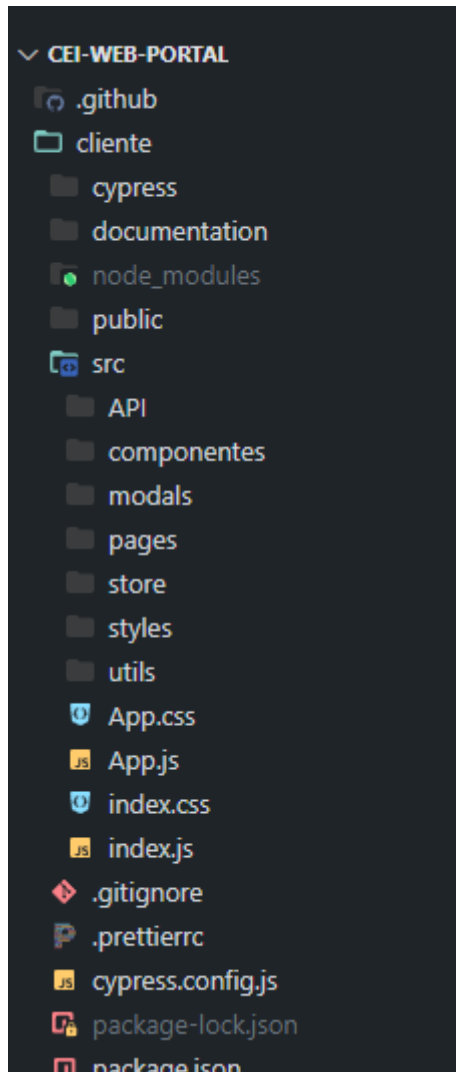


DOCUMENTACION CEI WEB PORTAL

<https://github.com/MGeovany/cei-web-portal>

Sobre la arquitectura utilizada

En cuanto a la estructura utilizada de carpetas utilizamos la Estructura por funcionalidad o modular de la cual es la principal que se usa en proyectos de react donde se dividio por carpetas y funciones cada componente.

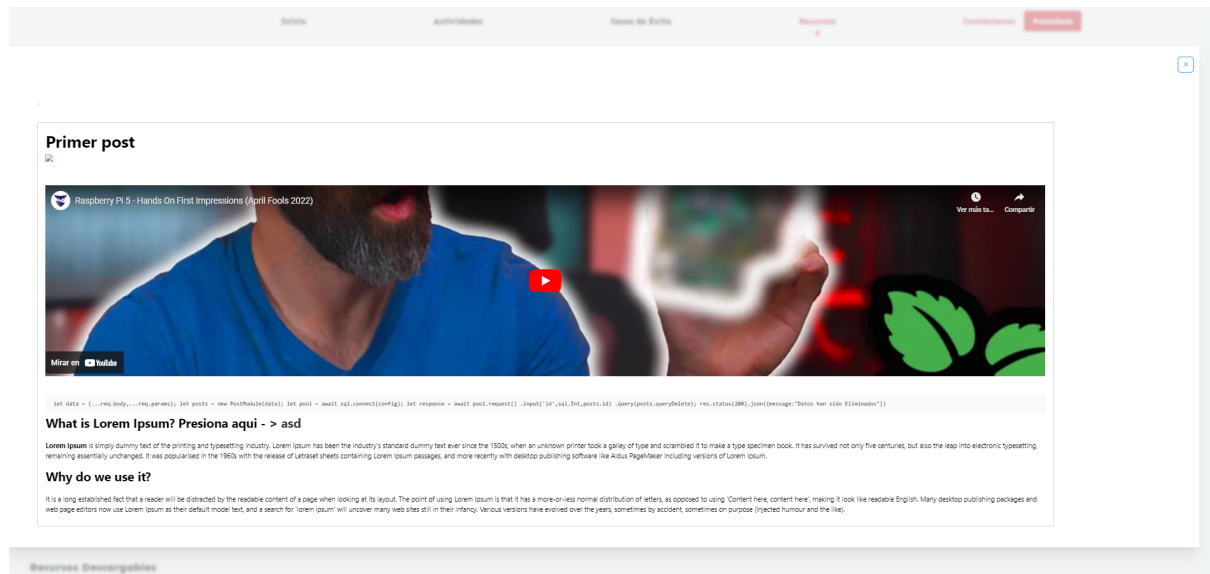


La parte del front end fue creada con `npx create-react-app` de la cual fue creada dentro de la carpeta de cliente para la parte visual y **server** para todo lo relacionado con el servidor y la base de datos.

Dentro de cliente tenemos **cypress** encargado de los test end-to-end, en **public** cuenta con muchos assets utilizados solamente como recursos del front-end asi como iconos y otros elementos que se necesitaron para la forma visual del sitio.

Dentro de **src** esta todo el codigo principal, **components** esta la parte que hace referencia a la modularizacion de porciones o piezas de codigo que pueden llegar a funcionar en distintos lugares.

En **modals** son todos los cuadros de dialogos que muestran o permiten al usuario interactuar con la aplicaicon



pages cuenta con todas las rutas.

store se explica mas adelante sin embargo es la carpeta que hace referencia al manejo de estado global el cual es controlado con Redux

styles hace referencia a estilos para componentes separados para mejor orden, de igual forma esto podria mejorarse haciendo una carpeta con cada componente y sus estilos.

utils se uso para incluir cualquier svg como codigo de react y luego ser importado como un componente jsx

Fuera de las carpetas se utilizo eslint standard junto con prettier del cual en sus documentos correspondientes esta la configuracion utilizada para cada uno.

API REST

La API REST está desarrollado en Javascript con ExpressJS en el cual se utilizaron las siguientes dependencias:

- **bcrypt:** para el hashing de la contraseña de la tabla de usuarios.
- **cors:** control de acceso a través del protocolo HTTP.
- **express:** Framework para desarrollar la aplicación
- **mssql:** librería para manipular los datos del Servidor de base de datos de MSSQL.
- **multer:** middleware para manejar los datos enviados por el atributo "Form Data"
- **sharp:** librería para manipular imágenes almacenadas.

dependencias de desarrollo

- **env-cmd:** ejecución de la aplicación con el archivo `.env`.
- **nodemon:** librería para ejecutar la aplicación en desarrollo.

