Trabajo de investigación

Sistema de Gestión de Proyectos

- Diseño y Programación de Software Multiplataforma DPS941 G01T -

Docente:

Ing. Alexander Alberto Siguenza Campos

Integrantes:

- William Alberto García Gómez - GG212522
- Christian Yahir López Hernández - LH212531
- Marco Rodrigo Funes Bonilla - FB200456
- Edwin Walberto Palacios Mejía - PM140373
- Alem Isai Vasquez Antillon - VA223253

Fecha:

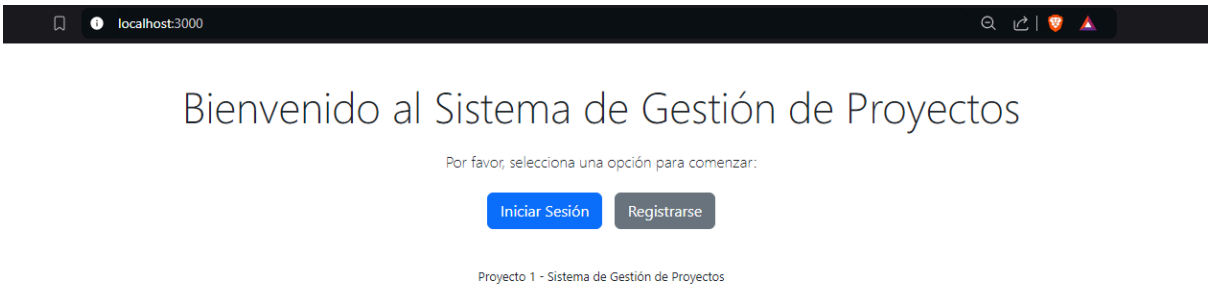
06 de octubre de 2024

Índice

Estructura del proyecto

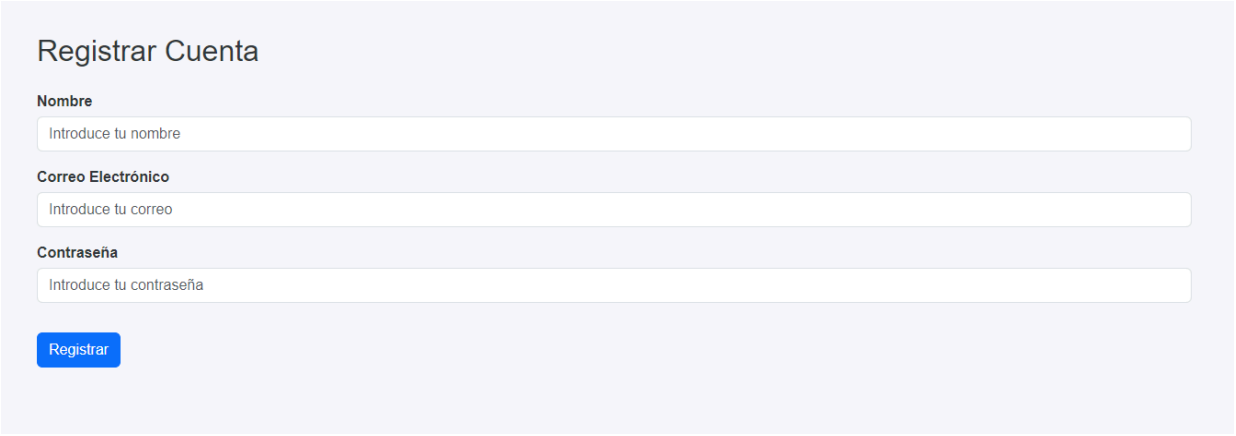
Visión general de las páginas

Main screen (page.js)



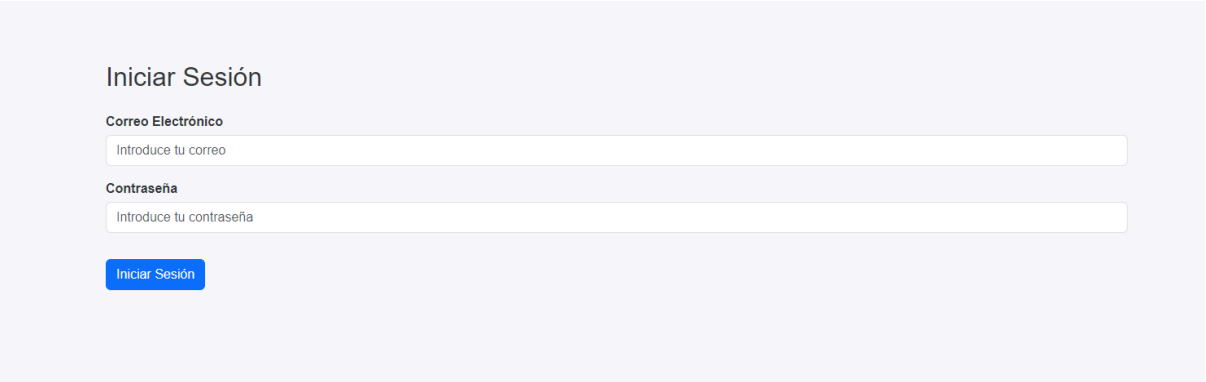
Pantalla principal que se muestra al iniciar al proyecto, y tenemos la opción de registrarnos o poder iniciar sesión en el sitio.

