Asset Inventario

Información General del Proyecto

Descripción

El proyecto permite ingresar el inventario que se tiene en cualquier establecimiento junto con su departamento.

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:

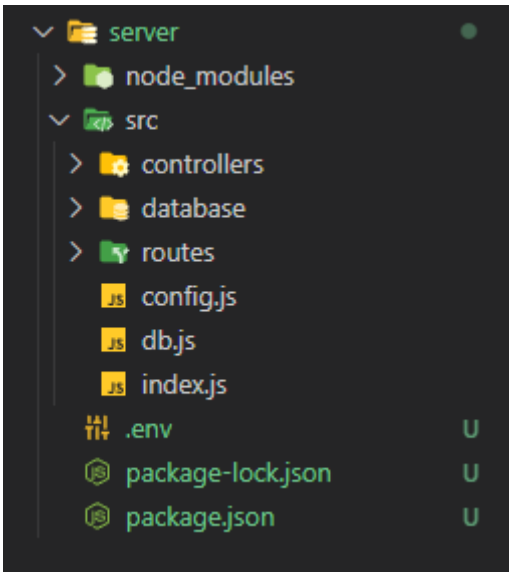
```
npm run dev
```

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

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

getAllInventario

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

```
const getAllInventario = async (req, res, next) => {
    try {
        const result = await pool.query(
            'select descripcion, cantidad, d.nombre_departamento, fecha_registro, observaciones from inventario as i inner join departamento a
            )
        res.status(200).json(result.rows)
    } catch (error) {
        next(error)
    }

}
```

getOneInventario

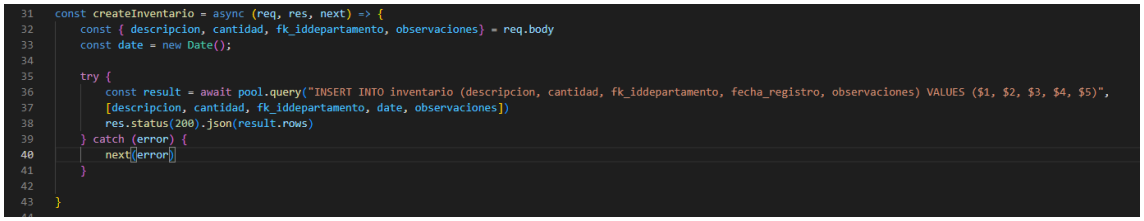
Este método nos ayuda a obtener un solo registro de la tabla de inventario. Para realizar este método se debe enviar el parámetro ID del inventario a buscar.

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

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

        if (result.rows.length === 0) {
            res.status(404).json({ message: 'No existen datos' })
        }

        res.status(200).json(result.rows)
    } catch (error) {
        next(error)
    }
}
```

createInventario

Este método nos ayuda a crear registros de inventario 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 inventario que fue creado en la base de datos
Descripción	String Ejemplo: PC-Escritorio Envía la descripción del activo a registrar en la tabla inventario
cantidad	Integer Ejemplo: 2 Envía la cantidad que tienen del activo a registrar.
fk_iddepartamento	Integer Ejemplo: 1 Envía el número ID del departamento que está registrado en la tabla de departamento.
observaciones	String Ejemplo: Se encuentra en buen estado Envía las observaciones encontradas del activo para registrarlo en la base de datos.



```
const createInventario = async (req, res, next) => {
    const { descripcion, cantidad, fk_iddepartamento, observaciones} = req.body
    const date = new Date();

    try {
        const result = await pool.query("INSERT INTO inventario (descripcion, cantidad, fk_iddepartamento, fecha_registro, observaciones) VALUES ($1, $2, $3, $4, $5)",
        [descripcion, cantidad, fk_iddepartamento, date, observaciones])
        res.status(200).json(result.rows)
    } catch (error) {
        next(error)
    }

}
```

updateInventario

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

Parámetro	Descripción
Id	Integer Ejemplo: 1 Envía el ID del inventario que fue creado en la base de datos
Descripción	String Ejemplo: PC-Escritorio Envía la descripción del activo a registrar en la tabla inventario
cantidad	Integer Ejemplo: 2 Envía la cantidad que tienen del activo a registrar.
fk_iddepartamento	Integer Ejemplo: 1 Envía el número ID del departamento que está registrado en la tabla de departamento.
observaciones	String Ejemplo: Se encuentra en buen estado Envía las observaciones encontradas del activo para registrarlo en la base de datos.