Rutas

Método	Ruta	Requerimientos
Casos		
GET	/casos	
GET	/casos/:id	
POST	/casos	body: titulo string cuerpo string tipo string imagenEncabezado string usuarioCreador int seccionCasos string
PUT	/casos	body id int titulo string cuerpo string tipo string imagenEncabezado string usuarioCreador int seccionCasos string
DELETE	/casos/:id	
Contactanos		
GET	/contactanos	
GET	/contactanos/:id	

POST	/contactanos	body: nombre string 50 email string 70 comentario string 300 telefono string 20
PUT	/contactanos	body: id int nombre string 50 email string 70 comentario string 300 telefono string 20
DELETE	/casos/:id	
Imágenes		
GET	/images/:name	
POST	/upload	FormData file any file
Inicio		
GET	/inicio	
Integrantes		
GET	/casos/:idproyecto/integrantes	
GET	casos/:idproyecto/integrantes/:id	
POST	/casos/:idproyecto/integrantes	body: idproyecto string nombre string correo string
PUT	/casos/:idproyecto/integrantes/:id	body: id int idproyecto string nombre string correo string
DELETE	casos/:idproyecto/integrantes/:id	
Login		
POST	/login	body: usuario string

		contrasena	string
Opciones para los selects			
GET	/casos/opciones/lugarsede	retorna value (id)	string
		label (nombre)	string
GET	/casos/opciones/organizador	retorna value (id)	string
		label (nombre)	string
Post			
GET	/post		
GET	/post/:id		
POST	/post	body: titulo	string
		cuerpo	string
		tipo	int
		imagenEncabezado	string
		usuarioCreador	int
		proyecto	int
PUT	/post	body id	int
		titulo	string
		cuerpo	string
		tipo	int
		imagenEncabezado	string
		usuarioCreador	int
		proyecto	int
DELETE	/post/:id		
Postulación			
GET	/postulacion		
GET	/postulacion/:id		
POST	/postulacion	body: representante	string
		correo	string
		cuenta	int
		celular	string
		genero	int
		tipo	int
		descripcion	string
		sede	int
		redesSociales	int
		rubro	int

		equipoTrabajo expectativas estado	int int int
PUT	/postulacion	body id trepresentante correo cuenta celular genero tipo descripcion sede redesSociales rubro equipoTrabajo expectativas estado	int string int string int int string int int int int int int
DELETE	/post/:id		
Post			
GET	/usuario		
GET	/usuario/:id		
POST	/usuario	body: usuario nombre string contrasena correo	string string string
PUT	/usuario	body id usuario nombre string contrasena correo	int string string string
DELETE	/usuario/:id		

Sobre la base de datos

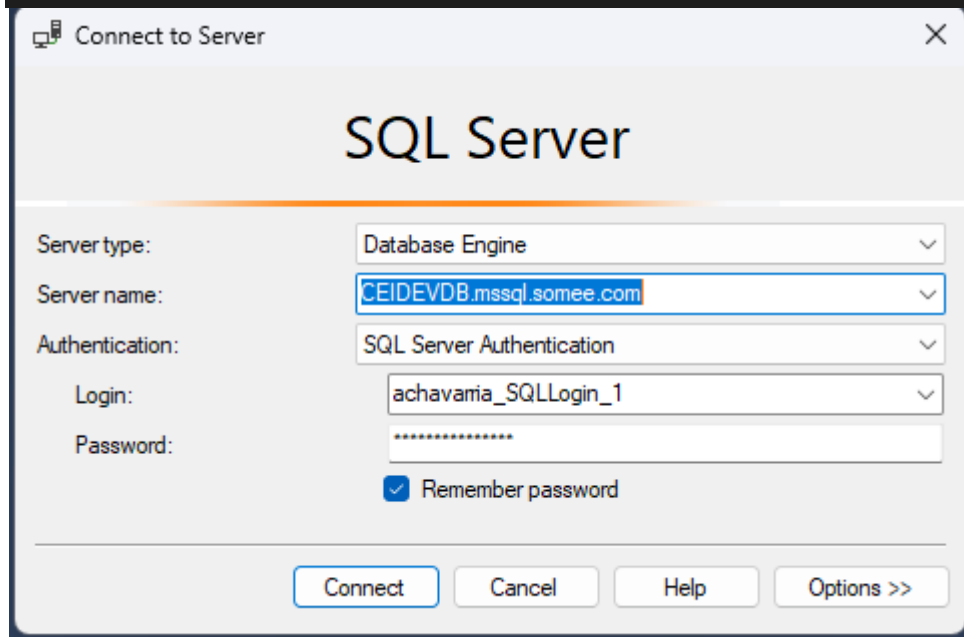
La base de datos está en el sitio de host gratuito somee.com, pero se puede para acceder a ella si usan los siguientes datos en Microsoft Sql Server:

SERVER=CEIDEVDB.mssql.somee.com

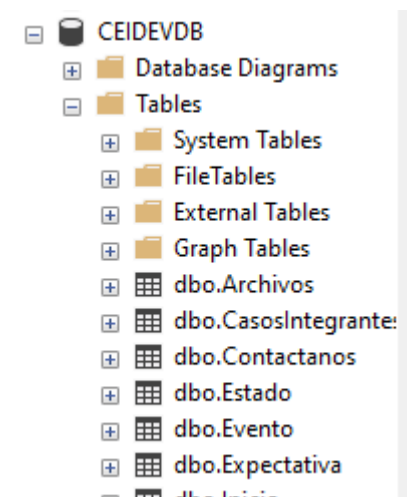
USER=achavarria_SQLLogin_1

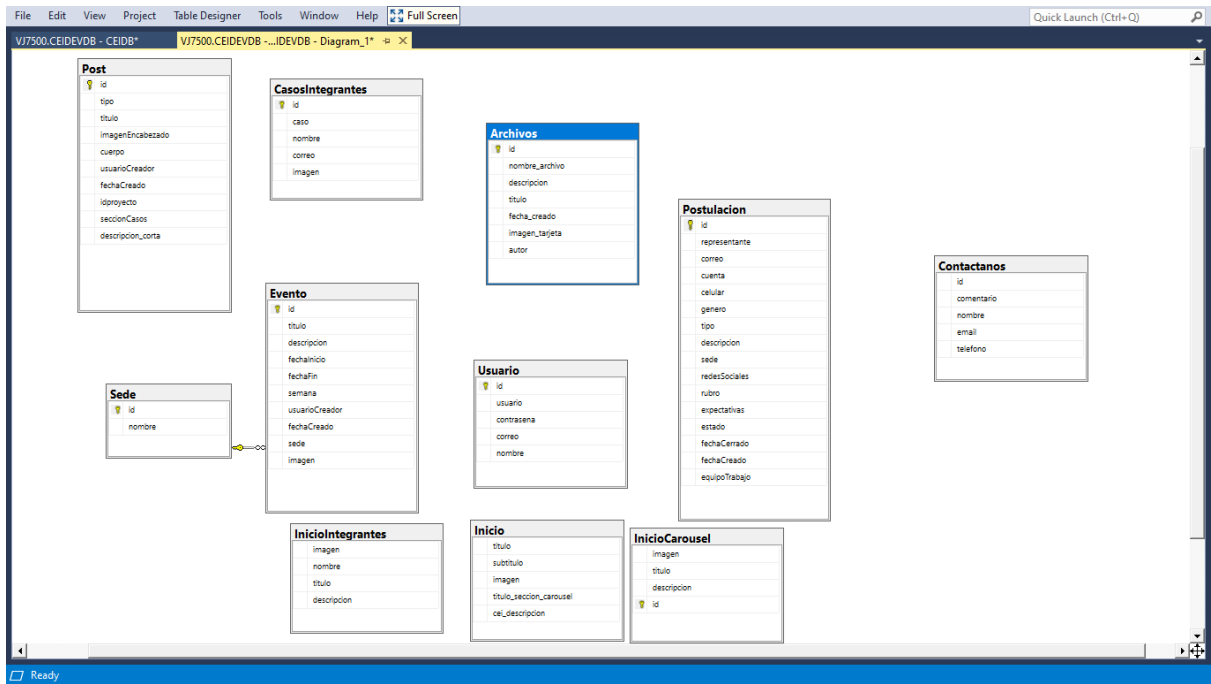
DB=CEIDEVDB

PASSWD=ljznvqsw71



El nombre de la base de datos es CEIDEVDB





Sobre los estilos y el responsive

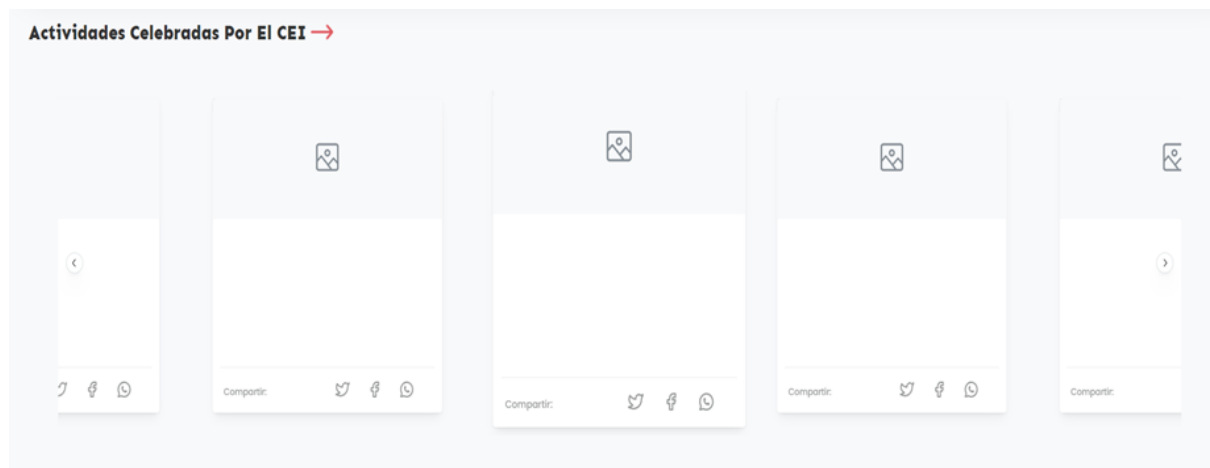
Para el desarrollo del proyecto utilizamos la librería de componentes mantine la cual fue guía en la implementación de elementos como lo fue el carrusel , modals , navbars , formularios , elementos responsive y entre otros componentes.

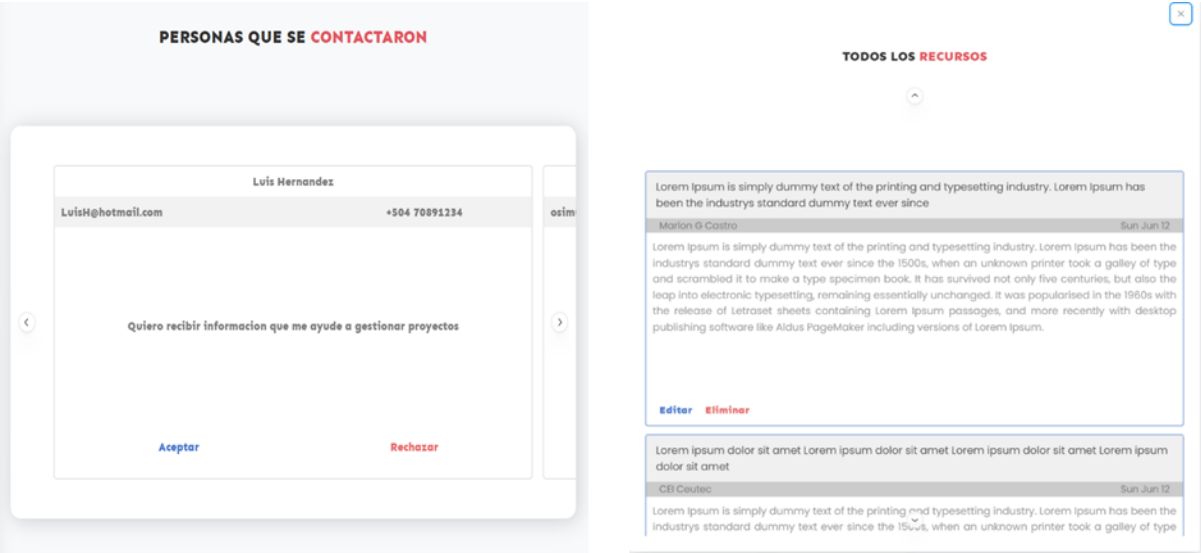
Carrusel

El elemento de carrusel fue utilizado para mostrar los diferentes blogs , actividades , casos y recursos de la pagina haciendo uso también del mismo para mostrar elementos de forma responsiva como lo es contactos y recursos