Registro (register.js)
Apartado para poder registrarse en el sistema, se solicita un nombre de usuario, correo electrónico y contraseña para poder registrar los datos.

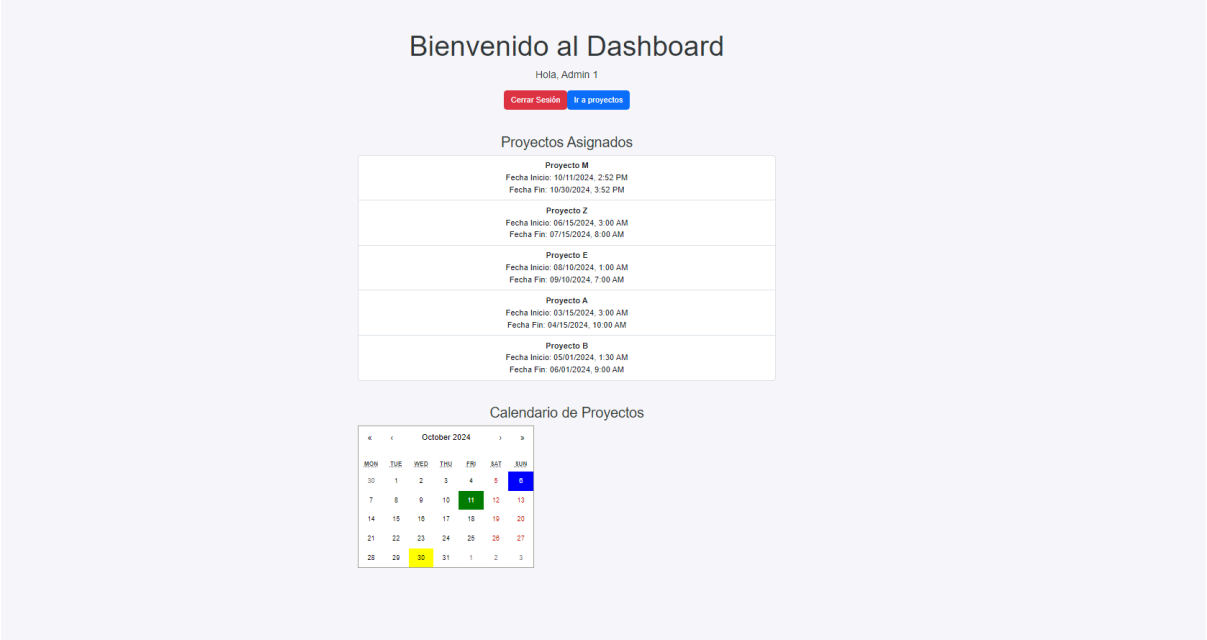


Inicio de sesión (login.js)

Una vez registrado, se redirecciona al usuario al login para poder iniciar sesión en el sistema con el respectivo correo electrónico, y contraseña.



Dashboard (dashboard.js)