```
const updateInventario = async (req, res, next) => {
    try {
        const { id } = req.params
        const { descripcion, cantidad, fk_iddepartamento, observaciones}
= req.body
        const date = new Date();
        const result = await pool.query("UPDATE inventario SET
descripcion = $2, cantidad = $3, fk_iddepartamento =$4, fecha_registro =
$5, observaciones = $6 WHERE idinventario = $1 RETURNING *",
        [id, descripcion, cantidad, fk_iddepartamento, date,
observaciones])

        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)
    }
}
```

deleteInventario

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

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

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

        const result = await pool.query('delete from inventario where
idinventario = $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 = {
    getAllInventario,
    getOneInventario,
    createInventario,
    updateInventario,
    deleteInventario
}
```

Routes

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

```
const router = require("express").Router();
const pool = require("../db");
const { createInventario, deleteInventario, getAllInventario,
getOneInventario, updateInventario } =
require('../controllers/inventario.controller')

router.get('/inventario', getAllInventario)

router.get('/inventario/:id', getOneInventario)

router.post('/inventario', createInventario)

router.put('/inventario/:id', updateInventario)

router.delete('/inventario/:id', deleteInventario)


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.

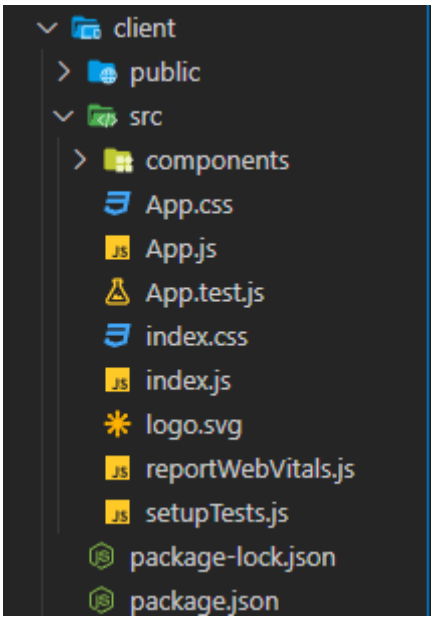
```
const {config} = require('dotenv')
config()

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 inventario.

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.

```
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="primary"
                            onClick={() => navigate("/inventario")}
                        >
                            Inventario
                        </Button>
                        <Button
                            variant="contained"
                            color="secondary"
                            onClick={() => navigate("/departamento")}
                            style={{marginLeft: ".5rem"}}
                        >
                            Departamentos
                        </Button>

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

InventarioList.js

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

Se creará un estado para los datos de inventario

```
const [inventario, setInventario] = useState([])
```

Para listar los registros, se podrá visualizar en una tabla que estará configurada de la siguiente manera.

```
const StyledTableCell = styled(TableCell)(({ theme }) => ({
    [`&.${tableCellClasses.head}`]: {
      backgroundColor: theme.palette.common.black,
      color: theme.palette.common.white,
    },
    [`&.${tableCellClasses.body}`]: {
      fontSize: 14,
    },
  }));
```

```
  const StyledTableRow = styled(TableRow)(({ theme }) => ({
    '&:nth-of-type(odd)': {
      backgroundColor: theme.palette.action.hover,
    },
    // hide last border
    '&:last-child td, &:last-child th': {
      border: 0,
    },
  }));
```

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

