

Asset Login

Información General del Proyecto

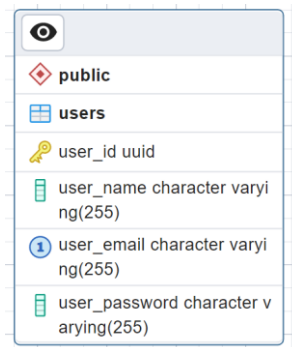
Descripción

El proyecto permite gestionar el acceso de usuarios a través de un login.

Prerrequisitos

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

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



public
users
user_id uuid
user_name character varying(255)
user_email character varying(255)
user_password character varying(255)

Para el BackEnd se ocupó las librerías:

- Express
- Cors
- pg
- jsonwebtoken
- bcrypt

Mientras que para el FrontEnd se ocupó las dependencias:

- React
- React-toastify
- React-router-dom
- reactstrap

Instalación

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

Para iniciar el backend se debe usar el siguiente comando:

```
cd server  
nodemon index
```

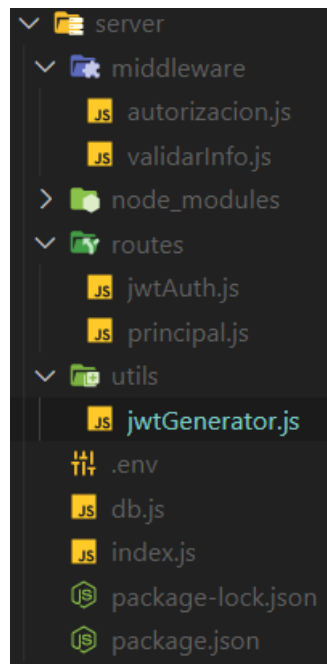
Se uso nodemon para que el servidor se refresque cada vez que se hace un cambio en el código.

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.



Middleware

Dentro de la carpeta middleware se encuentran 2 archivos: autorizacion.js y validarInfo.js

En el archivo autorizacion.js se encuentra el método para la validación del token:

```
1 const jwt = require("jsonwebtoken");  
2 require("dotenv").config();  
3  
4 module.exports = async (req, res, next) => {  
5  
6   //Verificar el Token  
7   try {  
8     const jwtToken = req.header("token");  
9  
10    if (!jwtToken) {  
11      return res.status(403).json("No esta autorizado");  
12    }  
13  
14    const payload = jwt.verify(jwtToken, process.env.jwtSecret);  
15    req.user = payload.user;  
16    next();  
17  
18  } catch (err) {  
19    console.error(err.message);  
20    return res.status(403).json("No esta autorizado");  
21  }  
22 };
```

Este método hace una comprobación del token en el header de la página.

En el archivo validarInfo.js se encuentran método de validación. Validación de una nomenclatura correcta para el correo y también hay un método que valida que los campos requeridos para el login sean correcto y no estén vacíos:

```
1 module.exports = (req, res, next) => {
2   const { email, name, password } = req.body;
3
4   function validEmail(userEmail) {
5     return /^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$/\\.test(userEmail);
6   }
7
8   if (req.path === "/register") {
9     console.log(!email.length);
10    if (![email, name, password].every(Boolean)) {
11      return res.status(401).json("Escriba sus credenciales");
12    } else if (!validEmail(email)) {
13      return res.status(403).json("Email invalido");
14    }
15  }
16  } else if (req.path === "/login") {
17    if (![email, password].every(Boolean)) {
18      return res.status(401).json("Escriba sus credenciales");
19    } else if (!validEmail(email)) {
20      return res.status(403).json("Email invalido");
21    }
22  }
23
24  next();
25 }
```

Rutas

En la carpeta rutas se encuentran 2 archivos: jwtAuth.js y principal.js

En jwtAuth.js podemos encontrar las rutas para registro, login y verificación.

