

Asset Empleado

Información General del Proyecto

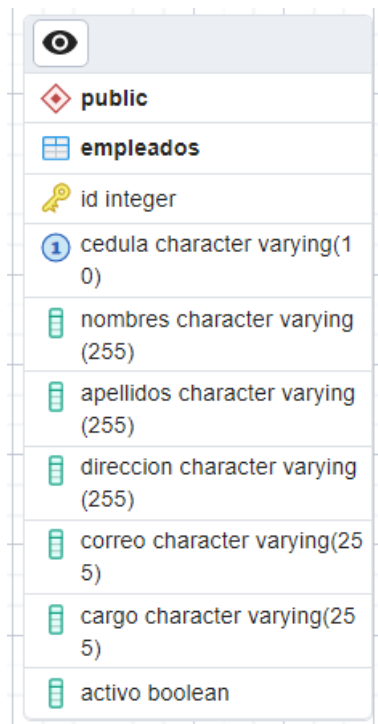
Descripción

El proyecto permite ingresar el empleado que se va a registrar en la base de datos.

Prerrequisitos

El proyecto fue realizado con tecnologías PERN Stack (PostgreSQL, Express, React y Node. Js.).

Para la base de datos se puede observar las tablas y los campos que contiene cada una de estas en el siguiente modelo.



The image shows a database schema diagram for a table named 'empleados' within a 'public' schema. The table has the following fields:

Field Name	Field Type
id	integer (Primary Key)
cedula	character varying(10)
nombres	character varying(255)
apellidos	character varying(255)
direccion	character varying(255)
correo	character varying(255)
cargo	character varying(255)
activo	boolean

Para el BackEnd se ocupó las librerías:

- Axios
- Cors
- Dotenv
- Express
- Morgan
- PG

Mientras que para el FrontEnd se ocupó las dependencias:

- React
- React-dom
- React-router-dom
- @mui/material

Instalación

Este proyecto consiste en un Web Frontend Application y Web Backend Application.

Para iniciar el BackEnd se debe usar el siguiente comando:

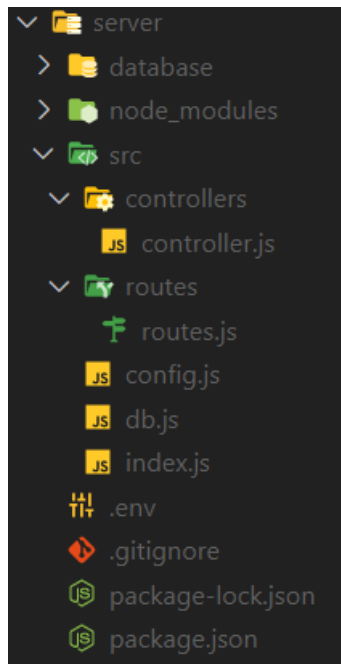
```
npm run dev
```

Para iniciar el FrontEnd se debe ocupar los siguientes comandos:

```
cd client  
npm start
```

BackEnd API Server

El BackEnd API Server está estructurado de la siguiente manera.



Controller

Dentro de la carpeta controllers se ubica el archivo controller.js que contiene los controladores para realizar las rutas.

Dentro del archivo inventario.controller.js se tiene los siguientes métodos:

getEmpleados

Este método nos ayuda a obtener todos los registros de la tabla de empleado.

```
//Encontrar empleados  
const getEmpleados = async (req, res, next) => {  
  try {  
    const empleados = await pool.query("SELECT * FROM empleados");  
    console.log(empleados);  
  }  
}
```

```

        res.json(empleados.rows);
    } catch (error) {
        next(error)
    }
}

```

getEmpleado

Este método nos ayuda a obtener un solo registro de la tabla de empleado. Para realizar este método se debe enviar el parámetro ID del empleado a buscar.

```

//Encontrar un empleado
const getEmpleado = async (req, res, next) => {
    try {
        const {id} = req.params
        const result = await pool.query('SELECT * FROM empleados WHERE id
=id', [id])
        if (result.rows.length === 0) return res.status(404).json({
message: 'Empleado no encontrado' })
        res.json(result.rows[0]);
    } catch (error) {
        next(error)
    }
}

```

crearEmpleado

Este método nos ayuda a crear registros de empleados dentro de la base de datos. Para ejecutar este método se debe enviar los siguientes parámetros en formato JSON.

