Asset Departamento

Información General del Proyecto

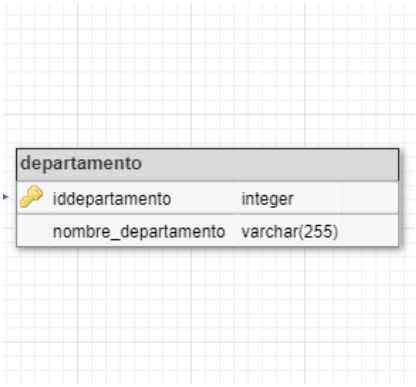
Descripción

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

Prerrequisitos

El proyecto fue realizado con tecnologías PERN Stack (PostreSQL, 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.



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:

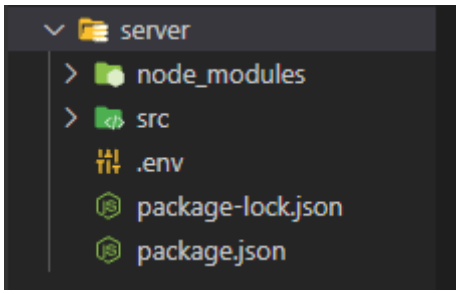
```
npm run dev
```

Para iniciar el frontend se debe ocupar los siguientes comandos:

```
cd client

npm start
```

Backend API Server

El Backed API Server está estructurado de la siguiente manera.



Controller

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

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

getAllDepartamento

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

```
const getAllDepartamento = async (req, res, next) => {
    try {
        const result = await pool.query('select * from departamento')
        res.json(result.rows)
    } catch (error) {
        next(error)
    }

}
```

getOneDepartamento

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

```
const getOneDepartamento = async (req, res, next) => {
    try {
        const { id } = req.params

        const result = await pool.query('select * from departamento where
iddepartamento = $1', [id])

        if (result.rows.length === 0) {
            res.status(400).json({ message: 'No existen datos' })
        }

        res.json(result.rows)
    } catch (error) {
        next(error)
    }
```

```
}
```

createDepartamento

Este método nos ayuda a crear registros de departamento 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 departamento que fue creado en la base de datos
nombre_departamento	String Ejemplo: Gerencia Envía el nombre del departamento a registrar en la base de datos

```
const createDepartamento = async (req, res, next) => {
    const { nombre_departamento } = req.body
    try {
            const result = await pool.query("INSERT INTO departamento
(nombre_departamento) VALUES ($1)", [nombre_departamento])
        res.json(result.rows)
    } catch (error) {
        next(error)
    }

}
```

updateDepartamento

Este método nos ayuda a actualizar registros de departamento 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 departamento que fue creado en la base de datos
nombre_departamento	String Ejemplo: Gerencia Envía el nombre del departamento a registrar en la base de datos

```
const updateDepartamento = async (req, res, next) => {
    try {
        const { id } = req.params
        const { nombre_departamento } = req.body
            const result = await pool.query("UPDATE departamento SET
nombre_departamento = $2 WHERE iddepartamento = $1 RETURNING *", [id,
nombre_departamento])

        if (result.rows.length === 0) {
```

```
            res.status(404).json({ message: 'No existen datos' })
        }
        res.status(200).json({ message: 'Actualizado' })
        //res.json(result.rows)
    } catch (error) {
        next(error)
    }
}
```

deleteDepartamento

Este método nos ayuda a eliminar registros de departamento 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 departamento que fue creado en la base de datos

```
const deleteDepartamento = async (req, res, next) => {
    try {
        const { id } = req.params

         const result = await pool.query('delete from departamento where
idDepartamento = $1 RETURNING *', [id])

        if (result.rows.length === 0) {
            res.status(400).json({ message: 'No existen datos' })
        }

        res.status(204)
    } catch (error) {
        next(error)
    }

}
```

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

```
module.exports = {
    getAllDepartamento,
    getOneDepartamento,
    createDepartamento,
    updateDepartamento,
    deleteDepartamento
}
```