Ruta para el registro:

```
//Ruta para el registro
router.post("/register", validarInfo, async (req, res) => {
  //1 req.body(nombre, email, contraseña)
  const { name, email, password } = req.body;
  try {
    //2 verificar si el usuario existe
    const user = await pool.query("SELECT *FROM users WHERE user_email=$1", [email]);
    if (user.rows.length !== 0) {
      return res.status(401).json("El usuario ya existe");
    }

    //3 Bcrypt para la contraseña del usuario
    const saltRound = 10;
    const salt = await bcrypt.genSalt(saltRound);
    const bcryptPassword = await bcrypt.hash(password, salt);

    //4 Registrar nuevo usuario
    const newUser = await pool.query("INSERT INTO users (user_name, user_email, user_password) VALUES ($1, $2, $3) RETURNING*",
      [name, email, bcryptPassword]);

    //5 Generar el Token JWT
    const token = jwtGenerator(newUser.rows[0].user_id);
    res.json({ token });
  } catch (err) {
    console.error(error.message);
    res.status(500).send("Error en el servidor");
  }
});
```

Ruta para el login:

```
//Ruta para el Login
router.post("/login", validarInfo, async (req, res) => {
  try {
    // 1 req.body
    const { email, password } = req.body;

    // 2 Verificar si el usuario existe, sino dar una alerta
    const user = await pool.query("SELECT *FROM users WHERE user_email =?", [
      email
    ]);

    if (user.rows.length === 0) {
      return res.status(401).json("El email es incorrecto");
    }

    // 3 Verificar si las contraseñas coinciden
    const validPassword = await bcrypt.compare(password, user.rows[0].user_password); //se retornara un boolean T o F
    if (!validPassword) {
      return res.status(401).json("La contraseña es incorrecta");
    }

    //4 Dar el token JWT
    const token = jwtGenerator(user.rows[0].user_id);
    res.json({ token });
  } catch (err) {
    console.error(error.message);
    res.status(500).send("Error en el servidor");
  }
})
```

Ruta para la autorización:

```
//Ruta para verificar el token
router.get("/verificado", autorizacion, async (req, res) => {
  try {
    res.json(true);
  } catch (err) {
    console.error(error.message);
    res.status(500).send("Error en el servidor");
  }
});
```

Esta ruta sirve para verificar la validez del token.

En principal.js se tiene un método que verifica la existencia del usuario en la base de datos

```
router.get("/", autorizacion, async (req, res) => {
  try {
    const user = await pool.query("SELECT user_name FROM users WHERE user_id =?", [
      req.user
    ]);
    res.json(user.rows[0]);
  } catch (err) {
    console.error(err.message);
    return res.status(403).json("Error en el servidor");
  }
})
```

Utilidades

En esta carpeta se encuentra el archivo jwtGenerator.js, dentro de este archivo hay un método que sirve para generar el token para el usuario.

```
const jwt = require("jsonwebtoken");
require("dotenv").config();

function jwtGenerator(user_id) {
  const payload = {
    user: user_id
  };

  return jwt.sign(payload, process.env.jwtSecret, { expiresIn: "1hr" });
}

module.exports = jwtGenerator;
```

este token expira en 1 hora.

Finalmente, en index.js tenemos un consolidado en donde se llama al middleware y las rutas.

```
const express = require("express");
const app = express();
const cors = require("cors");

//Middleware
app.use(express.json()); //req.body
app.use(cors());

//Rutas
app.use("/auth", require("./routes/jwtAuth"));

//ruta pagina principal
app.use("/principal", require("./routes/principal"));

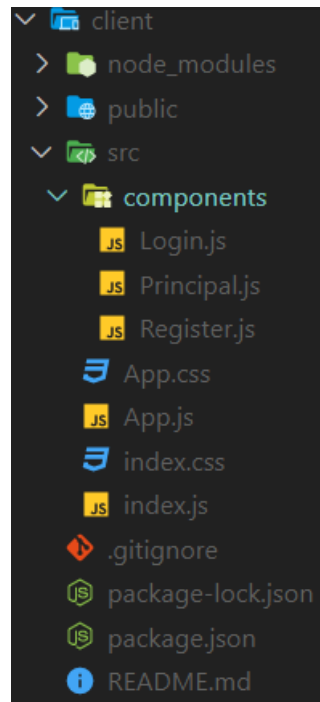
app.listen(5000,()=>{
  console.log("Servidor en el puerto 5000");
})
```

FrontEnd

El FrontEnd fue realizado con tecnología React. Este FrontEnd cuenta con 3 páginas: Pagina Login, pagina para el registro y la pagina principal.