Parámetro	Descripción
id	Integer Ejemplo: 1 Envía el id del empleado que fue creado en la base de datos.
cedula	String Ejemplo: "1002003001" Envía el número de cedula del empleado a registrar en la tabla empleado, este valor es único.
nombres	String Ejemplo: "Juan José" Envía los nombres para el empleado
apellidos	String Ejemplo: "Flores Aramburu" Envía los apellidos para el empleado
direccion	String Ejemplo: "Calle García moreno" Envía la dirección para el empleado
correo	String Ejemplo: "juanjo@gmail.com" Envía el email para el empleado
cargo	String Ejemplo: "Administrador" Elige de un listado precargado del asset Cargo y envía ese texto con el cargo para el empleado
activo	Boolean Ejemplo: true Cuando se crea un empleado nuevo se elige inicialmente de forma automática true y se envía el estado para el empleado.

```
//Crear empleados
const crearEmpleado = async (req, res, next) => {
  const { cedula, nombres, apellidos, direccion, correo, cargo } =
    req.body;

  try {
    const result = await pool.query("INSERT INTO empleados (cedula,
nombres, apellidos, direccion, correo, cargo) VALUES($1, $2, $3, $4, $5,
$6) RETURNING * ",
    [cedula, nombres, apellidos, direccion, correo, cargo]);

    res.json(result.rows[0]);

  } catch (error) {
    next(error)
  }
};
```

actualizarEmpleado

Este método nos ayuda a actualizar registros del empleado que está registrado en la base de datos. Para ejecutar este método se debe enviar el ID del departamento y actualizar y los siguientes parámetros en formato JSON.

Parámetro	Descripción
id	Integer Ejemplo: 1 Envía el id del empleado que fue creado en la base de datos.
cedula	String Ejemplo: "1002003002" Envía el número de cedula del empleado a registrar en la tabla empleado, este valor es único.
nombres	String Ejemplo: "Gabriel Gregorio" Envía los nombres para el empleado
apellidos	String Ejemplo: "García Moreno" Envía los apellidos para el empleado
direccion	String Ejemplo: "Calle Flores" Envía la dirección para el empleado
correo	String Ejemplo: "gaga@gmail.com" Envía el email para el empleado
cargo	String Ejemplo: "Gerente" Elige de un listado precargado del asset Cargo y envía ese texto con el cargo para el empleado

```
//Actualizar empleados
const actualizarEmpleado = async (req, res, next) => {
  try {
    const { id } = req.params;
```

```

    const { cedula, nombres, apellidos, direccion, correo, cargo } =
req.body;
    const result = await pool.query('UPDATE empleados SET cedula=$1,
nombres=$2, apellidos=$3, direccion=$4, correo=$5, cargo=$6 WHERE id=$7
RETURNING *',
    [cedula, nombres, apellidos, direccion, correo, cargo, id]);
    if (result.rows.length === 0) return res.status(404).json({
message: "Empleado no encontrado" });

    return res.json(result.rows[0])
  } catch (error) {
    next(error)
  }
};

```

eliminarEmpleado

Este método nos ayuda a eliminar registros de empleado en la base de datos. Para ejecutar este método se debe enviar el ID del departamento.