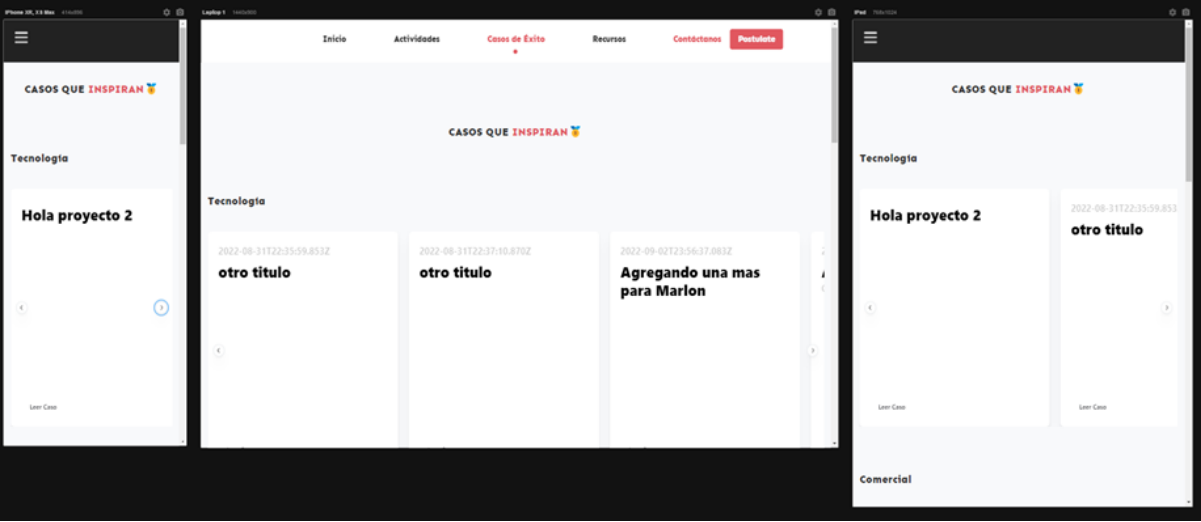
Link del elemento:

<https://mantine.dev/others/carousel/>





Carrusel Responsive

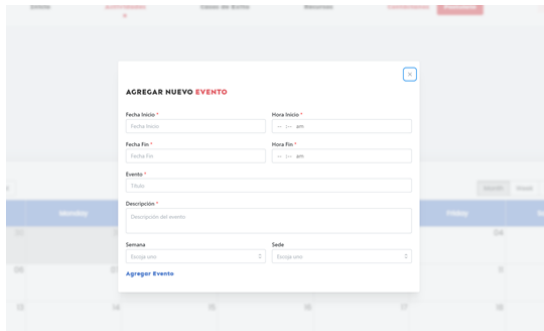


Modals

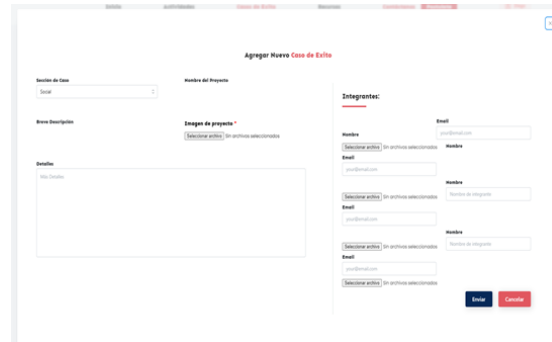
Elemento de modals fue aplicado mayoritariamente en la pestañas de administrador donde este se abriría en caso de necesitar un formulario u otras opciones de la sección.

Link del elemento de Modals:

<https://mantine.dev/core/modal/>

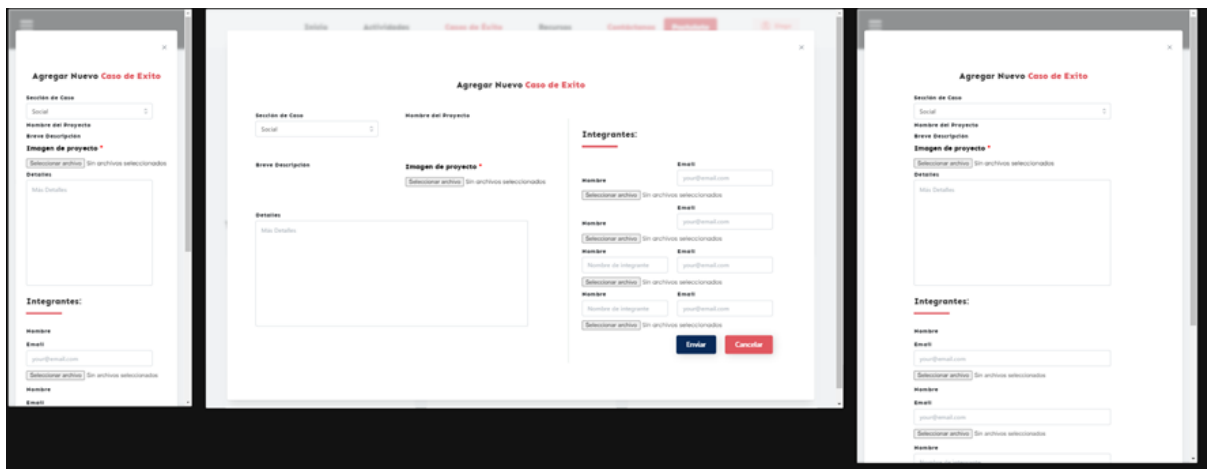


A screenshot of a modal window titled "AGREGAR NUEVO EVENTO". The modal contains several form fields: "Fecha Inicio" and "Fecha Fin" with date pickers, "Hora Inicio" and "Hora Fin" with time pickers, "Evento" with a title field and a description area, and "Semana" with a dropdown menu. There is also a "Agregar Evento" button at the bottom.



A screenshot of a modal window titled "Agregar Nuevo Caso de Exito". The modal contains several form fields: "Sección de Caso" with a dropdown menu, "Nombre del Proyecto" with a text field, "Breve Descripción" with a text area, "Imagen de proyecto" with a file upload button, "Detalles" with a text area, and "Integrantes" with a list of members, each having a name field, an email field, and a role dropdown. There are "Enviar" and "Cancelar" buttons at the bottom.

