Asset Productos

Información General del Proyecto

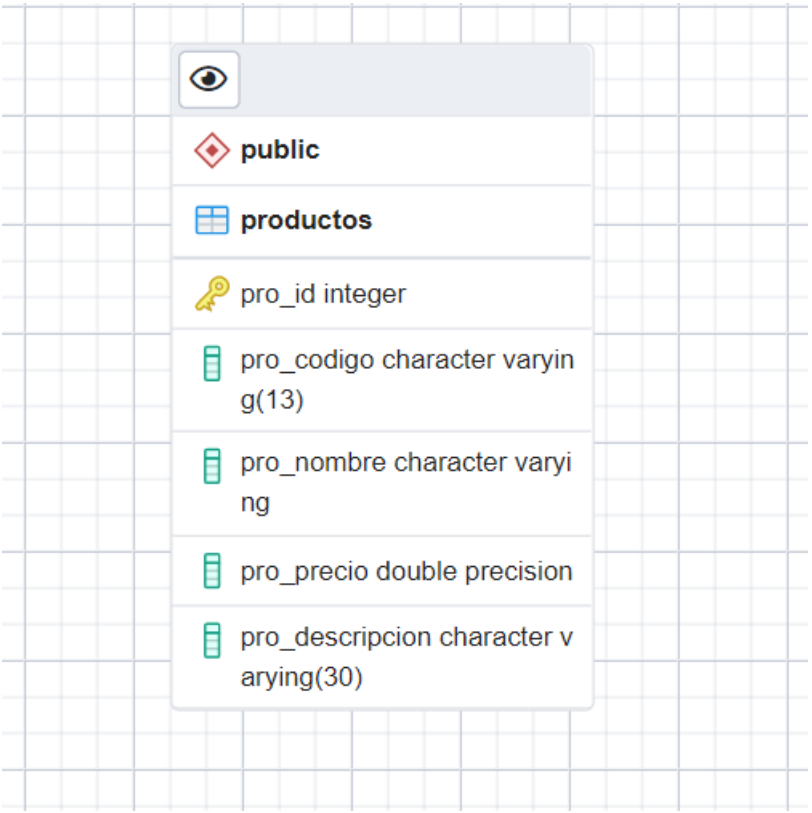
Descripción

El proyecto permite ingresar el producto 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:

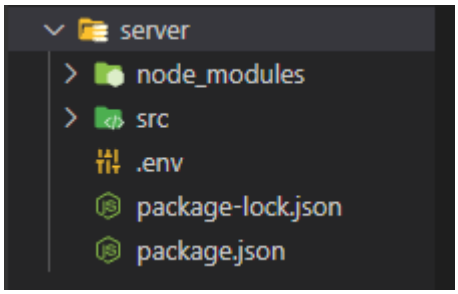
```
npm run dev
```

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 productos.controller.js que contiene los controladores para realizar las rutas.

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

getAllProductos

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

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

}
```

getOneProductos

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

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

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

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

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

```
}
```

createProductos

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

Parámetro	**Descripción**
pro_id	Integer Ejemplo: 1 Envía el ID del producto que fue creado en la base de datos
pro_codigo	String Ejemplo: PC-Escritorio Envía el código del activo a registrar en la tabla productos
pro_nombre	String Ejemplo: 2 Envía el nombre que tienen del activo a registrar.
pro_precio	Double Ejemplo: 1 Envía la cantidad económica del producto que está registrado en la tabla de productos.
pro_descripcion	String Ejemplo: Se encuentra en buen estado Envía las observaciones encontradas del activo para registrarlo en la base de datos.

```
const createProductos = async (req, res, next) => {
    const { pro_codigo, pro_nombre, pro_precio, pro_descripcion } =
req.body

    try {
        const result = await pool.query("INSERT INTO productos
(pro_codigo, pro_nombre, pro_precio, pro_descripcion) VALUES ($1, $2, $3,
$4)", [pro_codigo, pro_nombre, pro_precio, pro_descripcion])
        res.status(200).json(result.rows)
    } catch (error) {
        next(error)
    }

}
```

updateProductos

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

Parámetro	Descripción
pro_id	Integer Ejemplo: 1 Envía el ID del producto que fue creado en la base de datos
pro_codigo	String Ejemplo: iPhone 11 Envía el código del activo a registrar en la tabla productos
pro_nombre	String Ejemplo: 2 Envía el nombre que tienen del activo a registrar.
pro_precio	Double Ejemplo: 1 Envía la cantidad económica del producto que está registrado en la tabla de productos.
pro_descripcion	String Ejemplo: Se encuentra en buen estado Envía las observaciones encontradas del activo para registrarlo en la base de datos.