Parámetro	Descripción
Id	Integer Ejemplo: 1 Envía el ID del empleado que fue creado en la base de datos

```

//Eliminar empleados
const eliminarEmpleado = async (req, res, next) => {
  try {
    const { id } = req.params
    const result = await pool.query('DELETE FROM empleados WHERE
id=$1', [id])
    if (result.rowCount === 0) return res.status(404).json({ message:
"Empleado no encontrado" });
    return res.status(204);
  } catch (error) {
    next(error)
  }
};

```

actualizarEstado

Este método se ocupara para cambiar el estado(true o false) del empleado, si el estado esa en Activo, lo cambia a Inactivo y viceversa.

```
//Cambia el estado del empleado (true o false)
const actualizarEstado = async (req, res, next) => {
  try {
    const { id } = req.params;
    const result = await pool.query('UPDATE empleados SET activo=NOT
activo WHERE id=$1 ', [id]);
    if (result.rows.length === 0) {
      return res.status(404).json({ message: "Empleado no
encontrado" });
    }
    return res.json(result.rows[0]);
  } catch (error) {
    next(error)
  }
};
```

Una vez creados los métodos, exportamos para poder utilizarlos en las rutas.

```
module.exports = {
  getEmpleados,
  getEmpleado,
  crearEmpleado,
  eliminarEmpleado,
  actualizarEmpleado,
  actualizarEstado,
}
```

Routes

En esta carpeta se encuentra el archivo routes.js donde contiene las rutas para realizar las peticiones.

```
const { Router } = require('express');
const { getEmpleados, getEmpleado, crearEmpleado, eliminarEmpleado,
actualizarEmpleado, actualizarEstado } =
require('../controllers/controller.js');
const router = Router();

//RUTAS
//Empleados
//Obtener empleados
router.get('/empleados', getEmpleados);

router.get('/empleados/:id', getEmpleado);
```

```
//Crear nuevo empleado
router.post('/empleados', crearEmpleado);

//Eliminar empleado
router.delete('/empleados/:id', eliminarEmpleado);

//Actualizar empleado
router.put('/empleados/:id', actualizarEmpleado);
router.put('/empleados/estado/:id', actualizarEstado); //Para cambiar de
activo a inactivo

module.exports = router;
```

Index.js

En este archivo estará la configuración para inicializar el api. Aquí estará configurado para que el proyecto use las dependencias de Morgan, Express y cors.

```
app.use(morgan('dev'))
app.use(express.json());
app.use(cors());
```

También se llamará a las rutas que utilizará para realizar las peticiones

```
app.use(rutas)
app.use((err, req, res, next) => {
  return res.json({ message: err.message})
})
```

También estará puesto en cuál puerto va a correr nuestra API.

```
app.listen(4000)
console.log('Servidor corriendo en el puerto: 4000')
```

Db.js

En este archivo se tendrá la conexión a la base de datos PostgreSQL con las variables asignadas en el archivo config.js

```
const { password } = require("pg/lib/defaults")
const {db} = require('./config')
const Pool = require("pg").Pool

const pool = new Pool({
  user: db.user,
  password: db.password,
  host: db.host,
  port: db.port,
  database: db.database
```

```
});
```

```
module.exports = pool;
```

Config.js

En este archivo se asignará los parámetros para la conexión a la base de datos con variables de entorno.

```
const {config} = require('dotenv')
config()

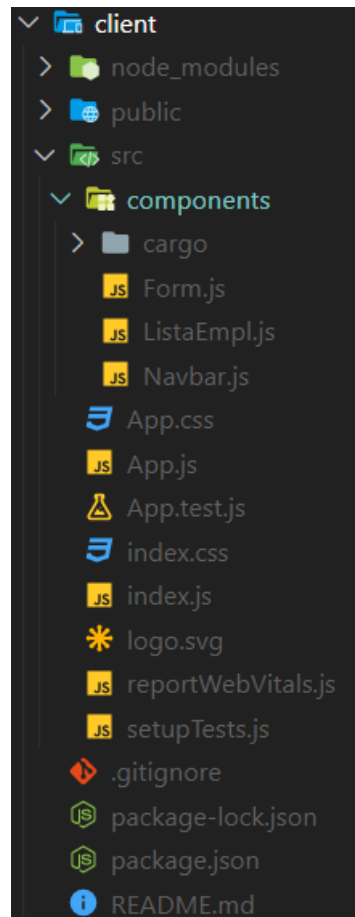
module.exports = {
  db:{
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
    database: process.env.DB_DATABASE
  }
}
```


FrontEnd

El FrontEnd fue realizado con tecnología React. Este FrontEnd cuenta con una página de inicio, página para listar inventario, crear o modificar departamento.

