist."}), 404

        # Check if project is associated with any tasks
        cursor.execute("SELECT * FROM tasks WHERE project_id = %s", (project_id,))
        tasks = cursor.fetchall()

        cursor.execute("SELECT * FROM team_members WHERE project_id = %s", (project_id,))
        members = cursor.fetchall()

        if tasks:
            return jsonify({"error": "Cannot delete project as it is associated with tasks."}), 400
```

Members.py

Members.py define varias rutas API para agregar, obtener, actualizar y cambiar la asignación de proyectos de los miembros utilizando datos JSON. Cada ruta primero evalúa la existencia de los miembros o proyectos involucrados antes de realizar operaciones y maneja errores específicos de la base de datos, devolviendo respuestas JSON.

```
from flask import Flask, jsonify, request

import mariadb

import sys

from config_externo import DATABASE_CONFIG



app = Flask(__name__)









try:



   conn = mariadb.connect(**DATABASE_CONFIG)



except mariadb.Error as e:



   print(f"Error on connection: {e}")
```

```
    sys.exit(1)




cursor = conn.cursor()


#obtener miembros por proyectos
@app.route('/api/get_members', methods=['GET'])
def get_members():
    cursor.execute("SELECT member_id, member_name, role
FROM team_members")
    members = cursor.fetchall()
    return jsonify(members)


from flask import request, jsonify


#insertar datos del nuevo miembro en la base de datos
```

```
@app.route('/api/add_members/<int:project_id>/<int:member
_id>', methods=['POST'])

def post_members(project_id, member_id):

   datos = request.json

   role = datos.get('role')

   member_name = datos.get('member_name')



   try:

       cursor.execute("INSERT INTO team_members
(member_id, member_name, role, project_id) VALUES (?, ?,
?, ?)", (member_id, member_name, role, project_id))

       conn.commit()

   except mariadb.Error as e:

       print(f"Error de base de datos: {e}")

       return jsonify({"error": "Error al procesar la
solicitud"}), 500



   return jsonify({"message": "Miembro añadido
correctamente", "member_id": member_id, "role": role}),
201
```

```
# Obtener todos los miembros del equipo
@app.route('/api/get_all_members', methods=['GET'])
def members():


   with conn.cursor() as cursor:
       cursor.execute("SELECT member_id, member_name,
role, project_id FROM team_members")
       result = cursor.fetchall()


   members = [{'member_id': row[0], 'member_name':
row[1], 'role': row[2], 'project_id': row[3]} for row in
result]


   response = jsonify({'members': members})
   response.headers.add("Access-Control-Allow-Origin",
'*')
   return response


#actualizar los dato del miembro en la base de datos
```

```
@app.route('/api/update_member/<int:project_id>/<int:memb
er_id>', methods=['POST'])
def update_member(project_id, member_id):
    datos = request.json
    member_name = datos.get('member_name')
    role = datos.get('role')

    try:
        with conn.cursor() as cursor:
            # Verificar si el proyecto existe
            cursor.execute("SELECT project_id FROM
projects WHERE project_id = ?", (project_id,))
            project = cursor.fetchone()
            if not project:
                return jsonify({"error": "Proyecto no
encontrado"}), 404

            # Verificar si el miembro existe
```

```
            cursor.execute("SELECT member_id FROM
team_members WHERE member_id = ? AND project_id = ?",
(member_id, project_id))

            member = cursor.fetchone()

            if not member:

                return jsonify({"error": "Miembro no
encontrado en el proyecto especificado"}), 404



            # Realizar la actualización

            cursor.execute("UPDATE team_members SET
member_name = ?, role = ? WHERE member_id = ? AND
project_id = ?",

                           (member_name, role, member_id,
project_id))

            if cursor.rowcount == 0:

                return jsonify({"error": "No se actualizó
ningún miembro, verifique los identificadores"}), 404

            conn.commit()

    except mariadb.Error as e:

        conn.rollback()

        print(f"Error updating member: {e}")
```

```
        return jsonify({"error": "Error al actualizar
miembro", "details": str(e)}), 500


    return jsonify({"message": "Miembro actualizado
correctamente"}), 200


#cambiar un miembro de un proyecto a otro(project_id)
@app.route('/api/change_member_project/<int:member_id>/<i
nt:new_project_id>', methods=['POST'])
def change_member_project(member_id, new_project_id):
    try:
        with conn.cursor() as cursor:
            # Verificar si el nuevo proyecto existe
            cursor.execute("SELECT project_id FROM
projects WHERE project_id = ?", (new_project_id,))
            if not cursor.fetchone():
                return jsonify({"error": "Proyecto no
encontrado"}), 404