```
const updateProductos = async (req, res, next) => {
    try {
        const { id } = req.params
        const { pro_codigo, pro_nombre, pro_precio, pro_descripcion } =
req.body
        const result = await pool.query("UPDATE productos SET pro_codigo
= $2, pro_nombre = $3, pro_precio =$4, pro_descripcion = $5, WHERE
pro_id  = $1 RETURNING *", [id, pro_codigo, pro_nombre, pro_precio,
pro_descripcion])

        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)
    }
}
```

deleteProductos

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

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

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

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

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

        res.status(200)
    } catch (error) {
        next(error)
    }

}
```

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

```
module.exports = {
    getAllProductos,
    getOneProductos,
    createProductos,
    updateProductos,
    deleteProductos
}
```

Routes

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

```
const router = require("express").Router();
const pool = require("../db");
const { createProductos, deleteProductos, getAllProductos,
getOneProductos, updateProductos } =
require('../controllers/productos.controller')

router.get('/productos', getAllProductos)

router.get('/productos/:id', getOneProductos)

router.post('/productos', createProductos)

router.put('/productos/:id', updateProductos)

router.delete('/productos/:id', deleteProductos)


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(3000, () =>{
    console.log("server is running in port 3000")
});
```

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.

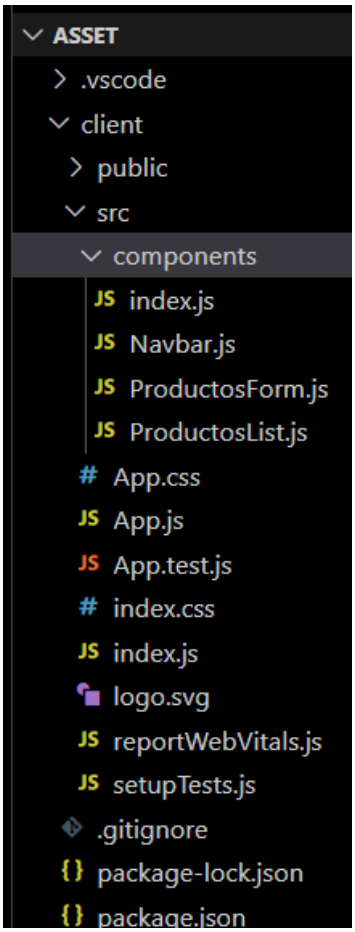
```
const {config} = require('dotenv')
config()

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 productos, crear, modificar o eliminar productos.

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.

ProductosList.js

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

Se creará un estado para los datos de productos

```
const [productos, setProductos] = useState([])
```

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

```
    const loadProductos = async () => {

    const response = await fetch('http://localhost:3000/productos)
    const data = await response.json()
    setProductos(data)
  }
```

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

```
  const handleDelete = async (pro_id) => {

    await fetch(`http://localhost:3000/productos/${pro_id}`, {
      method: "DELETE",
    })
    setProductos(productos.filter((productos) => productos.pro_id !==
pro_id));
  }
```

Con useEffect se podrá hacer que el componente cargue.

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

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

```
return (
    <>
      <h1>Lista Productos </h1>

      <div>
      <SearchIcon></SearchIcon>
      <input value={busqueda} onChange={handleChange}
      placeholder='Search name product'
      className='form-control inputBuscar'>
      </input>
      </div><br></br>
      <TableContainer component={Paper}>
        <Table sx={{ minWidth: 700 }} aria-label="customized table">
          <TableHead>
            <TableRow>
              <StyledTableCell align="right">ID</StyledTableCell>
              <StyledTableCell align="right">Código</StyledTableCell>
              <StyledTableCell align="right">Nombre</StyledTableCell>
              <StyledTableCell align="right">Precio</StyledTableCell>
              <StyledTableCell
align="right">Descripción</StyledTableCell>
              <StyledTableCell align="center">Opciones</StyledTableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {productos.map((productos) => (
              <StyledTableRow key={productos.pro_id}>