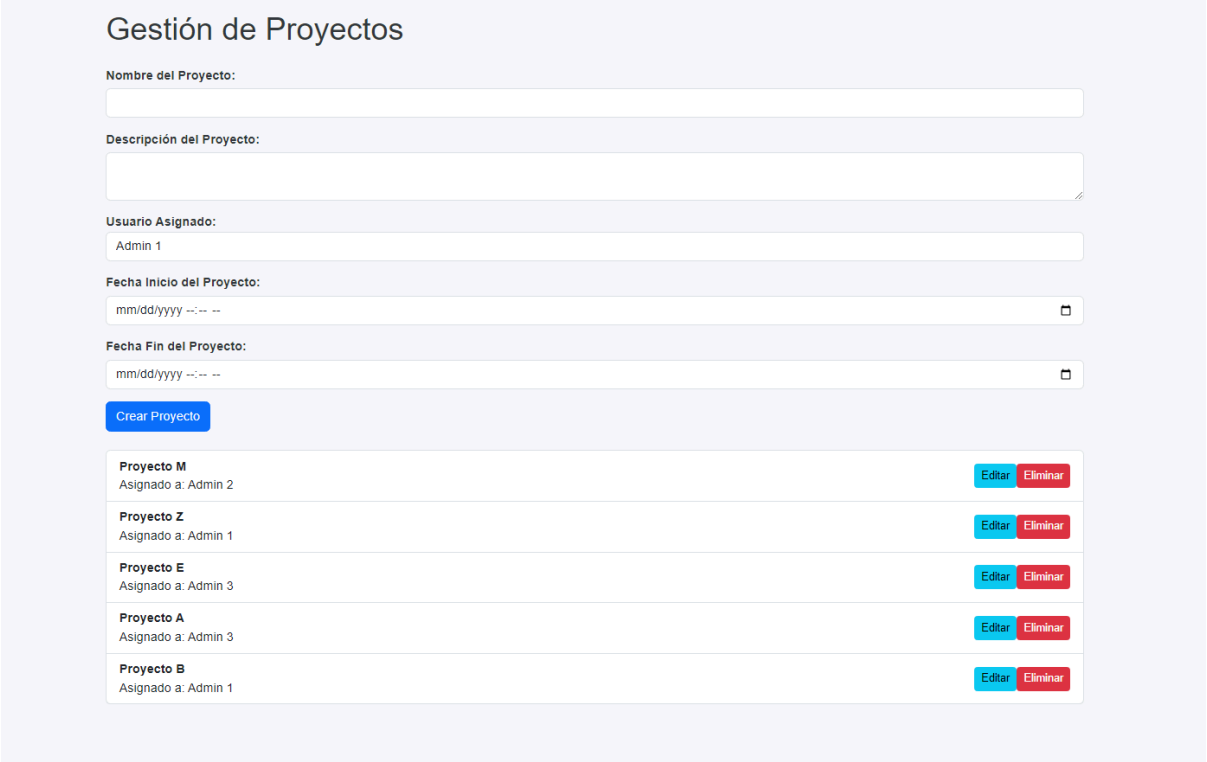


Al iniciar sesión, los usuarios son dirigidos al dashboard, que es la pantalla principal del sistema. En esta pestaña, damos la bienvenida al usuario logueado y le proporcionamos información clave sobre los proyectos que tiene asignados. En la parte superior, tenemos dos botones: uno para cerrar la sesión y otro para ir a la página de proyectos, donde se pueden gestionar más en detalle los proyectos activos. En el centro del Dashboard, se encuentra una lista de proyectos asignados, donde se muestra el nombre de cada proyecto, la fecha de inicio y la fecha de fin. Esto le da al usuario una visión general de los proyectos en los que está trabajando.

Además, en la parte inferior, hemos implementado un calendario interactivo que resalta las fechas de inicio y finalización de los proyectos. Esto permite al usuario ver de manera clara cuándo comienzan o terminan los proyectos asignados. El calendario también está

programado para resaltar el día actual, ayudando al usuario a mantenerse al tanto de sus tareas en tiempo real.

Gestión de Proyectos (proyectos.js)



Ahora, si el usuario desea gestionar un proyecto más a fondo, puede dirigirse a la Pantalla de Gestión de Proyectos. En esta pestaña, el usuario podrá ver un formulario de entrada en la parte superior, donde es posible crear un nuevo proyecto o actualizar uno existente. Los campos incluyen el nombre del proyecto, una descripción, el usuario al que se le asigna el proyecto, y las fechas de inicio y fin. Al lado de cada proyecto existente hay botones para crear o eliminar el proyecto según las acciones que queramos realizar. Debajo del formulario, hay una lista de proyectos ya creados, mostrando el nombre y el usuario asignado a cada uno. Dando así la visibilidad y el control que se espera en una aplicación de gestión de proyectos.

Componentes y Código

RegisterScreen.js
Este componente permite que los nuevos usuarios se registren en la aplicación creando una cuenta con su correo, nombre y contraseña.

```
import React, { useState } from 'react';
import api from './api'; // Importa la configuración de Axios
import { useRouter } from 'next/router';
import 'bootstrap/dist/css/bootstrap.min.css'; // Asegúrate de importar Bootstrap

const RegisterScreen = () => {
  const [userEmail, setUserEmail] = useState('');
  const [userPassword, setUserPassword] = useState('');
  const [userName, setUserName] = useState(''); // Nuevo campo para el nombre
  const [error, setError] = useState('');
  const router = useRouter();

  const handleRegister = async (event) => {
    event.preventDefault();
    try {
      const response = await api.post('/register', {
        email: userEmail,
        password: userPassword,
        name: userName,
        role: 'miembro', // Agregamos el rol 'miembro' al enviar los datos
      });

      console.log(response.data); // Muestra la respuesta de la API
      alert('Registro exitoso');
      router.push('/login'); // Redirige al usuario a la página de inicio de sesión
    } catch (error) {
      console.error('Hubo un problema con el registro.', error);
      if (error.response && error.response.data && error.response.data.error) {
        setError(error.response.data.error);
      } else {
        setError('Hubo un problema con el registro, intente de nuevo.');
      }
    }
  };
```

