Universidad Don Bosco

Primer proyecto en React

Materia: Diseño y Programación de Software Multiplataforma.

Profesor: Alexander Alberto Siguenza Campos .

Integrantes:

David Eduardo Rodríguez Vigil RV202840

Fecha de entrega:06 de octubre de 2024.

Parte del backend

Api/login/route.js

```
import bcrypt from "bcryptjs";
import { PrismaClient } from "@prisma/client";
import { generateToken } from "@/lib/auth";
import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function POST(request) {
    const body = await request.json();
    const { email, password } = body;

    if(!email || !password){
        return NextResponse.json({
            error: "Email y contraseña son requeridos"
        }, {status: 400});       You, 15 minutes ago • first commit
    }
    const userFind = await prisma.user.findUnique({
        where: {
            email: email
        }
    });
    if(!userFind){
        return NextResponse.json({error: "Usuario no encontrado"}, {status: 404});
    }
    const isValidPassword = await bcrypt.compare(password, userFind.password);
    if(!isValidPassword){
        return NextResponse.json({error: "Contraseña incorrecta"}, {status: 401});
    }
    const token = generateToken(userFind.id);
    return NextResponse.json({
```

En este apartado realizamos lo que es el api rest del login del usuario donde hacemos la conexión a la base de datos y realizamos las validaciones necesarias para poder realizare inicio de sesión en la página web.

Api/register/route.js

```
You, 16 minutes ago | 1 author (You)
import bcrypt from "bcryptjs";
import { PrismaClient } from "@prisma/client";
import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function POST(request) {
    try {
        const body = await request.json();
        const { email, password, username, rol } = body;
        if (!email || !password || !username || !rol){
            return NextResponse.json({
                message: "Falta un elemento requerido",
            },{
                status: 400
            });
        }
        const userFind = await prisma.user.findUnique({
            where: {
                email: email,
                username: username
            }
        });
        if (userFind) {
            return NextResponse.json({
                message: "El usuario ya existe",
                },{
                    status: 400
            });
```

En este apartado realizamos la funcionabilidad del api rest de registro de los usuarios donde hacemos las validaciones necesarias para poder registrar el usuario y poder ingresarlo a la base de datos y almacenarla ahí.

Lib/auth.js

```
import jwt from 'jsonwebtoken';

export const generateToken = (user) => {
    const token = jwt.sign({ userId: user.id }, process.env.SECRET_KEY, { expiresIn: '1h'});
};

export const verifyToken = (req) => {
    const token = req.headers.authorization?.split(' ')[1];
    if(!token){
        throw new Error('No Autorizado');
    }
    return jwt.verify(token, process.env.SECRET_KEY);
};
```

En este apartado realizamos la creación del token del middleware para la protección de los datos del usuario como tambien la verificación del token al momento de extraerlo en los demás enlaces.

Api/projects/route.js

```
import { PrismaClient } from "@prisma/client";
import { verifyToken } from "@/lib/auth";
import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function GET(request) {
    try {
        const user = verifyToken(request);
        const projects = await prisma.project.findMany({
            where: {
                userId: user.userId,
            },
            include: {
                tasks: true
            },
        });
        return NextResponse.json({projects});
    } catch (error) {
        return NextResponse.json({message: 'No Autorizado'},{ status: 401 });
    }
}
export async function POST(request) {
    const { name, description, complete } = request.body;
    try {
        const user = verifyToken(request);
        const newProject = await prisma.project.create({
            data: {
                name,
                description,
```

En este apartado realizamos la api rest de proyecto en la cual extraemos los datos de la tabla como también la creación de los proyectos e ingresarlos en la base de datos para el almacenaje de estos proyectos que estamos creando.

Api/projects/[id].js

```
import { PrismaClient } from "@prisma/client";
import { verifyToken } from "@/lib/auth";
import { NextResponse } from "next/server";

const prisma = new PrismaClient;

export async function GET(request) {
    const {id} = request.query;
    try {
        const user = verifyToken(request);
        const project = await prisma.project.findFirst({
            where: {
                id: Number(id),
                userId: user.userId
            }, include:{
                tasks: true
            }
        });
        if(!project){
            return NextResponse.json({ message: 'Proyecto no encontrado'});
        }
        return NextResponse.json({project},{status:200});
    } catch (error) {
        return NextResponse.json({ message: ' No autorizado'});
    }
}
export async function PUT(request) {
    const { name } = request.body;
    try {
        const user = verifyToken(request);
```

En este apartado realizamos el CRUD de los proyectos por medio del ID del proyecto que estemos necesitando realizar los cambios necesarios que se requerían además de la validación necesaria que el CRUD para realizar la funcionabilidad necesaria.