```
  const loadInventario = async () => {
    const response = await fetch('http://localhost:5000/inventario')
    const data = await response.json()
    setInventario(data)
  }
```

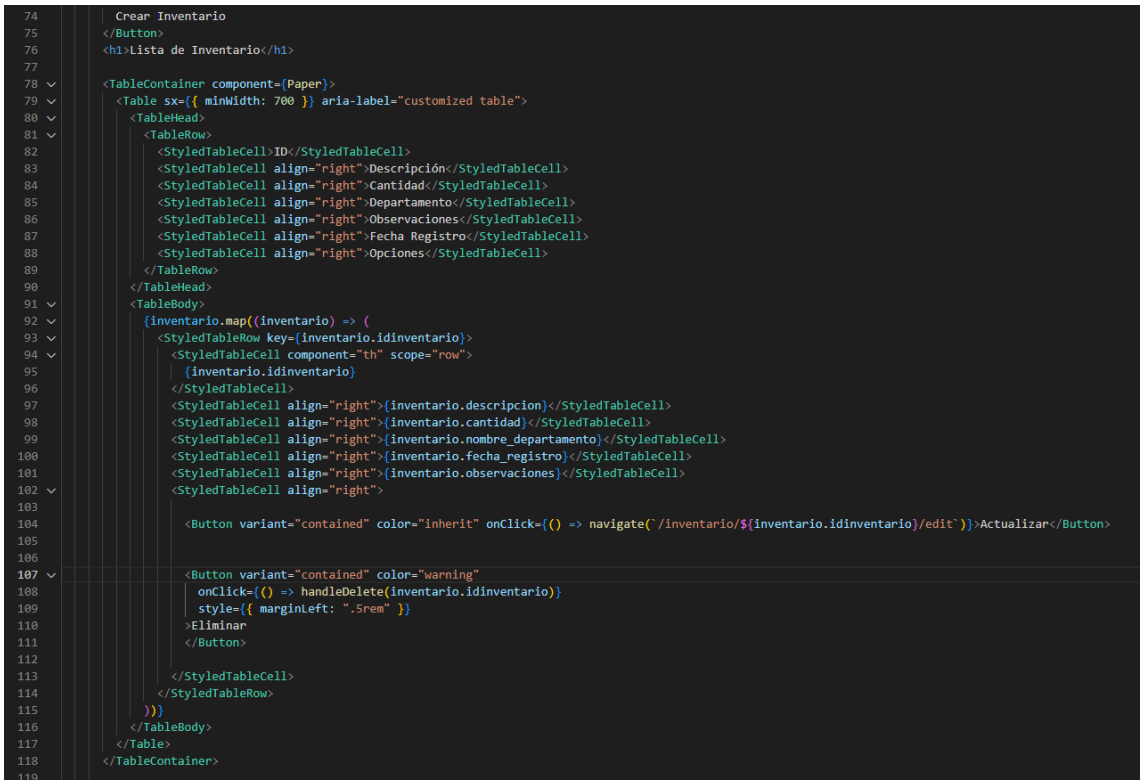
Se creó la constante handleDelete para realizar la petición DELETE de la API.

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

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

    setInventario(
      inventario.filter((inventario) => inventario.idinventario !==
idinventario));
    window.location.reload(true)
  }
```

Con useEffect se podrá hacer que el componente cargue.

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

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



```
          Crear Inventario
        </Button>
        <h1>Lista de Inventario</h1>

        <TableContainer component={Paper}>
          <Table sx={{ minWidth: 700 }} aria-label="customized table">
            <TableHead>
              <TableRow>
                <StyledTableCell>ID</StyledTableCell>
                <StyledTableCell align="right">Descripción</StyledTableCell>
                <StyledTableCell align="right">Cantidad</StyledTableCell>
                <StyledTableCell align="right">Departamento</StyledTableCell>
                <StyledTableCell align="right">Observaciones</StyledTableCell>
                <StyledTableCell align="right">Fecha Registro</StyledTableCell>
                <StyledTableCell align="right">Opciones</StyledTableCell>
              </TableRow>
            </TableHead>
            <TableBody>
              {inventario.map((inventario) => (
                <StyledTableRow key={inventario.idinventario}>
                  <StyledTableCell component="th" scope="row">
                    {inventario.idinventario}
                  </StyledTableCell>
                  <StyledTableCell align="right">{inventario.descripcion}</StyledTableCell>
                  <StyledTableCell align="right">{inventario.cantidad}</StyledTableCell>
                  <StyledTableCell align="right">{inventario.nombre_departamento}</StyledTableCell>
                  <StyledTableCell align="right">{inventario.fecha_registro}</StyledTableCell>
                  <StyledTableCell align="right">{inventario.observaciones}</StyledTableCell>
                  <StyledTableCell align="right">

                    <Button variant="contained" color="inherit" onClick={() => navigate(`/inventario/${inventario.idinventario}/edit`)}>Actualizar</Button>


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

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

InventarioForm.js

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

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

```
  const [inventario, setInventario] = useState({
    descripcion: "",
    cantidad: "",
    fk_iddepartamento: "",
    observaciones: ""
  })
```

También se creará el estado para departamento ya que para crear el inventario se necesita tener el id del departamento a cual pertenece.

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

Con la constante loadDepartamento se realizará la petición GET a la API de departamento.