Modals Responsive

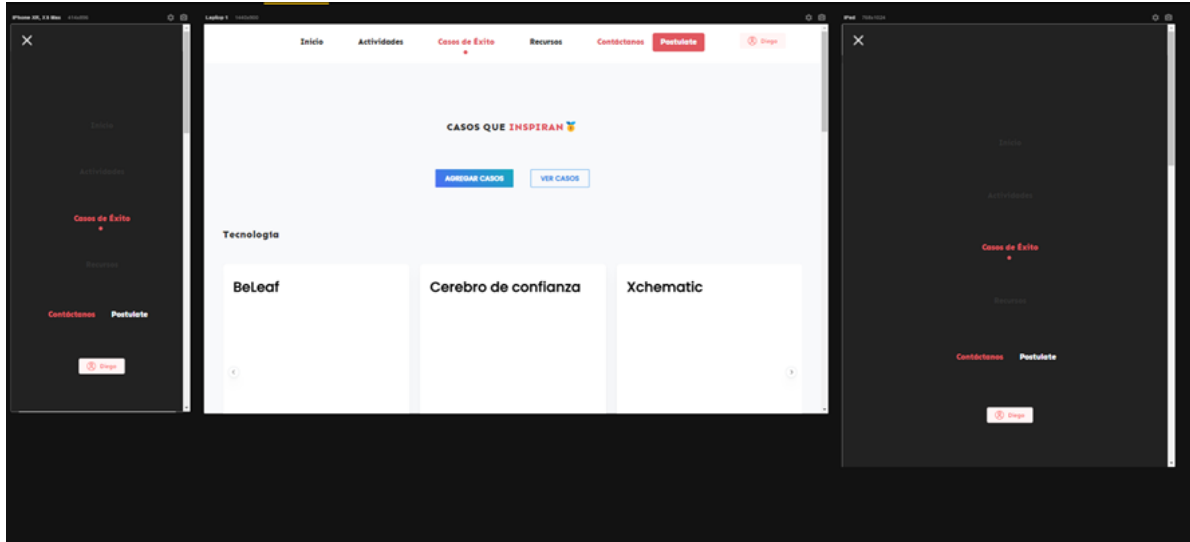


Three screenshots showing the "Agregar Nuevo Caso de Exito" modal in different responsive states. The first screenshot shows the modal on a small screen, where the layout is compact. The second screenshot shows the modal on a medium screen, where the layout is more spread out. The third screenshot shows the modal on a large screen, where the layout is even more spread out. The modal contains the same form fields as the previous screenshots, but the layout is adjusted to fit the screen size.