Estructura

La estructura del proyecto del FrontEnd es la siguiente.



Dentro de la carpeta components se encuentra el código de los formularios para cada una de las vistas.

Navbar.js

Aquí se encuentra el código para la barra de navegación que se mostrará en el proyecto.

```
import { AppBar, Box, Button, Container, Toolbar, Typography } from '@mui/material'
import { Link, useNavigate } from 'react-router-dom'
import AttributionIcon from '@mui/icons-material/Attribution';

export default function Navbar() {
  const navigate = useNavigate()

  return (
    <Box sx={{ flexGrow: 1 }}>
      <AppBar position='static' color='transparent' >
        <Container>
          <Toolbar>
            <Typography variant='h6' sx={{ flexGrow: 1 }}>
              <Link to="/" style={{ textDecoration: 'none', color: '#eee' }}>
                PERN
              </Link>
            </Typography>

            &nbsp;
            &nbsp;
            <Button variant='contained' color='warning' onClick={() => navigate("/cargo")}>
              <AttributionIcon />
              &nbsp;Cargo
            </Button>
          </Toolbar>
        </Container>
      </AppBar>
    </Box>
  )
}
```

ListaEmpl.js

Aquí se encuentra el código para listar los registros de empleados.

Se creará un estado para los datos de empleado

```
const [empleados, setEmpleados] = useState([]);
```

Se creó la constante cargarEmpleados para realizar la petición GET de la API creada anteriormente.

```
const cargarEmpleados = async () => {
  const response = await fetch("http://localhost:4000/empleados");
  const data = await response.json();
  setEmpleados(data);
};
```

Se creó la constante eliminarEmpleado para realizar la petición DELETE de la API.

```
const eliminarEmpleado = async (id) => {
  try {
    await fetch(`http://localhost:4000/empleados/${id}`, {
```

```

        method: "DELETE",
      });
      setEmpleados(empleados.filter((empleado) => empleado.id !==
id));

    } catch (error) {
      console.error(error);
    }
  }
};

```

Se tiene una constante estadoEmpleado, que se encarga de cambiar el estado del empleado (true o false):

```

const estadoEmpleado = async (id) => {
  try {
    await fetch(`http://localhost:4000/empleados/estado/${id}`, {
      method: "PUT",
    });
  } catch (error) {
    console.error(error);
  }
};

```

Con useEffect se podrá hacer que el componente cargue el empleado.

```

useEffect(() => {
  cargarEmpleados();
}, []);

```

El código de la vista quedaría de la siguiente manera dentro del return.

```

return (
  <>
    <h1>LISTA DE EMPLEADOS</h1>

    <Button variant='contained' color='success' onClick={() =>
navigate("/empleado/new")}>
      <AddReactionIcon />
      &nbsp; Nuevo empleado
    </Button>
    <TableContainer component={Paper} sx={{ maxHeight: '550PX' }}
style={{ backgroundColor: '#37474f' }} >
      <Table sx={{ minWidth: 650 }} aria-label='simple table'
stickyHeader >

```

```

        <TableHead >
            <TableRow >

                <TableCell align='center' sx={{ fontWeight:
'bold' }} >CEDULA</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>NOMBRES</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>APELLIDOS</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>DIRECCION</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>CORREO</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>CARGO</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>ESTADO</TableCell>
                <TableCell align='center' sx={{ fontWeight:
'bold' }}>OPCIONES</TableCell>

            </TableRow>
        </TableHead>
        <TableBody >
            {
                empleados.map((empleado) => (
                    <TableRow key={empleado.id} sx={{
'&:last-child td, &:last-child th': { border: 0 } }}>

                        <TableCell align='center' style={{
color: 'white' }}>{empleado.cedula}</TableCell>
                        <TableCell align='center' style={{
color: 'white' }}>{empleado.nombres}</TableCell>
                        <TableCell align='center' style={{
color: 'white' }}>{empleado.apellidos}</TableCell>
                        <TableCell align='center' style={{
color: 'white' }}>{empleado.direccion}</TableCell>
                        <TableCell align='center' style={{
color: 'white' }}>{empleado.correo}</TableCell>
                        <TableCell align='center' style={{
color: 'white' }}>{empleado.cargo}</TableCell>
                        <TableCell align='center' style={{
color: 'white' }}>{empleado.activo ? 'Activo' : 'Inactivo'}</TableCell>
                        <TableCell align='center'>

                            <Button variant='contained'
color='warning' onClick={() => estadoEmpleado(empleado.id) &&
window.location.reload()} style={{ marginLeft: ".5rem" }}>

