Universidad San Carlos de Guatemala
Escuela de Ciencias y Sistemas

Practicas Iniciales; Sección F-, Grupo 10

MANUAL TECNICO

Integrantes:

202200031-EDISONMAURICIOGARCÍARODRÍGUEZ

202202233-MATTHEWEMMANUELREYESMELGAR

202202481-JOSUÉNABÍHURTARTEPINTO

202211515-GONZALOFERNANDOPÉREZCAZÚN

202200061-JONATANSAMUELROJASMAEDA

201801601-YOSHUAINGNACIOTECÚNMARROQUÍN

Auxiliares:

202006353-JUANJOSUEZULETABEB

202004810-HENRYRONELYMENDOZAAGUILAR

Descripción general

La aplicación que se llevó a cabo es una aplicación desarrollada para un entorno web. Tiene como propósito crear un tipo de red social para poder tomar referencia de los distintos profesores de la facultad de Ingeniería.

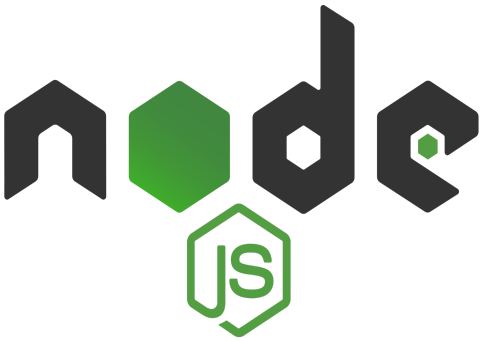
Frontend

Para el frontend se realizó en React, HTML y CSS es una forma de crear interfaces de usuario dinámicas y reutilizables para aplicaciones web. React es una librería de JavaScript que permite crear componentes, que son pequeñas piezas de la interfaz que contienen la estructura (HTML), la lógica (JavaScript) y los estilos (CSS) en un solo paquete. Estos componentes se pueden combinar y renderizar según el estado de la aplicación y las interacciones del usuario. HTML es el lenguaje que define la estructura y el contenido de una página web, mientras que CSS es el lenguaje que define el diseño y la apariencia de los elementos HTML. Para usar React, HTML y CSS en un proyecto frontend, se necesita un entorno de desarrollo que incluya herramientas como un editor de código, un gestor de paquetes, un compilador o transpilador, un servidor local y un navegador web.

```
import React, { Component } from 'react';
import '../App';
import '../App.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import axios from 'axios';
import Cookies from 'universal-cookie';

const baseUrl = "http://localhost:3001/login";
```

```
import React, { Component } from 'react';
import '../App';
import '../App.css';
```

Backend



Para el backend se utilizó el lenguaje de programación Javascript por medio de NodeJS como un servicio de REST API. Esta parte se encargará de las solicitudes de los clientes, comunicarse con la base de datos elaborada en MySQL para poder almacenar, recuperar información y realizar otras operaciones necesarias para el funcionamiento de la aplicación. Se utilizó NodeJS para crear un servicio de REST API que actúa como el puente entre el frontend y la base de datos MySQL. NodeJS es conocido por su eficiencia y escalabilidad en aplicaciones web y se integra perfectamente con bases de datos relacionales como MySQL. Este servicio de REST API es el corazón de la lógica empresarial de la aplicación. Se encarga de manejar las solicitudes de los clientes, procesar datos, realizar validaciones y conectarse a la base de datos MySQL para almacenar y recuperar información. A través de endpoints API bien definidos, el backend garantiza una comunicación fluida entre el frontend y la base de datos.

ExpressJS

Se usó el framework express para poder crear apis de una manera más fácil. Lo bueno de usar express es que es de licencia de código abierto, esto permite que su manipulación sea mucho más personalizable.

Base de datos

Para el almacenamiento de datos se ha utilizado MySQL, un sistema de gestión de bases de datos relacionales ampliamente reconocido por su confiabilidad y rendimiento. MySQL permite organizar y almacenar de manera eficiente la información necesaria para que la aplicación funcione correctamente. Además, se ha utilizado MySQL Workbench como una herramienta de administración de bases de datos que simplifica la creación, modificación y gestión de la base de datos. Esta combinación de MySQL y MySQL Workbench garantiza una administración eficaz de los datos