userEmail, **userPassword**, y **userName**: Almacenan los valores que ingresa el usuario al registrarse. **handleRegister** se ejecuta al enviar el formulario de registro y realiza una solicitud POST a la ruta /register en el backend, enviando los datos del usuario, incluyendo un rol predeterminado. Si el registro es exitoso, se redirige al usuario a la pantalla de inicio de sesión. En caso de error, actualiza el estado error para mostrar un mensaje de alerta.

```jsx
  return (
    <div className="container mt-5">
      <h2 className="mb-4">Registrar Cuenta</h2>
      {error && <div className="alert alert-danger">{error}</div>}
      <form onSubmit={handleRegister}>
        <div className="form-group">
          <label htmlFor="name">Nombre</label>
          <input
            type="text"
            className="form-control"
            id="name"
            placeholder="Introduce tu nombre"
            value={userName}
            onChange={(e) => setUserName(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="email">Correo Electrónico</label>
          <input
            type="email"
            className="form-control"
            id="email"
            placeholder="Introduce tu correo"
            value={userEmail}
            onChange={(e) => setUserEmail(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="password">Contraseña</label>
          <input
            type="password"
            className="form-control"
            id="password"
            placeholder="Introduce tu contraseña"
            value={userPassword}
            onChange={(e) => setUserPassword(e.target.value)}
            required
          />
        </div>
        <button type="submit" className="btn btn-primary mt-3">
          Registrar
        </button>
      </form>
    </div>
  );
};

export default RegisterScreen;
```

LoginScreen.js

```jsx
import React, { useState } from 'react';
import api from './api'; // Importa la configuración de Axios
import { useRouter } from 'next/router';
import 'bootstrap/dist/css/bootstrap.min.css'; // Asegúrate de importar Bootstrap

const LoginScreen = () => {
  const [userEmail, setUserEmail] = useState('');
  const [userPassword, setUserPassword] = useState('');
  const [error, setError] = useState('');
  const router = useRouter();

  const handleLogin = async (event) => {
    event.preventDefault();
    try {
      const response = await api.post('/login', {
        email: userEmail,
        password: userPassword,
      });

      if (response.data.message === 'Inicio de sesión exitoso') {
        // Re-verificar la sesión para asegurar que el usuario está logueado
        const sessionCheck = await api.get('/session');
        console.log('Resultado de la verificación de sesión:', sessionCheck.data);

        if (sessionCheck.data.loggedIn) {
          // Si la sesión es válida, redirigir al dashboard
          alert('Inicio de sesión exitoso');
          router.push('/dashboard');
        } else {
          setError('Error al iniciar sesión. Intente de nuevo.');
        }
      } else {
        setError('Error al iniciar sesión.');
      }
    } catch (error) {
      console.error('Hubo un problema con el inicio de sesión.', error.response?.data);
      setError(error.response?.data?.error || 'Error desconocido al iniciar sesión');
    }
  };
```

Este componente permite a los usuarios iniciar sesión en la aplicación con su correo electrónico y contraseña.

userEmail y userPassword: almacenan los valores de correo y contraseña que ingresa el usuario.
error: muestra mensajes de error si la autenticación falla.
Función handleLogin: esta función se ejecuta cuando el usuario envía el formulario.Realiza una solicitud POST a la ruta /login en el backend, enviando el correo electrónico y la contraseña. Si el backend devuelve una respuesta de éxito, hace una verificación de sesión

mediante un GET a /session para asegurarse de que el usuario esté autenticado. En caso de que la sesión sea válida, redirige al usuario al dashboard. Si no, muestra un mensaje de error.

```
  return (
    <div className="container mt-5">
      <h2 className="mb-4">Iniciar Sesión</h2>
      {error && <div className="alert alert-danger">{error}</div>}
      <form onSubmit={handleLogin}>
        <div className="form-group">
          <label htmlFor="email">Correo Electrónico</label>
          <input
            type="email"
            className="form-control"
            id="email"
            placeholder="Introduce tu correo"
            value={userEmail}
            onChange={(e) => setUserEmail(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="password">Contraseña</label>
          <input
            type="password"
            className="form-control"
            id="password"
            placeholder="Introduce tu contraseña"
            value={userPassword}
            onChange={(e) => setUserPassword(e.target.value)}
            required
          />
        </div>
        <button type="submit" className="btn btn-primary mt-3">
          Iniciar Sesión
        </button>
      </form>
    </div>
  );
};

export default LoginScreen;
```

DashboardScreen.js

Dashboard se muestra una vista general de la aplicación después de que el usuario ha iniciado sesión. Aquí, se muestran proyectos asignados, estadísticas, y se proporciona acceso a diferentes funcionalidades.

```
import React, { useEffect, useState } from 'react';
import { checkSession, logout, getProyectos } from './api';
import 'bootstrap/dist/css/bootstrap.min.css';
import { useRouter } from 'next/router';
import Calendar from 'react-calendar';
import 'react-calendar/dist/Calendar.css';
import { isSameDay, parseISO, format } from 'date-fns';

const DashboardScreen = () => {
  const [loggedIn, setLoggedIn] = useState(false);
  const [userData, setUserData] = useState(null);
  const [proyectos, setProyectos] = useState([]);
  const [selectedDate, setSelectedDate] = useState(new Date());
  const [error, setError] = useState(null);
  const router = useRouter();

  useEffect(() => {
    const verifySession = async () => {
      try {
        const sessionData = await checkSession();
        console.log('Datos de sesión:', sessionData);

        if (sessionData.loggedIn) {
          setLoggedIn(true);
          setUserData(sessionData.user);

          const proyectosAsignados = await getProyectos();
          console.log('Proyectos recibidos:', proyectosAsignados);

          if (proyectosAsignados && proyectosAsignados.length > 0) {
            setProyectos(proyectosAsignados);
            console.log('Proyectos almacenados:', proyectosAsignados);
          } else {
            console.log('No hay proyectos asignados.');
          }
        } else {
          setLoggedIn(false);
          router.push('/login');
        }
      } catch (error) {
        setError('Error al verificar la sesión o cargar los proyectos.');
        console.error('Error al verificar la sesión:', error);
      }
    };

    verifySession();
  }, [router]);
```