Navbar

Barra de navegación de la página

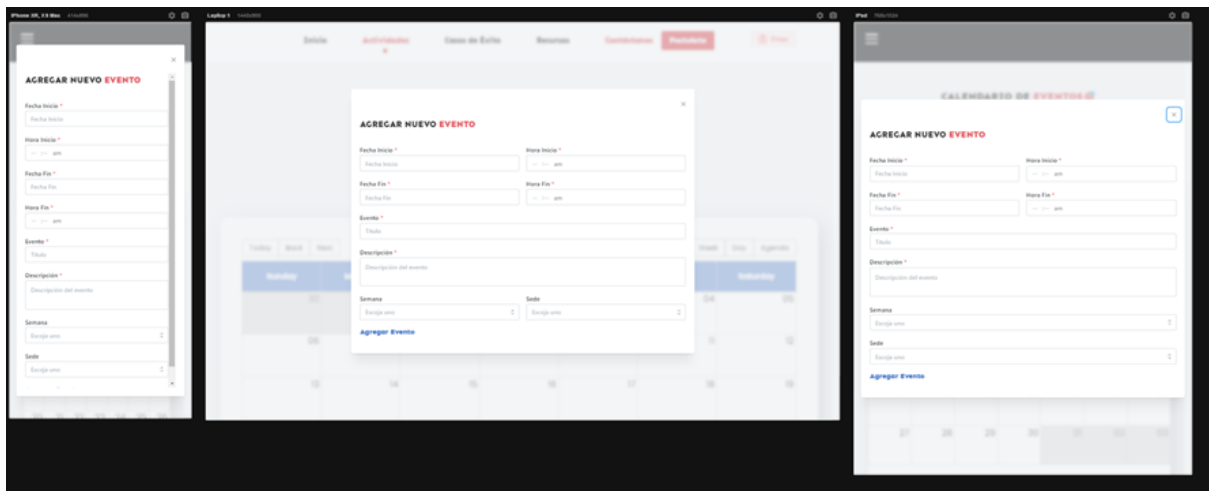
NavBar Responsive

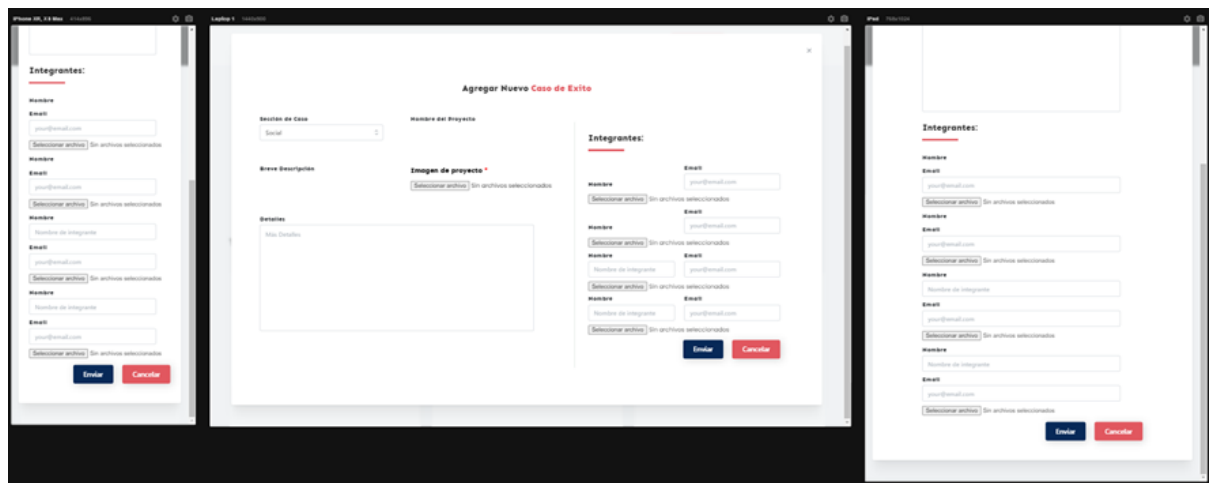


formularios

Formularios usados para la agregar y remover información de la pagina

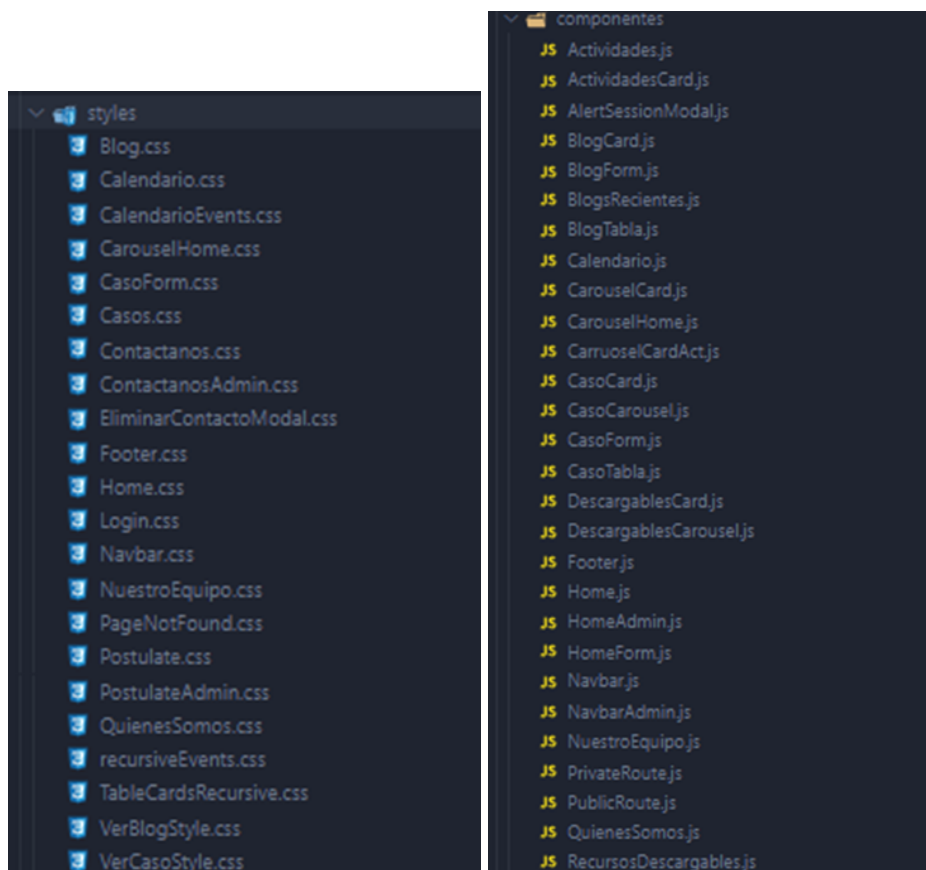
Formularios Responsive





Elementos y su hoja de estilos

Se implementaron múltiples archivos de estilo (.css) para la estilización de la mayoría de los componentes creados , en donde adentro del mismo se definían las reglas del para que los elementos fueran responsivos al cambio del tamaño de la pagina.



Sobre los componentes

Relación entre componentes.

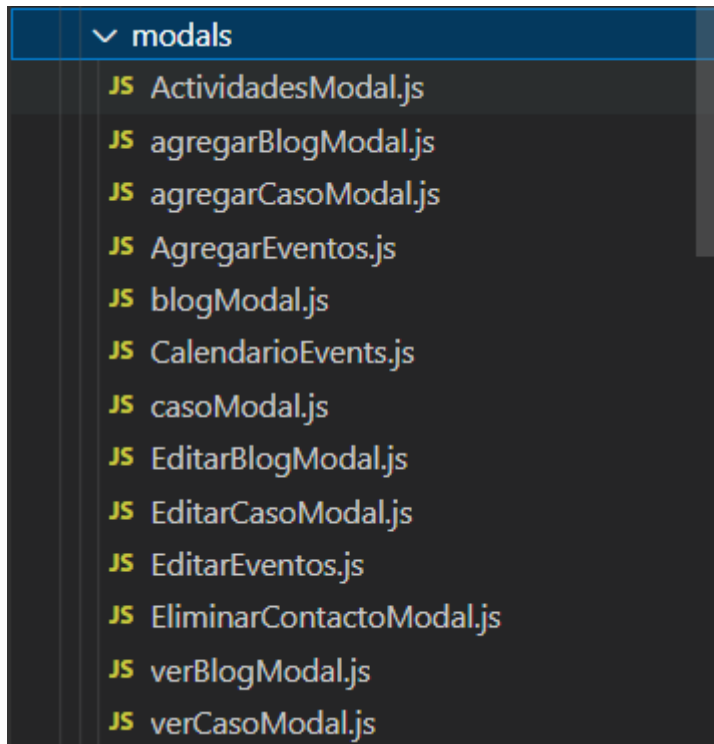
Los componentes los hemos dividido para dos interfaces gráficas diferentes por un lado tenemos a los componentes para el interfaz gráfica de los usuarios y del otro lado a los de los administradores. De igual forma tenemos componentes que utilizamos en ambas interfaces gráficas.

Para diferenciar los componentes exclusivamente para el interfaz gráfica de los administradores les agregamos la palabra “Admin” al final del nombre del componente, algunos ejemplos de estas páginas son: HomeAdmin.js, NavbarAdmin.js, BlogAdmin.js, CasosAdmin.js, etc... La lista total de las páginas con referencia tanto a la interfaz gráfica de los administradores como la de los usuarios la pueden encontrar en el archivo donde instanciamos los componentes en este caso la encontrarán en el archivo llamado “App.js” ahí podrán ver que tenemos dentro de tags “Route” la llamada de los componentes tanto para usuarios (que son los primeros en instanciarse) como la de los administradores.

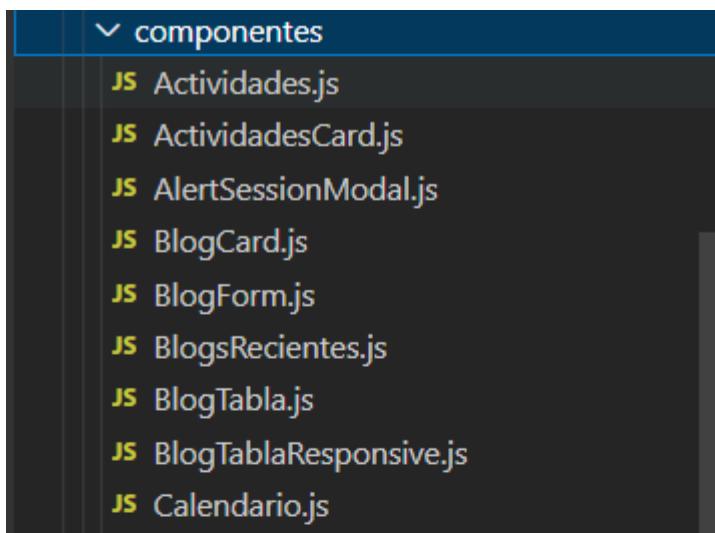
```
JS App.js x
cliente > src > JS App.js > App
72     ActividadesModal,
73     EliminarContactoModal
74   }
75   >
76   <Router>
77     <Routes>
78       <Route path="/" element={PublicRoute} />
79       <Route index element={LandingPage} />
80       <Route path="/calendario" element={CalendarioPage} />
81       <Route path="/casos-de-exito" element={Casos} />
82       <Route path="/blog" element={Blog} />
83       <Route path="/contactanos" element={Contactanos} />
84       <Route path="/postulate" element={Postulate} />
85       <Route path="/login" element={Login} />
86     </Route>
87
88     <Route path="/admin" element={PrivateRoute} />
89       <Route index element={Dashboard} />
90       <Route path="/admin/casos-de-exito" element={CasosAdmin} />
91       <Route path="/admin/calendario" element={CalendarioAdmin} />
92       <Route path="/admin/blog" element={BlogAdmin} />
93       <Route path="/admin/postulate" element={PostulateAdmin} />
94       <Route
95         path="/admin/contactanos"
96         element={ContactanosAdmin} />
97     </Route>
98   </Routes>
99   <Route path="*" element={PageNotFound} />
100 </Router>
101 <Footer />
102 </Router>
```