Diccionario de funciones

Frontend

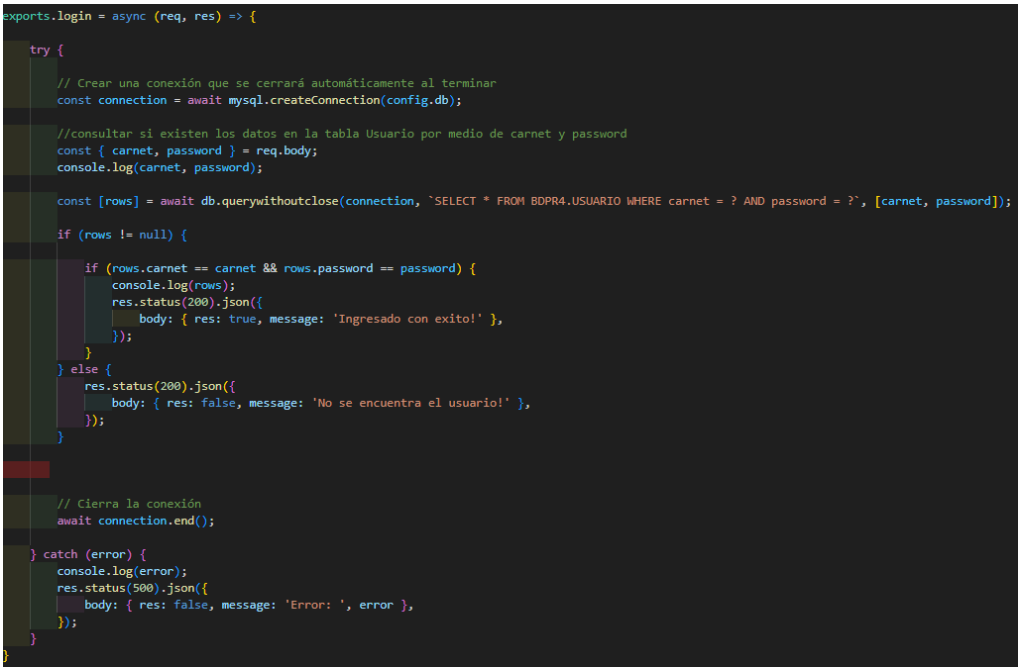
1. **JSX**: Es una sintaxis extendida de JavaScript que te permite escribir código HTML dentro de tu código JavaScript, lo que facilita la creación y lectura de los componentes. Por ejemplo, puedes usar JSX para crear un componente que muestre el nombre y el cargo de un empleado.
2. **Hooks**: Son funciones especiales que te permiten usar el estado y otras características de React dentro de los componentes funcionales, que son más simples y fáciles de reutilizar que los componentes de clase. Por ejemplo, puedes usar el hook useState para crear una variable de estado que almacene el nombre del empleado y una función para actualizarlo.
3. **Props**: Son los atributos o propiedades que se pasan a los componentes como argumentos, lo que permite personalizar su comportamiento y apariencia. Por ejemplo, puedes pasar el nombre y el cargo del empleado como props al componente que los muestra.
4. **Eventos**: Son las acciones que ocurren en la interfaz, como hacer clic, escribir o mover el mouse, y que pueden ser capturadas por los componentes para ejecutar alguna función. Por ejemplo, puedes usar el evento onChange para detectar cuando el usuario cambia el nombre del empleado en un input2.
5. **Renderizado condicional**: Es la técnica que permite mostrar u ocultar ciertos elementos o componentes según alguna condición lógica. Por ejemplo, puedes usar el operador ternario para mostrar un mensaje de error si el nombre del empleado está vacío.
6. **Axios:** Axios es una popular biblioteca de JavaScript que se utiliza para realizar solicitudes HTTP en aplicaciones web, en particular en aplicaciones frontend (como las creadas con React, Vue, Angular) y en servidores Node.js. Proporciona una interfaz simple y basada en promesas para interactuar con servicios web y API. Aquí te muestro cómo utilizar Axios para realizar solicitudes HTTP en JavaScript:
7. **baseurl:**La variable baseUrl en JavaScript se utiliza para almacenar una URL base que se puede concatenar con rutas específicas para formar URL completas. Esta es una práctica común en el desarrollo web para evitar escribir la URL completa en cada solicitud o enlace.

Backend

- **registro()**: Crea una conexión a una base de datos MySQL, obtiene datos del cuerpo de una solicitud HTTP, luego inserta esos datos en una tabla llamada "USUARIO" y cierra la conexión automáticamente al finalizar. Finalmente, devuelve una respuesta JSON que indica si la operación fue exitosa o si se produjo un error en el proceso.