Si el usuario no tiene una sesión válida, se redirige a la página de inicio de sesión (/login). Al estar autenticado, se cargan los datos del usuario y los proyectos asignados.

Cuando el usuario está autenticado y tiene proyectos asignados, estos se muestran en una lista, y cada proyecto muestra su nombre, fecha de inicio y fecha de fin. Además, se utiliza la biblioteca react-calendar para presentar un calendario mensual.

```
const handleLogout = async () => {
  try {
    await logout();
    router.push('/login');
  } catch (error) {
    setError('Error al cerrar sesión.');
    console.error('Error al cerrar sesión:', error);
  }
};

// Función para determinar qué días se deben resaltar en el calendario
const tileClassName = ({ date, view }) => {
  if (view === 'month') {
    const isToday = isSameDay(date, new Date());

    // Verificar si la fecha es una fecha de inicio de algún proyecto
    const isStartDate = proyectos.some((p) => {
      const startDate = parseISO(p.fecha_inicio);
      return isSameDay(startDate, date);
    });

    // Verificar si la fecha es una fecha de fin de algún proyecto
    const isEndDate = proyectos.some((p) => {
      const endDate = parseISO(p.fecha_fin);
      return isSameDay(endDate, date);
    });

    let classNames = [];
    if (isToday) {
      classNames.push('today');
    }
    if (isStartDate) {
      classNames.push('start-date');
    }
    if (isEndDate) {
      classNames.push('end-date');
    }

    return classNames.length > 0 ? classNames.join(' ') : null;
  }
  return null;
};
```

```jsx
return (
  <div className="container mt-5">
    <div className="row justify-content-center">
      <div className="col-md-8 text-center">
        {error && <div className="alert alert-danger">{error}</div>}
        {loggedIn ? (
          <>
            <h1 className="display-4">Bienvenido al Dashboard</h1>
            {userData && <p className="lead">Hola, {userData.name}</p>}

            {/* Botones de Cerrar Sesión e Ir a Proyectos */}
            <div className="mt-3">
              <button className="btn btn-danger" onClick={handleLogout}>
                Cerrar Sesión
              </button>
              <button
                className="btn btn-primary ml-2"
                onClick={() => router.push('/proyectos')}
              >
                Ir a proyectos
              </button>
            </div>

            {/* Sección de Proyectos Asignados */}
            <div className="mt-5">
              <h3>Proyectos Asignados</h3>
              {proyectos.length > 0 ? (
                <ul className="list-group">
                  {proyectos.map((proyecto) => (
                    <li key={proyecto.id} className="list-group-item">
                      <strong>{proyecto.nombre}</strong>
                      <br />
                      Fecha Inicio: {format(parseISO(proyecto.fecha_inicio), 'Pp')}
                      <br />
                      Fecha Fin: {format(parseISO(proyecto.fecha_fin), 'Pp')}
                    </li>
                  ))}
                </ul>
              ) : (
                <p>No tienes proyectos asignados.</p>
              )}
            </div>

            {/* Calendario para mostrar proyectos */}
            <div className="mt-5">
              <h3>Calendario de Proyectos</h3>
              <Calendar
                value={selectedDate}
                onChange={setSelectedDate}
                tileClassName={tileClassName}
              />
            </div>
          </>
        ) : (
          <h1 className="display-4 text-danger">Por favor, inicia sesión</h1>
        )}
      </div>
```

ProyectosScreen.js

Funcionalidad para la gestión de proyectos, permitiendo a los usuarios crear, actualizar, ver y eliminar proyectos.

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

const ProyectosScreen = () => {
  const [proyectos, setProyectos] = useState([]);
  const [usuarios, setUsuarios] = useState([]);
  const [usuarioActual, setUsuarioActual] = useState(null); // Estado para almacenar el usuario actual
  const [nuevoProyecto, setNuevoProyecto] = useState({
    nombre: '',
    descripcion: '',
    usuarioAsignado: '', // Aquí guardamos el ID del usuario
    fecha_inicio: '',
    fecha_fin: '',
    estado: 'pendiente',
  });
  const [editando, setEditando] = useState(false);
  const [proyectoActual, setProyectoActual] = useState(null);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchProyectos = async () => {
      try {
        const response = await api.get('/proyectos'); // Ruta para obtener todos los proyectos
        console.log('Proyectos obtenidos:', response.data); // Log para depuración
        setProyectos(Array.isArray(response.data) ? response.data : []);
      } catch (error) {
        setError('No se pudieron cargar los proyectos');
        console.error('Error al cargar proyectos:', error); // Log para depuración
      }
    };