```
const loadDepartamento = async () => {
    const response = await fetch('http://localhost:5000/departamentos')
    const data2 = await response.json()
    setDepartamento(data2)
  }
```

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

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

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

```
const handleChange = (e) => {
```

```
    setInventario({ ...inventario, [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/inventario/${params.id}`, {
        method: 'PUT',
        body: JSON.stringify(inventario),
        headers: { "Content-Type": "application/json" }
      })

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

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

```
const loadOneInventario = async (idinventario) => {
    const res = await
fetch(`http://localhost:5000/inventario/${params.id}`)
    const data = await res.json()
    console.log(data)
    setInventario({ descripcion: data.descripcion, cantidad:
data.cantidad, observaciones: data.observaciones })
    setEditing(true)
  }
```

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

```jsx
return (
  <Grid
    container
    alignItems="center"
    direction="column"
    justifyContent="center"
  >
    <Grid item xs={4}>
      <Card
        sx={{ mt: 10 }}
        style={{
          backgroundColor: "#1E272E",
          padding: "1rem",
        }}
      >
        <Typography variant="h5" textAlign="center" color="white">
          {editing ? "Editar Inventario" : "Crear Inventario"}
        </Typography>
        <CardContent>
          <form onSubmit={handleSubmit}>
            <TextField
              variant="filled"
              placeholder="Descripción"
              label="Derscripción"
              multiline
              rows={4}
              sx={{
                display: "block",
                margin: ".5rem 0",
              }}
              name="descripcion"
              onChange={handleChange}
              value={inventario.descripcion}
              inputProps={{ style: { color: "white" } }}
              InputLabelProps={{ style: { color: "white" } }}
            />
            <TextField
              variant="outlined"
              label="Cantidad"
              placeholder="Cantidad"
              sx={{
                display: "block",
                margin: ".5rem 0",
              }}
              name="cantidad"
              onChange={handleChange}
              value={inventario.cantidad}
              inputProps={{ style: { color: "white" } }}
              InputLabelProps={{ style: { color: "white" } }}
            />

            <TextField id="outlined-select-currency"
              select
              label="Departamento"
              defaultValue=""
              helperText="Seleccione un departamento"

              name="departamento"
              onChange={handleChange}
              value={inventario.fk_iddepartamento}
              inputProps={{ style: { color: "white" } }}
              InputLabelProps={{ style: { color: "white" } }}>


              {departamento.map((departamento) => {
                <MenuItem key={departamento.iddepartamento} value={departamento.iddepartamento}>{departamento.nombre_departamento}</MenuItem>
              })}

            </TextField>


            <TextField
              variant="filled"
              placeholder="Observaciones"
              label="Observaciones"
              multiline
              rows={4}
              sx={{
                display: "block",
                margin: ".5rem 0",
              }}
              name="observaciones"
              onChange={handleChange}
              value={inventario.observaciones}
              inputProps={{ style: { color: "white" } }}
              InputLabelProps={{ style: { color: "white" } }}
```

```
                rows={4}
                sx={{
                  display: "block",
                  margin: ".5rem 0",
                }}
                name="observaciones"
                onChange={handleChange}
                value={inventario.observaciones}
                inputProps={{ style: { color: "white" } }}
                InputLabelProps={{ style: { color: "white" } }}
              />

            </form>
            <Button
                type="submit"
                variant="contained"
                color="primary"
                disabled={!inventario.descripcion || !inventario.cantidad ||!inventario.fk_iddepartamento || !inventario.observaciones}
              >
                Guardar
              </Button>
          </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 InventarioList from './components/InventarioList'
import InventarioForm from './components/InventarioForm'
import Index from './components/index'
import Navbar from './components/Navbar'
import { Container } from '@mui/material'
export default function App() {
  return (
    <BrowserRouter>
    <Navbar/>
      <Container>
        <Routes>
        <Route path='/' element={<InventarioList />} />
          <Route path='/inventario' element={<InventarioList />} />
          <Route path='/inventario/new' element={<InventarioForm />} />
          <Route path='/inventario/:id/edit' element={<InventarioForm
/>}/>
        </Routes>
      </Container>
    </BrowserRouter>
  )
}
```