/api/Task/route.js

```
import { PrismaClient } from "@prisma/client";
import { verifyToken } from "@/lib/auth";
import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function GET(request) {
    const user = verifyToken(request);      You, 31 minutes ago • first commit
    try {
        const tasks = await prisma.task.findMany({
            where:{
                projectId: Number(request.query.projectId)
            }
        });
        return NextResponse.json(tasks);
    } catch (error) {
        return NextResponse.json({message: ' Error de Servidor'});
    }
}
export async function POST(request) {
    const { name, description, projectId} = request.body;

    try {
        const user =verifyToken(request);
        const task = await prisma.task.create({
            data: {
                name,
                description,
                projectId: Number(projectId),
            },
```

En este apartado se realizan las acciones de la api rest de las tareas que se asignaran a los proyectos, ademas de realizar la validación necesarias para la información tambien realizamos la conexión de la base de datos.

Api/Task/[id].js

```
import { PrismaClient } from "@prisma/client";
import { verifyToken  } from "@/lib/auth";
import { NextResponse } from "next/server";

const prisma = new PrismaClient();

export async function GET(request) {
    const {id} = request.query;
    try {
        const user = verifyToken(request);
        const task = await prisma.task.findUnique({
            where: {
                id: id
            },
        });
        if(!task){
            return NextResponse.json({message: 'Tarea no Encontrada'},{status: 404});
        }
        return NextResponse.json(task);
    } catch (error) {
        return NextResponse.json({message: 'Error encontrar la tarea'},{status:500});
    }
}
export async function PUT(request) {
    const {id} = request.query;
    const { name, description } = request.body;
    try {
        const user = verifyToken(request);
        const task = await prisma.task.findUnique({
```

En este apartado realizamos el CRUD de las tareas por medio del ID del proyecto que estemos necesitando realizar los cambios necesarios que se requerían además de la validación necesaria que el CRUD para realizar la funcionabilidad necesaria y la conexión de la base de datos para guardar los datos.

Parte del front-end

/login/pages.jsx

```jsx
"use client";
import { useState, useContext } from "react";
import axios from "axios";
import { useRouter } from "next/navigation";
import { AuthContext } from "../AuthContext";

const Login = () => {
    const  router  = useRouter();
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [error, setError] = useState(null);
    const { login } = useContext(AuthContext);

    const handleLogin = async (e) => {
        e.preventDefault();
        try {
            const response = await axios.post('/api/auth/login', {
                email,
                password
            })
            const { token } = response.data;
            login(token);
            router.push('/Projects');       You, 43 minutes ago • first commit
        } catch (error) {
            if (error.response) {
                console.error('Error en la respuesta del servidor:', error.response.data);
                setError(error.response.data.message || 'Error en el servidor');
            } else if (error.request) {
                console.error('No se recibió respuesta del servidor:', error.request);
                setError('No se recibió respuesta del servidor');
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para el inicio de sesión del usuario ya asi por entrar a la aplicación como la creación del token para la seguridad de la información.

/register/page.jsx

```jsx
"use client";
import { useState } from "react";
import axios from "axios";
import { useRouter } from "next/navigation";

const Register = () => {
    const route = useRouter();
    const [username, setUsername] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [rol, setRol] = useState("");
    const [error, setError] = useState("");

    const handleRegister = async (e) => {
        e.preventDefault();
        try {
            const response = await axios.post('/api/auth/register',{
                username,
                email,
                rol,
                password
            });
            console.log(response.data);
            route.push('/login');      You, 45 minutes ago • first commit
        } catch (error) {
            console.error('Error en el servidor');
        }
    };
    return (
        <div>
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para el registro del usuario.

/authcontext.js

```
"use client";
import React, { createContext, useState, useEffect } from 'react';

// Crear el contexto de autenticación
export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [token, setAuthToken] = useState(null);

  useEffect(() => {
    // Revisar si el token JWT está en el localStorage al cargar la página
    const savetoken = localStorage.getItem('token');
    if (savetoken) {
      setAuthToken(savetoken);
      // Opcional: obtener datos de usuario desde la API usando el token
    }
  }, []);

  const login = (token) => {
    setAuthToken(token);
    localStorage.setItem('token', token);
  };

  const logout = () => {
    setAuthToken(null);
    localStorage.removeItem('token');
  };

  return (
    <AuthContext.Provider value={{ token, login, logout }}>
```

En este apartado creamos la funcionalidad para el almacenaje del token de sesión que se ocupara para cargar la página y la función necesaria para realizar el login y el logout y guardar los datos del token.