    const fetchUsuarios = async () => {
      try {
        const response = await api.get('/usuarios'); // Ruta para obtener usuarios
        console.log('Usuarios obtenidos:', response.data); // Log para depuración
        setUsuarios(Array.isArray(response.data) ? response.data : []);
      } catch (error) {
        setError('No se pudieron cargar los usuarios');
        console.error('Error al cargar usuarios:', error); // Log para depuración
      }
    };

    const fetchUsuarioActual = async () => {
      try {
        const response = await api.get('/session'); // Ruta para obtener la sesión del usuario actual
        console.log('Usuario actual:', response.data);
        setUsuarioActual(response.data.user); // Almacena el usuario actual en el estado
        setNuevoProyecto((prevState) => ({
          ...prevState,
          usuarioAsignado: response.data.user.id, // Establecemos el ID como valor por defecto
        }));
      } catch (error) {
        console.error('Error al obtener el usuario actual:', error);
      }
    };

    fetchProyectos();
```

```
      fecha_inicio: '',
      fecha_fin: '',
      estado: 'pendiente',
    });
  } catch (error) {
    setError('Error al crear el proyecto');
    console.error('Error al crear proyecto:', error);
  }
};

const handleActualizarProyecto = async (e) => {
  e.preventDefault();

  // Validar que el usuario asignado existe
  const usuario = usuarios.find(u => u.id === parseInt(nuevoProyecto.usuarioAsignado));

  if (!usuario) {
    setError('El usuario asignado no existe');
    return;
  }

  try {
    // Actualizar proyecto existente
    await api.put(`/proyectos/${proyectoActual.id}`, {
      ...nuevoProyecto,
      usuarioAsignado: usuario.id  // Enviamos el ID del usuario al backend
    });
    console.log('Proyecto actualizado');
    // Actualiza el proyecto en la lista
    setProyectos(proyectos.map(p => (p.id === proyectoActual.id ? { ...p, ...nuevoProyecto } : p)));

    // Reiniciar los valores del formulario y el estado de edición
    setNuevoProyecto({
      nombre: '',
      descripcion: '',
      usuarioAsignado: usuarioActual ? usuarioActual.id : '', // Restablece el ID del usuario actual
      fecha_inicio: '',
      fecha_fin: '',
      estado: 'pendiente',
    });
    setEditando(false);
    setProyectoActual(null);
  } catch (error) {
    setError('Error al actualizar el proyecto');
    console.error('Error al actualizar proyecto:', error);
  }
};

const handleEditarProyecto = (proyecto) => {
  setNuevoProyecto({
    nombre: proyecto.nombre,
    descripcion: proyecto.descripcion,
    usuarioAsignado: proyecto.usuario_asignado ? proyecto.usuario_asignado.toString() : '', // Colocamos el ID del usuario asignado
    fecha_inicio: proyecto.fecha_inicio,
    fecha_fin: proyecto.fecha_fin,
    estado: proyecto.estado || 'pendiente',
```

Se gestionan los proyectos en la aplicación. Al cargarse, se obtiene la lista de proyectos y usuarios, así como el usuario actual. Este componente permite crear nuevos proyectos y actualizarlos o eliminarlos según sea necesario.

También, contiene un formulario para ingresar el nombre, descripción, fechas y muestra la lista de proyectos, con botones para editar o eliminar cada uno. Si ocurre un error durante las acciones, se muestra un mensaje.

```jsx
return (
  <div className="container">
    <h1 className="mb-4">Gestión de Proyectos</h1>

    <form onSubmit={handleCrearProyecto} className="mb-4">
      <div className="form-group">
        <label>Nombre del Proyecto:</label>
        <input
          type="text"
          value={nuevoProyecto.nombre}
          onChange={(e) => setNuevoProyecto({ ...nuevoProyecto, nombre: e.target.value })}
          className="form-control"
          required
        />
      </div>

      <div className="form-group">
        <label>Descripción del Proyecto:</label>
        <textarea
          value={nuevoProyecto.descripcion}
          onChange={(e) => setNuevoProyecto({ ...nuevoProyecto, descripcion: e.target.value })}
          className="form-control"
          required
        />
      </div>

      <div className="form-group">
        <label>Usuario Asignado:</label>
          <select
            value={nuevoProyecto.usuarioAsignado}
            onChange={(e) => setNuevoProyecto({ ...nuevoProyecto, usuarioAsignado: e.target.value })}
            className="form-control"
            required
          >
            {usuarios.length === 0 ? (
              <option value="">Cargando usuarios...</option>
            ) : (
              usuarios.map((usuario) => (
                <option key={usuario.id} value={usuario.id}>
                  {usuario.name}
                </option>
              ))
            )}
          </select>
      </div>