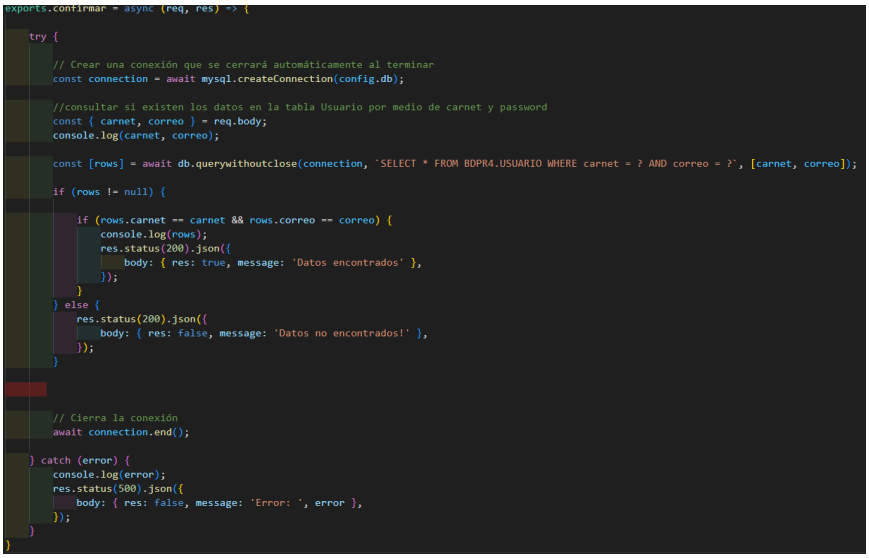
```
exports.registro = async (req, res) => {

    try {

        // Crear una conexión que se cerrará automáticamente al terminar
        const connection = await mysql.createConnection(config.db);

        // Obtener los datos del body
        const { carnet, nombre, apellido, correo, password } = req.body;


        // Insertar los datos en la tabla Usuario
        await db.querywithoutclose(connection, `INSERT INTO BDPR4.USUARIO (carnet, nombre, apellido, correo, password) VALUES (?, ?, ?, ?, ?)`, [carnet, nombre, apellido, correo, password]);

        // Cierra la conexión
        await connection.end();

        res.status(200).json({
            body: { res: true, message: 'Ingresado con exito!' },
        });
    } catch (error) {
        console.log(error);
        res.status(500).json({
            body: { res: false, message: 'Error: ', error },
        });
    }
}
```

- **login()**: Crea una conexión a la base de datos MySQL, consulta si existen datos en la tabla "USUARIO" comparando el "carnet" y la "password" proporcionados en la solicitud HTTP, y luego responde con un mensaje de éxito si los datos coinciden o un mensaje de error si no se encuentra el usuario en la base de datos



```
exports.login = async (req, res) => {

    try {

        // Crear una conexión que se cerrará automáticamente al terminar
        const connection = await mysql.createConnection(config.db);

        //consultar si existen los datos en la tabla Usuario por medio de carnet y password
        const { carnet, password } = req.body;
        console.log(carnet, password);

        const [rows] = await db.querywithoutclose(connection, `SELECT * FROM BDPR4.USUARIO WHERE carnet = ? AND password = ?`, [carnet, password]);

        if (rows != null) {

            if (rows.carnet == carnet && rows.password == password) {
                console.log(rows);
                res.status(200).json({
                    body: { res: true, message: 'Ingresado con exito!' },
                });
            }
        } else {
            res.status(200).json({
                body: { res: false, message: 'No se encuentra el usuario!' },
            });
        }


        // Cierra la conexión
        await connection.end();

    } catch (error) {
        console.log(error);
        res.status(500).json({
            body: { res: false, message: 'Error: ', error },
        });
    }
}
```

- **confirmar()**: Crea una conexión a la base de datos MySQL, consulta si existen datos en la tabla "USUARIO" comparando el "carnet" y el "correo" proporcionados en la solicitud HTTP, y luego responde con un mensaje de éxito si los datos coinciden o un mensaje de error si no se encuentran los datos en la base de datos. La conexión a la base de datos se cierra automáticamente al finalizar la operación.



```
exports.confirmar = async (req, res) => {

    try {

        // Crear una conexión que se cerrará automáticamente al terminar
        const connection = await mysql.createConnection(config.db);

        //consultar si existen los datos en la tabla Usuario por medio de carnet y password
        const { carnet, correo } = req.body;
        console.log(carnet, correo);

        const [rows] = await db.querywithoutclose(connection, `SELECT * FROM BDPR4.USUARIO WHERE carnet = ? AND correo = ?`, [carnet, correo]);

        if (rows != null) {

            if (rows.carnet == carnet && rows.correo == correo) {
                console.log(rows);
                res.status(200).json({
                    body: { res: true, message: 'Datos encontrados' },
                });
            }
        } else {
            res.status(200).json({
                body: { res: false, message: 'Datos no encontrados!' },
            });
        }


        // Cierra la conexión
        await connection.end();

    } catch (error) {
        console.log(error);
        res.status(500).json({
            body: { res: false, message: 'Error: ', error },
        });
    }
}
```