                <StyledTableCell
align="right">{productos.pro_id}</StyledTableCell>
                <StyledTableCell
align="right">{productos.pro_codigo}</StyledTableCell>
                <StyledTableCell
align="right">{productos.pro_nombre}</StyledTableCell>
                <StyledTableCell
align="right">{productos.pro_precio}</StyledTableCell>
                <StyledTableCell
align="right">{productos.pro_descripcion}</StyledTableCell>
                <StyledTableCell align="right">
```

```
                  <Button variant='contained' color='inherit' onClick={()
=> navigate(`/productos/${productos.pro_id}/edit`)}>
                    Actualizar
                  </Button>

                  <Button variant='contained' color='warning' onClick={()
=> handleDelete(productos.pro_id)} style={{ marginLeft: ".5rem" }}>
                    Eliminar
                  </Button>

                </StyledTableCell>
              </StyledTableRow>
            ))}
          </TableBody>
        </Table>
      </TableContainer>

    </>
  )
}
```

ProductosForm.js

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

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

```
const [productos, setProductos] = useState({
    pro_id: "",
    pro_codigo: "",
    pro_nombre: "",
    pro_precio: "",
    pro_descripcion: "",
  });
```

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

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

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

```
const handleChange = e =>
    setProductos({ ...productos, [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:3000/productos/${params.pro_id}`, {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(productos),
      });
    } else {
      await fetch("http://localhost:3000/productos", {
        method: "POST",
        body: JSON.stringify(productos),

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

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

```
const loadOneProductos = async (pro_id) => {
    const res = await fetch(`http://localhost:3000/productos/${pro_id}`)
    const data = await res.json()
    setProductos({ pro_id, pro_codigo: data.pro_codigo, pro_nombre:
data.pro_nombre, pro_precio: data.pro_precio, pro_descripcion:
data.pro_descripcion })
    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={3}>
        <Card sx={{ mt: 5 }} style={{ backgroundColor: '#1e272e',
padding: '1rem' }}>
          <Typography variant='5' textAlign='center' color='white'>
            {editing? "Editar Productos" : "Crear Productos"}
          </Typography>
          <CardContent>
```

```
            <form onSubmit={handleSubmit}>
              <TextField
                variant='filled'
                label='Ingrese el código'
                sx={{ display: 'block', margin: '.5rem 0' }}
                name="pro_codigo"
                value={productos.pro_codigo}
                onChange={handleChange}
                inputProps={{ style: { color: "white" } }}
                InputLabelProps={{ style: { color: "white" } }}
              />
              <TextField
                variant='filled'
                label='Ingrese el nombre'
                sx={{ display: 'block', margin: '.5rem 0' }}
                name="pro_nombre"
                value={productos.pro_nombre}
                onChange={handleChange}
                inputProps={{ style: { color: "white" } }}
                InputLabelProps={{ style: { color: "white" } }}
              />
              <TextField
                variant='filled'
                label='Ingrese el precio'
                sx={{ display: 'block', margin: '.5rem 0' }}
                name="pro_precio"
                value={productos.pro_precio}
                onChange={handleChange}
                inputProps={{ style: { color: "white" } }}
                InputLabelProps={{ style: { color: "white" } }}
              />
              <TextField
                variant='filled'
                label='Ingrese la descripción'
                multiline
                rows={3}
                sx={{ display: 'block', margin: '.5rem 0' }}
                name="pro_descripcion"
                value={productos.pro_descripcion}
                onChange={handleChange}
                inputProps={{ style: { color: "white" } }}
                InputLabelProps={{ style: { color: "white" } }}
              />
              <Button
                variant='contained'
                color='primary'
                type='submit'
                disabled={!productos.pro_codigo || !productos.pro_nombre
|| !productos.pro_precio ||
```

```
                  !productos.pro_descripcion}>
                Guardar
              </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, Route, Routes } from 'react-router-dom';
import ListaProductos from './components/ProductosList';
import FormProductos from './components/ProductosForm';
import Menu from './components/Navbar';
import { Container } from '@mui/material'

export default function App() {
  return (
    <BrowserRouter>
     <Menu/>
      <Container>
        <Routes>
          <Route path="/" element={<ListaProductos />} />
          <Route path="/productos/nuevo" element={<FormProductos />} />
          <Route path="/productos/:pro_id/edit"
element={<FormProductos/>}/>
        </Routes>
        </Container>
    </BrowserRouter>
  );
}
```