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.

Login.js

Primero se ve un método que verifica las credenciales requeridas para hacer login

```
const Login = ({ setAuth }) => {  
  //definiendo los inputs para las credenciales  
  const [inputs, setInputs] = useState({  
    email: "",  
    password: ""  
  });  
  
  const { email, password } = inputs; // inputs requeridos  
  const onChange = e => {setInputs({ ...inputs, [e.target.name]: e.target.value });};  
  
  const onSubmitForm = async e => {  
    e.preventDefault();  
    try { //petición utilizada para comprobar las credenciales de acceso  
      const body = { email, password }  
      const response = await fetch("http://localhost:5000/auth/login", {  
        method: "POST",  
        headers: {  
          "Content-Type": "application/json"  
        },  
        body: JSON.stringify(body)  
      });  
  
      const parseRes = await response.json();  
      //verificación del token  
      if (parseRes.token) {  
        localStorage.setItem("token", parseRes.token);  
        setAuth(true);  
        toast.success("Inicio de sesión exitoso"); //mensaje alerta  
      } else {  
        setAuth(false); // cuando el token es incorrecto no permite el acceso  
        toast.error(parseRes); //mensaje alerta  
      }  
    } catch (err) {  
      console.error(err.message);  
    }  
  }  
};
```

Seguidamente tenemos el código que se usa para visualizar el formulario:

```
return (
  <div className="container mt-4">
    <div className="row justify-content-center">
      <div className="col-md-6">
        <Card className="bg-secondary">
          <CardBody>
            <h1 className="text-center text-white">Inicio de sesion</h1>
            <Fragment>
              <form onSubmit={onSubmitForm}>
                <input type="email" name="email" placeholder="Escriba su Email"
                  className="form-control my-3" value={email} onChange={e => onChange(e)} />
                <input type="password" name="password" placeholder="Escriba su contraseña"
                  className="form-control my-3" value={password} onChange={e => onChange(e)} />
                <button className="btn btn-success btn-block">Iniciar Sesión</button>
              </form>
              <br />
              <Link to="/register" className="btn btn-info btn-block">Regitrarse</Link>
            </Fragment>
          </CardBody>
        </Card>
      </div>
    </div>
  </div>
);
};
```

Principal.js

En este archivo se tiene un método que hace una petición GET para obtener los datos del usuario que se ha iniciado sesión, también hay un método para cerrar sesión, mismo que retira el acceso al token.

```
const Principal = ({ setAuth }) => {

  const [name, setName] = useState("");
  async function getName() {
    try {
      const res = await fetch("http://localhost:5000/principal/", {
        method: "GET",
        headers: { token: localStorage.token }
      });
      const parseRes = await res.json();
      setName(parseRes.user_name);
    } catch (err) {
      console.error(err.message);
    }
  };

  const logout = e => {
    e.preventDefault();
    localStorage.removeItem("token");
    setAuth(false);
    toast.success("Acaba de cerrar sesion");
  };

  useEffect(() => {
    getName();
  }, []);
};
```

Seguidamente se encuentran las etiquetas que hacen visible la página principal:

```
return (  
  <Fragment>  
    <h1> Pagina principal</h1>  
    <h2>Bienvenido {name}</h2>  
    <button onClick={e => logout(e)} className="btn btn-danger" >Salir</button>  
  </Fragment>  
);
```