      <div className="form-group">
        <label>Fecha Inicio del Proyecto:</label>
        <input
          type="datetime-local"
          value={nuevoProyecto.fecha_inicio}
          onChange={(e) => setNuevoProyecto({ ...nuevoProyecto, fecha_inicio: e.target.value })}
          className="form-control"
          required
        />
      </div>
```

WithAuth.js

Protege las rutas de la aplicación, permitiendo el acceso solo a usuarios autenticados. Cuando el componente se carga, ejecuta una función verifySession que utiliza checkSession() para determinar si hay una sesión activa. Si no se detecta una sesión, redirige al usuario a la página de login usando router.push('/login'). Si hay una sesión activa, el componente original (WrappedComponent) se muestra como normalmente lo haría, pasando cualquier prop que se necesite.

```jsx
import { useEffect } from 'react';
import { useRouter } from 'next/router';
import { checkSession } from './api'; // Función que verifica si hay una sesión activa

const WithAuth = (WrappedComponent) => {
  const AuthenticatedComponent = (props) => {
    const router = useRouter();

    useEffect(() => {
      const verifySession = async () => {
        const sessionData = await checkSession();
        console.log('Sesión verificada:', sessionData); // Agrega este log para depurar
        if (!sessionData.loggedIn) {
          router.push('/login');
        }
      };

      verifySession();
    }, []);

    return <WrappedComponent {...props} />;
  };

  return AuthenticatedComponent;
};

export default WithAuth;
```

Estructura de páginas

_app.js

Este archivo se encarga de envolver la aplicación y es el punto de entrada principal. Importa los estilos globales de Bootstrap y el archivo de estilos globals.css. Luego, renderiza el componente de página actual (Component) que se esté visitando, pasando cualquier propiedad (pageProps) que necesite. Este archivo asegura que los estilos globales y la configuración general se apliquen en todas las páginas de la aplicación.

```
// Importamos Bootstrap y los estilos globales
import 'bootstrap/dist/css/bootstrap.min.css'; // Importa Bootstrap globalmente
import '../app/globals.css'; // Si global.css está en src/app

// Este es el componente principal de la aplicación
export default function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />;
}
```

dashboard.js

Define la página del Dashboard, donde se importa el componente DashboardScreen para mostrar el contenido principal. Esta página está envuelta en WithAuth, lo que significa que solo los usuarios autenticados pueden acceder a ella. Si un usuario no está autenticado, se redirige automáticamente a la página de login.

```
import DashboardScreen from '../components/DashboardScreen';
import Link from 'next/link'; // Para utilizar enlaces de navegación
import WithAuth from '../components/WithAuth'; // Importamos WithAuth

const DashboardPage = () => {
  return (
    <div>
      {/* Renderizamos el contenido del Dashboard */}
      <DashboardScreen />
    </div>
  );
};

// Exportamos la página con la protección de WithAuth
export default WithAuth(DashboardPage);
```

login.js

Contiene la página de login, importa y muestra el componente LoginScreen, el cual maneja el formulario de inicio de sesión. Esta página no está protegida, ya que se necesita permitir el acceso a cualquier usuario que desee iniciar sesión.

```
import LoginScreen from '../components/LoginScreen';

const LoginPage = () => {
  return (
    <div>
      <LoginScreen />
    </div>
  );
};

export default LoginPage;
```

register.js

Registro de nuevos usuarios. Importa el componente RegisterScreen, que contiene el formulario y lógica de registro. Al renderizar RegisterScreen, se presenta un formulario para que el usuario ingrese su nombre, correo electrónico y contraseña para crear una cuenta.

```
import RegisterScreen from '../components/RegisterScreen';

const RegisterPage = () => {
  return (
    <div>
      <RegisterScreen />
    </div>
  );
};