            # Verificar si el miembro existe
```

```
            cursor.execute("SELECT member_id FROM
team_members WHERE member_id = ?", (member_id,))

            if not cursor.fetchone():

                return jsonify({"error": "Miembro no
encontrado"}), 404



            # Actualizar el proyecto del miembro

            cursor.execute("UPDATE team_members SET
project_id = ? WHERE member_id = ?", (new_project_id,
member_id))

            conn.commit()



    except mariadb.Error as e:

        conn.rollback()

        print(f"Error updating member's project: {e}")

        return jsonify({"error": "Error al actualizar el
proyecto del miembro", "details": str(e)}), 500



    return jsonify({"message": "Proyecto del miembro
actualizado correctamente"}), 200
```

```
# Definición de la ruta para obtener miembros por
project_id
@app.route('/api/get_members_by_project/<int:project_id>'
, methods=['GET'])
def get_members_by_project_route(project_id):
    # Llama a la función que contiene la lógica de la base
de datos
    return get_members_by_project(project_id)


# Lógica para obtener los miembros por project_id
def get_members_by_project(project_id):
    try:
        with conn.cursor() as cursor:
            # Ejecutar la consulta SQL para obtener
miembros que pertenecen en algún proyecto en específico
            cursor.execute("SELECT member_id, member_name
FROM team_members WHERE project_id = ?", (project_id,))
            members = cursor.fetchall()
            # Preparar los datos para la respuesta
```

```
            member_data = [{'member_id': member[0],
'member_name': member[1]} for member in members]

            return jsonify(member_data)

    except mariadb.Error as e:

        # Manejar errores(si hay) de la consulta

        print(f"Database query error: {e}")

        return jsonify({"error": "Error retrieving
members"}), 500



# Configuración de la conexión a la base de datos

try:

    conn = mariadb.connect(user='tu_usuario',
password='tu_contraseña', host='tu_host', port=3306,
database='tu_base_de_datos')

except mariadb.Error as e:

    print(f"Error al conectar a MariaDB: {e}")

    sys.exit(1)



app = Flask(__name__)
```

```
# Ruta para actualizar los datos de un miembro específico
@app.route('/api/update_member/<int:member_id>',
methods=['PUT'])
def update_member_route(member_id):
    return update_member(member_id)


def update_member(member_id):
    try:
        cursor = conn.cursor()
        # Comprobar si el miembro existe
        cursor.execute("SELECT * FROM team_members WHERE
member_id = %s", (member_id,))
        member = cursor.fetchone()

        if member is None:
            return jsonify({"error": "Member does not
exist."}), 404

        # Obtener datos actualizados de la solicitud
        data = request.json
```

```
        name = data.get('name')
        email = data.get('email')

        # Actualizar los datos del miembro
        cursor.execute("UPDATE team_members SET name = %s,
email = %s WHERE member_id = %s", (name, email,
member_id))
        conn.commit()

        return jsonify({"message": "Member updated
successfully."}), 200
    except Exception as e:
        conn.rollback()
        return jsonify({"error": str(e)}), 500
    finally:
        cursor.close()

if __name__ == '__main__':

    app.run(host='0.0.0.0', port=5000)
```

Gunicorn and supervisor

Gunicorn

Gunicorn se utiliza para mantener el servidor web siempre en funcionamiento, garantizando que esté listo para manejar solicitudes entrantes de manera constante. Esto ayuda a asegurar que la aplicación web esté disponible y accesible para los usuarios en todo momento.

```
GNU nano 6.2                                        /etc/nginx/sites-available/default

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                proxy_pass http://172.16.5.78:5000;  # Cambia la IP y puerto según sea necesario
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto $scheme;
        }
```

Supervisor

Este bloque de configuración define las instrucciones para Supervisor, una herramienta de control de procesos, sobre cómo manejar y monitorear un programa específico en un servidor. Se establece el nombre del programa, el comando para iniciar la aplicación, el directorio de trabajo, el usuario bajo el cual se ejecutará, y la configuración para iniciar automáticamente y reiniciar en caso de fallos. También se especifica la ubicación de los archivos de registro para errores estándar y salida estándar del programa. En resumen, este bloque configura Supervisor para asegurar que la aplicación se ejecute de manera confiable y se gestione adecuadamente en el servidor.

```
GNU nano 6.2                                /etc/supervisor/conf.d/tu_app.conf
[program:tu_app]
command=/home/rvalentin8653/Documents/project_management_backend2/project_management_backend/venv/bin/gunicorn -b 0.0.0.0:5000 app2:a>
directory=/home/rvalentin8653/Documents/project_management_backend2/project_management_backend
user=root
autostart=true
autorestart=true
stderr_logfile=/var/log/tu_app/tu_app.err.log
stdout_logfile=/var/log/tu_app/tu_app.out.log
```