```

```

                                {empleado.activo ?
<CheckBoxIcon style={{ color: 'white' }} /> : <CheckBoxOutlineBlankIcon
style={{ color: 'white' }} />}

                                </Button>
                                &nbsp; &nbsp; &nbsp;

                                {empleado.activo ?
                                <Button variant='contained'
color='primary' onClick={() =>
navigate(`/empleados/${empleado.id}/edit`) } >
                                <EditIcon style={{ color:
'white' }} />
                                </Button>
                                :
                                <Button variant='contained'
disabled='true' color='primary' onClick={() =>
navigate(`/empleados/${empleado.id}/edit`) } >
                                <EditIcon style={{ color:
'white' }} />
                                </Button>
                                }

                                <Button variant='contained'
color='error' onClick={() => eliminarEmpleado(empleado.id) &&
window.location.reload()} style={{ marginLeft: ".5rem" }}>
                                <DeleteIcon style={{ color:
'white' }} />
                                </Button>

                                </TableCell>
                                </TableRow>
                                ))
                                }
                                </TableBody>
                                </Table>
                                </TableContainer>

                                </>
);

```

Form.js

Aquí se tendrá el código para crear o actualizar el registro de empleado.

Primero se creará el estado para los datos de empleado y se pondrá con valores en blanco.

```
const [empleado, setEmpleado] = useState({
  cedula: "",
  nombres: "",
  apellidos: "",
  direccion: "",
  correo: "",
  cargo: ""
});
```

Se creará un estado para saber si se va a editar o a crear el empleado.

```
const [editing, setEditing] = useState(false)
```

En handleChange se ocupará para guardar los inputs apenas se cambie o se agregue valor.

```
const handleChange = (e) => setEmpleado({ ...empleado, [e.target.name]:
e.target.value });
```

En handleSubmit se realizará las peticiones sea POST o PUT dependiendo del caso a realizar.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true)

  if (editing) {
    await fetch(`http://localhost:4000/empleados/${params.id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(empleado)
    });

    Swal.fire({
      position: 'bottom-end',
      icon: 'success',
      title: 'Empleado actualizado correctamente',
      showConfirmButton: false,
      timer: 1200
    })
  } else {
    await fetch('http://localhost:4000/empleados', {
      method: 'POST',
      body: JSON.stringify(empleado),
```

```

        headers: { 'Content-Type': 'application/json' }
      })

    }
    setLoading(false)
    navigate('/')
  };

```

También se tiene la constante `cargarEmpleado` para cargar un solo registro del empleado. Aquí se realizará la petición GET, pero al tratarse de consultar un registro se enviará el parámetro ID del registro del empleado a cargar. Esto servirá para llenar los inputs al momento de actualizar el empleado

```

const cargarEmpleado = async (id) => {
  const res = await fetch(`http://localhost:4000/empleados/${id}`)
  const data = await res.json();
  setEmpleado({ cedula: data.cedula, nombres: data.nombres, apellidos:
data.apellidos, direccion: data.direccion, correo: data.correo, cargo:
data.cargo });
  setEditing(true)
};

```

En sí, la vista quedaría de la siguiente manera dentro del return.