Los componentes que utilizamos se dividen en 3 carpetas:

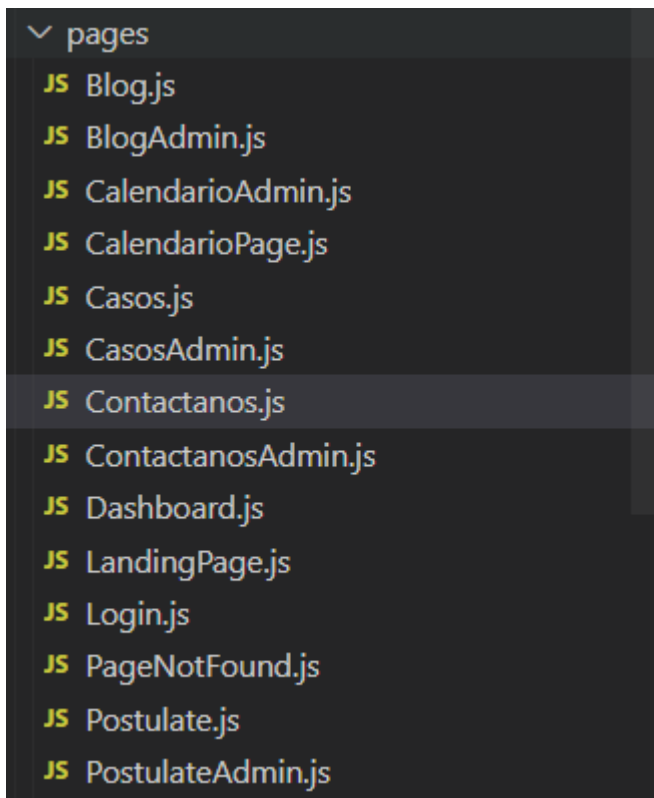
- La carpeta nombrada “modals” en donde se encuentran las configuraciones que tienen todos los modales (ventanas que se abren cuando el usuario interactúa con los carruseles) algunos de estos archivos son: ActividadesModal.js, agregarBlogModal.js, blogModal.js, etc...



- La carpeta nombrada “componentes” en donde tenemos los diferentes componentes que usamos en nuestras interfaces gráficas que son 30 archivos.



- Por último la carpeta nombrada “pages” en donde tenemos archivos que consumen componentes para ordenarlos en la página web.



Sobre la autenticación

Archivos involucrados en la autenticación:

- API/index.js (Contiene el endpoint para verificar el acceso)
- pages/login.js (Contiene la pantalla de login)
- componentes/AlertSessionModal.js (Contiene el modal para renovar la sesion)
- store/slices/login/loginSlice.js (Contiene la logica del login)
- componentes/navbaradmin.js (Contiene el boton para desloguearse)
- componentes/PrivateRoute.js (Limita el acceso a los usuario logueados)
- componentes/PublicRoute.js (Limita el acceso a los usuario no logueados)

Index.js

Recibe como parámetro el usuario y la contraseña, luego devuelve una respuesta desde el servidor para saber si tiene acceso o no a la aplicación

```
export class Auth {
  static validate = async function(usuario, contraseña){
    let resultValidate = {'access':false, 'data':null};


    return await fetch('https://cei1.herokuapp.com/1.0.0/login',{
      method: "POST",
      body: JSON.stringify({usuario: usuario, contraseña: contraseña}),
      headers: {"Content-type": "application/json; charset=UTF-8"}
    })
    .then(response => response.json())
    .then(json => {
      if(json.usuario){
        resultValidate.access = true;
        resultValidate.data = json
        return resultValidate;
      }else{
        resultValidate.access = false;
        resultValidate.data = 'Usuario o contraseña incorrecta';
        return resultValidate;
      }
    })
    .catch(err => {
      resultValidate.access = false;
      resultValidate.data = 'Ocurrió un error al validar el acceso';
      return resultValidate;
    });
  }
}
```

Login.js

Bienvenido de vuelta!

Usuario *

Contraseña *



Olvidó su contraseña?

Ingresar

Autenticación para pruebas:

```
usuario: admin
```

clave: 1234

URL: <http://localhost:3000/login>

La estructura del login está en la función return()

```

return (
  <>
    <div className='login-container'>
      <Container size={820} my={60}>
        <Title
          align='center'
          sx={({theme}) => ({
            fontFamily: `Poppins, ${theme.fontFamily}`,
            fontWeight: 900,
            color: '#333'
          })}
        >
          Bienvenido de vuelta!
        </Title>
        <Paper withBorder shadow='md' p={30} mt={30} radius='md'>
          <form onSubmit={validateAuth()}>
            <TextInput
              label='Usuario'
              placeholder='Tu usuario'
              required
              {...form.getInputProps('usuario')}
            />
            <PasswordInput
              label='Contraseña'
              placeholder='Tu Contraseña'
              required
              mt='md'
              {...form.getInputProps('password')}
            />
            <Group position='apart' mt='md'>
              <Space w="md" />
              <a
                onClick={(event) => event.preventDefault()}
                href='/login'
                size='sm'
              >

```

Esta función muestra la notificación de 'error' o 'cargando' cuando el usuario intenta acceder a la aplicación a través del login

```

const NotificationLogin = () => {
  if (showNotification) {
    return notificationLoading ? (
      <Notification
        loading
        title="Verificando acceso"
        disallowClose
      ></Notification>
    ) : (
      <Notification
        icon={<IconX size={18} />}
        color="red"
        onClose={() => setShowNotification(false)}
      >{messageNotificationError}</Notification>
    )
  }
}

```



Usuario o contraseña incorrecta



Verificando acceso

Esta función es la que se ejecuta al presionar el botón de 'Ingresar' para darle acceso al usuario o negarlo según la respuesta del servidor con las credenciales

```

const validateAuth = () => {
  return form.onSubmit((values) => {
    setShowNotification(true);
    setNotificationLoading(true);
    Auth.validate(values.usuario, values.password).then(response => {
      if(response.access){
        let data = response.data;
        dispatch(login({ usuario: data.usuario, correo: data.correo, nombre: data.nombre }));
      }else{
        setMessageNotificationError(response.data);
        setNotificationLoading(false);
      }
    });
  });
}

```

AlertSessionModal.js

Contiene la estructura del modal

```

return (
  <>
    <Modal
      closeOnClickOutside={false}
      opened={opened}
    >
      <h4 id='countdown'>La sesión expirara en {countdown}s, ¿Deseas mantener la sesión actual?</h4>
      <br/>
      <Group position="apart" spacing="xl" grow>
        <Button onClick={aceptar}>Aceptar</Button>
        <Button color="red" onClick={cerrar} float='right'>Cerrar</Button>
      </Group>
    </Modal>
    <button id='alertSessionModalButton' onClick={() => mostrarModal()}></button>
  </>
);