Routes

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

```
const router = require("express").Router();
const pool = require("../db");
const    {getAllDepartamento,    getOneDepartamento,    createDepartamento,
deleteDepartamento,                updateDepartamento}                =
require('../controllers/departamento.controller')

router.get('/departamentos', getAllDepartamento)

router.get('/departamentos/:id', getOneDepartamento)

router.post('/departamentos', createDepartamento)

router.put('/departamentos/:id', updateDepartamento)

router.delete('/departamentos/:id', deleteDepartamento)


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

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

```
app.listen(5000, () =>{
    console.log("server is running in port 5000")
});
```

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.

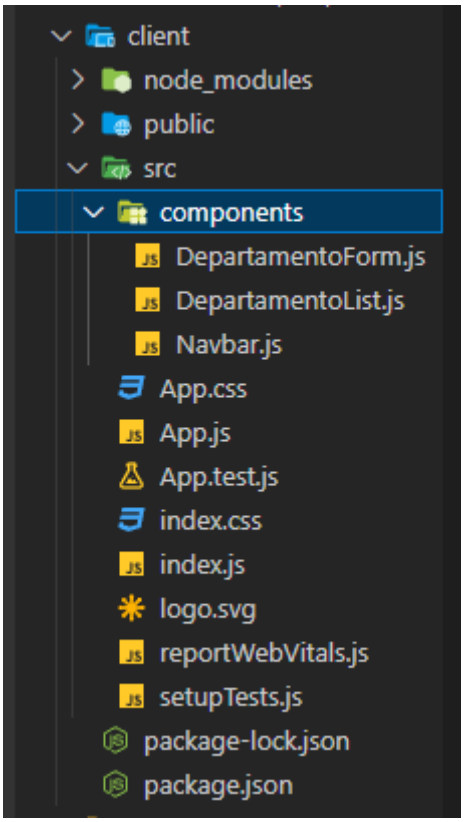
```
const {config} = require('dotenv')
config()

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
    }
}
```

Front end

El front end 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 front end 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.

```jsx
import * as React from "react";
import AppBar from "@mui/material/AppBar";
import Box from "@mui/material/Box";
import Toolbar from "@mui/material/Toolbar";
import Typography from "@mui/material/Typography";
import Button from "@mui/material/Button";
import { Container } from "@mui/material";

import { useNavigate, Link } from "react-router-dom";

export default function Navbar() {
    const navigate = useNavigate();
    return (
        <Box sx={{ flexGrow: 1 }}>
            <AppBar position="static" color="transparent">
                <Container>
                    <Toolbar>
                        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
                            <Link to="/" style={{ textDecoration: "none", color: '#eeeeee' }}>
                                Inicio
                            </Link>
                        </Typography>
                        <Button
                            variant="contained"
                            color="secondary"
                            onClick={() => navigate("/departamento")}
                            style={{marginLeft: ".5rem"}}
                        >
                            Departamentos
                        </Button>

                    </Toolbar>
                </Container>
            </AppBar>
        </Box>
    )
}

```

DepartamentoList.js

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

Se creará un estado para los datos de inventario

```jsx
const [departamento, setDepartamento] = useState([])
```

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

```jsx
  const loadDepartamento = async () => {

  const response = await fetch('http://localhost:5000/departamentos')
  const data = await response.json()
  setDepartamento(data)
}
```

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

```jsx
const handleDelete = async (iddepartamento) => {
```

```
    await fetch(`http://localhost:5000/departamentos/${iddepartamento}`,
{
      method: "DELETE",
    });

    setDepartamento(
      departamento.filter((departamento) => departamento.iddepartamento
!== iddepartamento));
    window.location.reload(true)
  }
```

Con useEffect se podrá hacer que el componente cargue.