```

return (
  <Grid container alignItems='center' justifyContent='center'>
    <Grid item xs={5} alignContent='center'>
      <Card sx={{ mt: 5 }} style={{ backgroundColor: '#1e272e',
padding: '2rem' }}>
        <Typography variant='h5' textAlign='center' color='white'>
          {editing ? "Modificar Empleado" : "Registrar nuevo empleado"}
        </Typography>
        <CardContent>
          <form onSubmit={handleSubmit}>
            <TextField fullWidth variant='filled' label='Nro de cedula'
sx={{ display: 'block', margin: '.5rem 0' }}
              name="cedula" value={empleado.cedula}
              onChange={handleChange}
              inputProps={{ style: { color: 'white' } }}
              InputLabelProps={{ style: { color: 'white' } }} />

            <TextField fullWidth variant='filled' label='Nombres' sx={{
display: 'block', margin: '.5rem 0' }}
              name="nombres" value={empleado.nombres}
              onChange={handleChange}
              inputProps={{ style: { color: 'white' } }}
              InputLabelProps={{ style: { color: 'white' } }} />
          </form>
        </CardContent>
      </Card>
    </Grid item>
  </Grid>
)

```

```

        <TextField fullWidth variant='filled' label='Apellidos'
sx={{ display: 'block', margin: '.5rem 0' }}
        name="apellidos" value={empleado.apellidos}
onChange={handleChange}
        inputProps={{ style: { color: 'white' } }}
InputLabelProps={{ style: { color: 'white' } }} />

        <TextField fullWidth variant='filled' label='Direccion'
sx={{ display: 'block', margin: '.5rem 0' }}
        name="direccion" value={empleado.direccion}
onChange={handleChange}
        inputProps={{ style: { color: 'white' } }}
InputLabelProps={{ style: { color: 'white' } }} />

        <TextField fullWidth variant='filled' type={'email'}
label='Correo' sx={{ display: 'block', margin: '.5rem 0' }}
        name="correo" value={empleado.correo}
onChange={handleChange}
        inputProps={{ style: { color: 'white' } }}
InputLabelProps={{ style: { color: 'white' } }} />

        <Box sx={{ minWidth: 120 }}>
          <FormControl fullWidth>
            <InputLabel id="demo-simple-select-label" style={{
color: 'white' }}>Cargo</InputLabel>
            <Select
              labelId="demo-simple-select-label"
              id="demo-simple-select"
              style={{ color: 'white' }}
              name="cargo"
              value={empleado.cargo}
              label="Cargo"
              onChange={handleChange}
            >
              {cargo.map((cargo) => (
                <MenuItem key={cargo.id} value={cargo.tipocargo}>
                  {cargo.tipocargo}
                </MenuItem>
              ))}
            </Select>
          </FormControl>
        </Box>

        <br />
        <Button variant='contained' style={{ display: 'block',
margin: '0 auto' }} textAlign='center' color='primary' type='submit'
disabled={

```



```

        !empleado.cedula ||
        !empleado.nombres ||
        !empleado.apellidos ||
        !empleado.direccion ||
        !empleado.correo ||
        !empleado.cargo
      }>
      {loading ? (<CircularProgress color="inherit" size={24}
/>) : (editing ? "Modificar empleado" : "Registrar empleado")}
    </Button>

  </form>
</CardContent>
</Card>
</Grid>
</Grid>
)

```

App.js

En este archivo contendrá el código para inicializar el proyecto del FrontEnd.

Aquí estableceremos las rutas que permitirá la debida navegación dentro del proyecto.

```

import { BrowserRouter, Routes, Route } from 'react-router-dom'
import Form from './components/Form'
import ListaEmpl from './components/ListaEmpl'
import CargoForm from './components/cargo/Cargoform'
import CargoList from './components/cargo/Cargolist'
import Menu from './components/Navbar'
import { Container } from '@mui/material'

export default function App() {
  return (
    <BrowserRouter>
      <Menu />
      <Container>
        <Routes>
          <Route path="/" element={<ListaEmpl />} />
          <Route path="/empleado/new" element={<Form />} />
          <Route path="/empleados/:id/edit" element={<Form />} />
        </Routes>
      </Container>
    </BrowserRouter>
  )
}

```