Register.js

Aquí se encuentra el método que tiene la petición POST, mismo que es necesario para enviar los valores al BackEnd:

```
const Register = ({ setAuth }) => {  
  //inicializacion de los inputs requeridos  
  const [inputs, setInputs] = useState({  
    name: "",  
    email: "",  
    password: ""  
  });  
  
  const { name, email, password } = inputs;  
  const onChange = e => setInputs({ ...inputs, [e.target.name]: e.target.value });  
  
  const onSubmitForm = async e => {  
    e.preventDefault();  
    //consulaa POST necesaria para hacer el resgistro  
    try {  
      const body = { email, password, name };  
      const response = await fetch("http://localhost:5000/auth/register",  
        {  
          method: "POST",  
          headers: {  
            "Content-type": "application/json"  
          },  
          body: JSON.stringify(body)  
        })  
    );  
  
    const parseRes = await response.json();  
    if (parseRes.token) {  
      localStorage.setItem("token", parseRes.token);  
      setAuth(true);  
      toast.success("Registro exitoso");  
    } else {  
      setAuth(false);  
      toast.error(parseRes);  
    }  
  } catch (err) {  
    // ...  
  }  
}
```


Seguidamente podemos observar las etiquetas que hacen visible el formulario:

```
return (
  <div className="container mt-4">
    <div className="row justify-content-center">
      <div className="col-md-6">
        <Card className="bg-secondary">
          <CardBody>
            <h1 className="text-center text-white">Registrarse</h1>
            <Fragment>
              <form onSubmit={onSubmitForm}>
                <input type="text" name="name" value={name} placeholder="Escriba su nombre"
                  onChange={e => onChange(e)} className="form-control my-3"
                />
                <input type="text" name="email" value={email} placeholder="Escriba su email"
                  onChange={e => onChange(e)} className="form-control my-3"
                />
                <input type="password" name="password" value={password} placeholder="Escriba su contraseña"
                  onChange={e => onChange(e)} className="form-control my-3"
                />
                <button className="btn btn-success btn-block form-control my-3">Registrar</button>
              </form>
              <Link to="/login" className="btn btn-info btn-block">Inicio de sesión</Link>
            </Fragment>
          </CardBody>
        </Card>
      </div>
    </div>
  </div>
);
```

Finalmente, en la carpeta principal client se tiene el archivo App.js donde se encuentran métodos de verificación, mismos que permiten se mantenga una correcta autenticación:

```
//Importacion de los componentens necesarios apra la autorizacion
import Principal from "../components/Principal";
import Login from "../components/Login";
import Register from "../components/Register";

toast.configure();// cnfiguracion para las notificaciones

function App() {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const setAuth = boolean => {
    setIsAuthenticated(boolean);
  };

  //se usa auth para verificar que el usuario tiene un token valido de acceso.
  async function isAuth(){
    try{
      const response = await fetch("http://localhost:5000/auth/verificado", {
        method: "POST",
        headers: { token: localStorage.token }
      });

      const parseRes = await response.json();
      parseRes === true ? setIsAuthenticated(true) : setIsAuthenticated(false);
    } catch (err) {
      console.error(err.message);
    }
  }
}
```

Además, se tiene el return que se usará en la validación y redireccionamiento:

```
return (
  <Fragment>
    <Router>
      <div className="container">
        <Switch>
          <Route
            exact
            path="/login"
            render={props =>
              !isAuthenticated ? (
                <Login {...props} setAuth={setAuth} />
              ) : (
                <Redirect to="/principal" />
              )
            }
          />
          <Route
            exact
            path="/register"
            render={props =>
              !isAuthenticated ? (
                <Register {...props} setAuth={setAuth} />
              ) : (
                <Redirect to="/principal" />
              )
            }
          />
          <Route
            exact
            path="/principal"
            render={props =>
              isAuthenticated ? (
                <Principal {...props} setAuth={setAuth} />
              ) : (
                <Redirect to="/login" />
              )
            }
          />
        </Switch>
      </div>
    </Router>
  </Fragment>
)
```