export default RegisterPage;
```

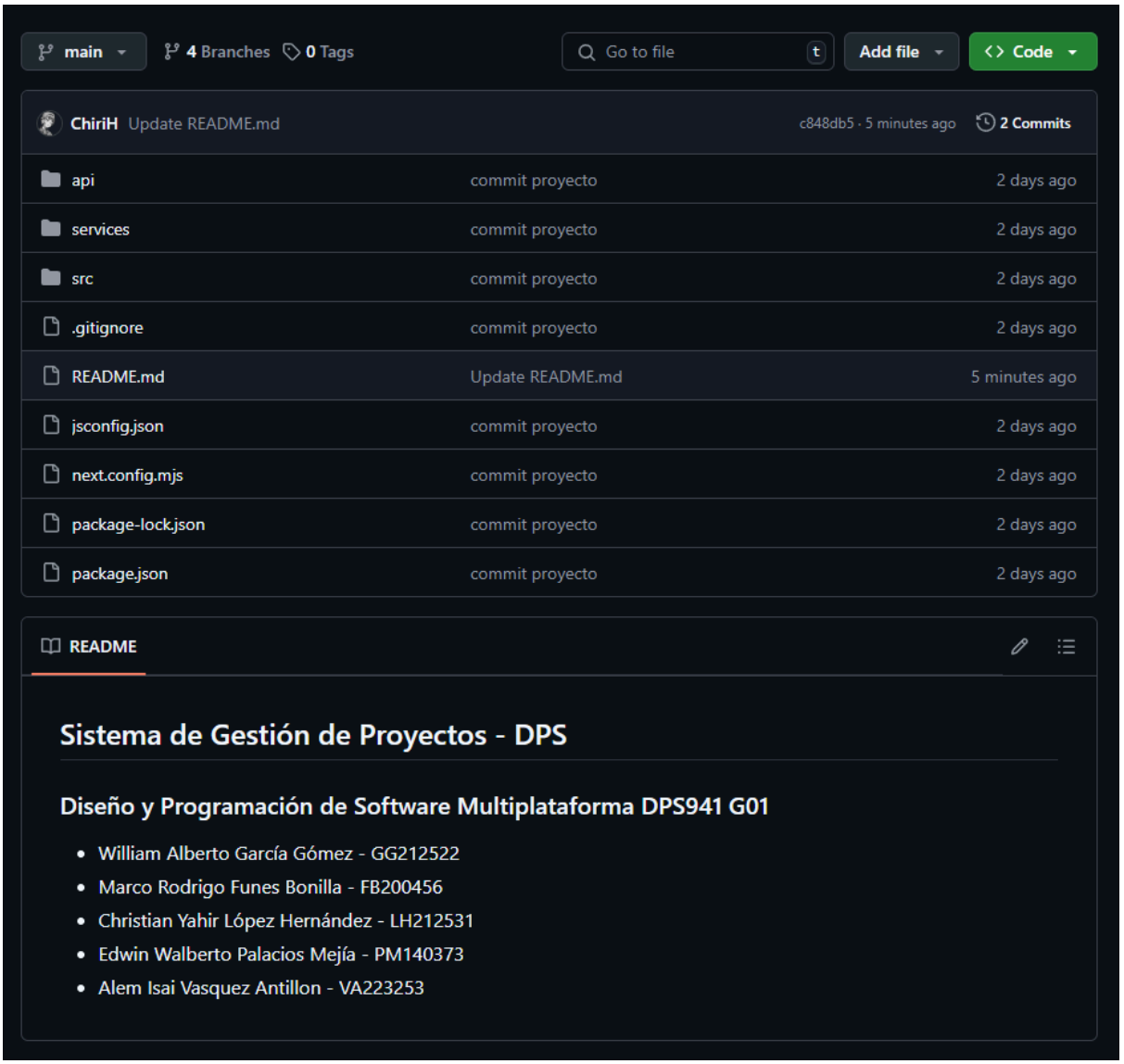
page.js

Define la página de inicio y utiliza useRouter de Next.js para redirigir a los usuarios a las páginas de inicio de sesión o registro según su elección.

```
"use client"; // Indica que este es un componente del lado del cliente

import { useRouter } from 'next/navigation';
import 'bootstrap/dist/css/bootstrap.min.css';
import styles from "./page.module.css"; // Estilos personalizados

export default function Home() {
  const router = useRouter(); // Instancia de useRouter para manejar la navegación

  const handleLoginClick = () => {
    router.push('/login'); // Redirigir a la página de Login
  };

  const handleRegisterClick = () => {
    router.push('/register'); // Redirigir a la página de Registro
  };

  return (
    <div className="container mt-5">
      <main className="text-center">
        <h1 className="display-4 mb-4">Bienvenido al Sistema de Gestión de Proyecto
        <p className="lead mb-4">Por favor, selecciona una opción para comenzar:</p

        <div className="d-flex justify-content-center gap-3">
          <button onClick={handleLoginClick} className="btn btn-primary btn-lg">
            Iniciar Sesión
          </button>
          <button onClick={handleRegisterClick} className="btn btn-secondary btn-lg
            Registrarse
          </button>
        </div>
      </main>

      <footer className="text-center mt-5">
        <p>Proyecto 1 - Sistema de Gestión de Proyectos</p>
      </footer>
    </div>
  );
}
```

GitHub

Repositorio: https://github.com/ChiriH/Proyecto1DPS/
Vercel: https://proyecto1-dps.vercel.app/



¿Cómo ejecutar el sistema?

Para acceder al sistema se puede dirigir directamente al sitio en Vercel. También, se puede descargar el archivo como ZIP localmente.

1. Seleccionar Download ZIP para obtener el archivo, y una vez descargado descomprimirlo.
2. Instalar las dependencias abriendo una terminal y navegar a la carpeta descomprimida del proyecto.
3. Ejecutar el comando npm install para instalar todas las dependencias necesarias.
4. Configurar la API en servidor local como XAMPP o MAMP, dentro de una carpeta “api”.
5. Ejecuta el proyecto en la terminal con el comando npm run dev para iniciar el servidor de desarrollo.
6. Abrir y dirigirse a http://localhost:3000 para ver la aplicación en funcionamiento.