```
useEffect(() => {
    loadDepartamento()
  }, [])
```

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

```
 return (

    <>
      <Button
        variant="contained"
        color="primary"
        sx={{ display: 'block', margin: '.5rem 0' }}
        onClick={() => navigate("/departamento/new")}
      >
        Crear Departamento
      </Button>
      <h1>Lista de Departamentos</h1>
      {
        departamento.map((departamento) => (
          <Card style={{
            marginBottom: ".7rem",
            backgroundColor: '#1e272e'
          }}
            key={departamento.iddepartamento}>
            <CardContent style={{
              display: "flex",
              justifyContent: "space-between"
            }}>
              <div style={{ color: "white" }}>
                <Typography
>{departamento.nombre_departamento}</Typography>
              </div>
              <div>
                <Button variant="contained" color="inherit" onClick={()
=>
```

```
navigate(`/departamento/${departamento.iddepartamento}/edit`)}>Actualizar
</Button>


                <Button variant="contained" color="warning"
                  onClick={() =>
handleDelete(departamento.iddepartamento)}
                  style={{ marginLeft: ".5rem" }}
                >Eliminar
                </Button>
              </div>
            </CardContent>
          </Card>
        ))
      }
    </>
  )
}
```

DepartamentoForm.js

Aquí se tendrá el codigo para crear o actualizar el registro de departamento.

Primero se creará el estado para los datos de departamento y se pondrá con valores nulos.

```
const [departamento, setDepartamento] = useState({
    nombre_departamento: ''
  })
```

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

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

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

```
const handleChange = (e) => {
    setDepartamento({ ...departamento, [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();

    if (editing) {
      await fetch(`http://localhost:5000/departamentos/${params.id}`, {
        method: 'PUT',
        body: JSON.stringify(departamento),
```

```
        headers: { "Content-Type": "application/json" }
      })

    }else{
    await fetch('http://localhost:5000/departamentos', {
      method: 'POST',
      body: JSON.stringify(departamento),
      headers: { "Content-Type": "application/json" }
    })
  }
```

También se tendrá loadOnaDepartamento para cargar un solo registro del departamento. Aquí se realizará la petición GET pero al tratarse de consultar un registro se enviará el parámetro ID del registro del departamento a cargar.

```
const loadOneDepartamento = async (iddepartamento) => {
    const res = await
fetch(`http://localhost:5000/departamentos/${params.id}`)
    const data = await res.json()
    console.log(data)
    setDepartamento({ nombre_departamento: data.nombre_departamento })
    setEditing(true)
  }
```

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

```
return (
    <Grid container direction='column' alignItems='center'
justifyContent='center'>
      <Grid item xs={2}>
        <Card sx={{ mt: 10 }} style={{
          background: "#1e272e",
          padding: "1rem"
        }}>
          <Typography variant="h5" textAlign="center" color="white">
            {editing ? "Editar Departamento" : "Crear Departamento"}
          </Typography>
          <CardContent>
            <form onSubmit={handleSubmit}>
              <TextField
                variant="filled"
                placeholder="Nombre Departamento"
                label="Nombre del Departamento"
                sx={{ display: 'block', margin: '.5rem 0' }}
                name="nombre_departamento"
                value={departamento.nombre_departamento || ""}
                onChange={handleChange}
                inputProps={{ style: { color: "white" } }}
```

```
                InputLabelProps={{ style: { color: "white" } }}></TextField>
              <Button variant="contained" color="primary" type="submit" disabled={!departamento.nombre_departamento}>
                Crear Departamento
              </Button>
            </form>
          </CardContent>
        </Card>
      </Grid>
    </Grid>
  )
}
```

App.js

En este archivo contendrá el código para inicializar el proyecto del front end.

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

```
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import DepartamentoList from './components/DepartamentoList'
import DepartamentoForm from './components/DepartamentoForm'
import Navbar from './components/Navbar'
import { Container } from '@mui/material'
export default function App() {
  return (
    <BrowserRouter>
    <Navbar/>
      <Container>
        <Routes>
        <Route path='/' element={<DepartamentoList />} />
          <Route path='/departamento' element={<DepartamentoList />} />
          <Route path='/departamento/new' element={<DepartamentoForm />} />
          <Route path='/departamento/:id/edit' element={<DepartamentoForm />}/>
        </Routes>
      </Container>
    </BrowserRouter>
  )
}
```