/Proyects/pages.jsx

```jsx
'use client';
import { useEffect, useState, useContext } from "react";
import { AuthContext } from "../AuthContext";
import axios from "axios";
import { useRouter } from "next/navigation";

const Projects = () => {
    const {token} = useContext(AuthContext);
    const [projects, setProjects] = useState([]);
    const [ error , setError] = useState(null);
    const router = useRouter();

    useEffect(()=>{
        console.log("token", token);
        if(!token){
            router.push('/login');
            return;
        }       You, 55 minutes ago • first commit
        const fetchProjects = async () => {
            try {
                const response = await axios.get('/api/project', {
                    headers:{
                        Authorization: `Bearer ${token}`
                    }
                })
                setProjects(response.data);
            } catch (error) {
                setError(error.message);
            }
        };
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para para el listado de los proyectos que se presentaran en el listado a los usuarios.

/CreateProjects/page.jsx

```
"use client";      You, 55 minutes ago • first commit
import { useState, useContext } from "react/cjs/react.production.min";
import { AuthContext } from "../AuthContext";
import axios from 'axios';

const CreateProject = () => {
    const [name, setName ] = useState('');
    const [description, setDescription] = useState('');
    const [complete, setComplete] = useState(null);
    const [error, setError] = useState(null);
    const { authToken } = useContext(AuthContext);

    const handleCreateProject  = async (e) => {
        e.preventDefault();
        try {
            const response = await axios.post('/api/project',
                {
                    name,
                    description,
                    complete
                },
                {headers:
                    {
                        'Authorization': `Bearer ${authToken}`
                    }
                });
            const data = await response.json();
            if(response.ok){
                console.log(data);
            }else{
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para la Creacion de los proyectos.

```
"use client";       You, 1 hour ago • first commit
import { useEffect, useState, useContext } from "react";
import axios from "axios";
import { AuthContext } from "./AuthContext";

const CreateTask = ({ projectId }) => {
    const [name, setName ] = useState("");
    const [description, setDescription] = useState("");
    const [error, setError] = useState(null);
    const {authToken} = useContext(AuthContext);

    const handleCreateTask = async (e) => {
        e.preventDefault();
        try {
            const response = await axios.post(`/api/projects/${projectId}/tasks`,{
                name,
                description
                },
                {
                headers: {
                    Authorization: `Bearer ${authToken}`
                }
            });
            console.log(response.data);
        } catch (error) {
            setError(error.message);
        }
    };
    if(!authToken){
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para el listado de las diferentes tareas de los proyectos.

```
"use client";
import { useContext, useState } from "react";       You, 1 minute ago • Uncommitted char
import axios from "axios";
import { AuthContext } from "../AuthContext";

const DeleteTask = ({ projectId, taskId}) => {
    const { token } = useContext(AuthContext);
    const [error , setError] = useState(null);

    const handleDeleteTask = async () => {
        try {
            const response = await axios.delete(`/projects/${projectId}/tasks/${
                taskId
                }`, {
                    headers: {
                        Authorization: `Bearer ${token}`
                    }
                });
                console.log(response.data);
        } catch (error) {
            setError(error.message);
        }
    };
    if(!token){
        return <p>No tienes autorizacion</p>
    }
    return (
        <div>
          <button onClick={handleDeleteTask}>Eliminar Tarea</button>
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para la eliminación de las tareas.

```
"use client";
import { useState, useContext } from "react";       You, 1 minute ago • Unco
import axios from "axios";
import { AuthContext } from "./AuthContext";

const UpdateTask = ({projectId, TaskId}) => {
    const [task, setTask] = useState({name: '', description: ''});
    const { token } = useContext(AuthContext);
    const [error, setError] = useState(null);

    const handleUpdateTask = async (e) => {
        e.preventDefault();
        try {
            const response = await axios.put(`/api/tasks/${TaskId}`,
                {name: task.name, description: task.description},
                {headers: {'Authorization': `Bearer ${token}`}
            });
            console.log(response.data);
        } catch (error) {
            setError(error.message);
        }
    };
    if(!token){
        return <Redirect to="/login"/>
    }
    return(
        <div>
        <h1>Actualizar Tarea</h1>
        {error && <p>{error}</p>}
        <form onSubmit={handleUpdateTask}>
```

En este apartado realizamos la parte de la vista del front-end y las funciones necesarias con axios para la conectividad de la base de datos y realizar las peticiones necesarias para la actualización de las tareas de los proyectos.