```

idTimer: es el ID del timer para poder detenerlo cuando el usuario presione uno de los dos botones

maxSecondsAlertSession: es el tiempo máximo que espera el modal por una respuesta del usuario

countdown: es el número por el que empieza la cuenta regresiva

```
var idTimer = null;
const maxSecondsAlertSession = 30;
var countdown = maxSecondsAlertSession;
```

Es la función que muestra y actualiza el timer

```
const mostrarModal = () => {
  setOpened(true);
  clearInterval(idTimer);
  idTimer = setInterval(function(){
    if(countdown <= 0){
      cerrar();
    } else {
      document.getElementById("countdown").innerHTML = `La sesión expirara en ${countdown}s, ¿Deseas mantener la sesión actual?`;
    }
    countdown -- 1;
  }, 1000)
}
```

reinicia el timer

```
const resetTimer = () => {
  clearInterval(idTimer);
  countdown = maxSecondsAlertSession;
}
```

función que se ejecuta cuando el usuario acepta renovar la sesión

```
const aceptar = () => {
  resetTimer();
  newSession();
  setOpened(false);
}
```

Función que se ejecuta cuando el contador llega a cero o el usuario presiona el botón para no renovar la sesión

```
const cerrar = () => {
  resetTimer();
  removeSession();
  window.location.reload();
}
```

LoginSlice.js

Este es el manejador de esta del login, aquí se encuentra la función para loguearse y desloguearse

```
export const loginSlice = createSlice({
  name: 'user',
  initialState: {
    isLoggedIn: validateLogged()
  },
  reducers: {
    login: (state, action) => {
      dataSession.usuario = action.payload.usuario
      dataSession.correo = action.payload.correo
      dataSession.nombre = action.payload.nombre
      state.isLoggedIn = true
      newSession()
    },
    logout: (state, action) => {
      dataSession.usuario = ''
      dataSession.correo = ''
      dataSession.nombre = ''
      state.isLoggedIn = false
      removeSession()
    }
  }
})

export const { login, logout, isAuth } = loginSlice.actions

export const selectUser = (state) => state.user.usuario
export default loginSlice.reducer
```

Key: es la el valor del hash para encriptar los datos

maxMinutesSession: es tiempo que dura cada sesión

dataSession: es la información que se almacena el en localStorage

```
const key = 'CEI2022'
const maxMinutesSession = 15

let dataSession = {
  expira: '',
  usuario: '',
  correo: '',
  nombre: ''
}
```

Esta función obtiene la sesión guarda del localStorage si es que existe, si no existe retorna null


```
export const getSession = () => {
  let session = localStorage.getItem('SESSION')
  if (session) {
    var bytes = CryptoJS.AES.decrypt(session, key)
    return JSON.parse(bytes.toString(CryptoJS.enc.Utf8))
  }
  return null
}
```

Obtiene los minutos que lleva la sesión actual

```
const getMinutesSession = () => {
  let session = getSession()
  if (session) {
    var startSession = new Date(session.expira)
    var now = new Date()
    return (now.getTime() - startSession.getTime()) / 1000 / 60
  }
  return null
}
```

Genera una nueva sesión cada vez que el usuario se loguea o renueva la sesión después de cumplir el tiempo máximo de la sesión

```
export const newSession = () => {
  dataSession.expira = new Date().toLocaleString()
  var ciphertext = CryptoJS.AES.encrypt(
    JSON.stringify(dataSession),
    key
  ).toString()
  localStorage.setItem('SESSION', ciphertext)
  watchSession()
}
```

Elimina la sesión después de que el usuario se desloguea o se cumple el tiempo máximo para la sesión

```
export const removeSession = () => {
  localStorage.removeItem('SESSION')
}
```

Monitorea a cada segundo el tiempo de la sesión, si la sesión ya venció muestra al usuario el modal para renovarla o salir definitivamente

```
const watchSession = () => {
  setTimeout(() => {
    if (!validateLogged()) {
      alertSession()
    } else {
      let session = getSession()
      dataSession.usuario = session.usuario
      dataSession.correo = session.correo
      dataSession.nombre = session.nombre
    }
  }, 1000)
}
```

Muestra el modal de que la sesión ya expiro, para que el usuario la pueda renovar

```
const alertSession = () => {
  let modal = document.getElementById('alertSessionModalButton')
  if (modal) {
    modal.click()
  }
}
```

Valida que el usuario esté logueado, verificando que exista una sesion en localStorage y si existe valida que no haya expirado todavia

```
const validateLogged = () => {
  let minutesSession = getMinutesSession()
  const isLoggedIn =
    minutesSession > maxMinutesSession || minutesSession == null ? false : true
  isLoggedIn ? watchSession() : removeSession()
  return isLoggedIn
}
```

navbaradmin.js

Muestra el botón con el nombre del usuario logueado y cuando se da clic en él muestra otro botón para desloguearse

```

</li>
<Popover width={200} position='bottom' withArrow shadow='md'>
  <Popover.Target>
    <Button variant='light' color='red'>
      <ThemeIcon color='red' size='sm' radius='xl' variant='outline'>
        <IconUser size={34} />
      </ThemeIcon>
      <Space w='xs' />
      {getSession().nombre}
    </Button>
  </Popover.Target>
  <Popover.Dropdown>
    <Center>
      <Button
        color='red'
        leftIcon={<IconLogout size={14} />}
        size='md'
        compact
        style={{ zIndex: 100 }}
        onClick={() => salir()}
      >
        Cerrar sesion
      </Button>
    </Center>
  </Popover.Dropdown>
</Popover>
</li>
</ul>

```

PrivateRoute.js

Verifica si el usuario está logueado para permitirle entrar a las páginas administrativas, si no esta logueado y intenta ingresar a las páginas administrativas lo redirecciona al login

```

const useAuth = () => {
  const user = useSelector((state) => state.login)
  return user.isLogged;
}

const PrivateRoute = () => {
  return useAuth() ? (
    <>
      <NavbarAdmin />
      <Outlet />
      <AlertSessionModal />
    </>
  ) : (
    <Navigate to='/login' />
  )
}

export default PrivateRoute;

```

PublicRoute.js

Se repite el caso anterior pero al reves

```

const useAuth = () => {
  const user = useSelector((state) => state.login)
  return user.isLogged
}

const isLoginPage = () => {
  return window.location.pathname === '/login' ? true : false
}

const PublicRoute = () => {
  return useAuth() ? (
    <Navigate to='/admin' />
  ) : isLoginPage() ? (
    <Outlet />
  ) : (
    <>
      <Navbar />
      <Outlet />
    </>
  )
}

export default PublicRoute

```