- **restablecer()**: Crea una conexión a la base de datos MySQL, recibe datos del cuerpo de la solicitud HTTP, y luego actualiza la contraseña del usuario en la tabla "USUARIO" si el "carnet" coincide con el proporcionado en la solicitud, reemplazando la contraseña anterior por la nueva. Después de completar la operación, cierra automáticamente la conexión a la base de datos y responde con un mensaje de éxito si la contraseña se actualizó correctamente o un mensaje de error en caso de problemas durante el proceso.

```
exports.restablecer = async (req, res) => {

    try {

        // Crear una conexión que se cerrará automáticamente al terminar
        const connection = await mysql.createConnection(config.db);

        // Obtener los datos del body
        const { carnet, password } = req.body;

        // Insertar los datos en la tabla Usuario donde el carnet sea igual al carnet ingresado se debe sustituir la contraseña por la nueva
        await db.querywithoutclose(connection, `UPDATE BDPR4.USUARIO SET password = ? WHERE carnet = ?`, [password, carnet]);

        // Cierra la conexión
        await connection.end();

        res.status(200).json({
            body: { res: true, message: 'Actualizado con exito!' },
        });
    } catch (error) {
        console.log(error);
        res.status(500).json({
            body: { res: false, message: 'Error: ', error },
        });
    }
}
```

Coneccion a la base de datos:

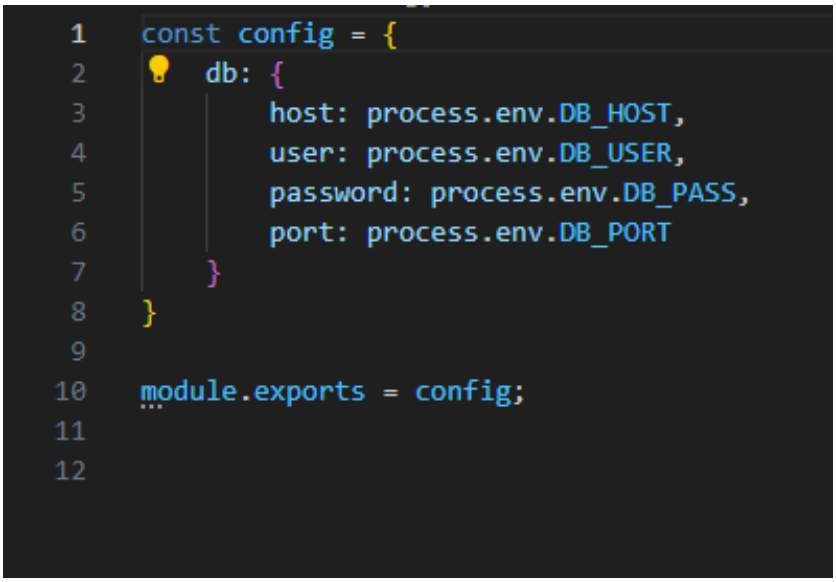
./db/conección.js

Todas nuestras funciones que consulten o que llamen a nuestra función. Lo primero que realizará es llamar a la función consulta cada vez que la solicita envía lo que necesita realizar como por ejemplo una consulta, un select etc, Los parámetros con los cuales les hará la petición a nuestra base de datos.

```
const mysql = require('mysql2/promise')
const config = require('./config')

async function query(sql, params) {
    const connection = await mysql.createConnection(config.db)
    const [results,] = await connection.execute(sql, params)

    connection.end(function (err) {
        console.log("Conexion cerrada")
        if (err) {
            return console.log("error conexion: ", err.message);
        }
        console.log("Conexion cerrada exitosamente");
    });
    connection.destroy()
    return results;
}

//PARA NO CERRAR LA CONEXION
async function querywithoutclose(connection, sql, params) {
    const [results,] = await connection.execute(sql, params);
    return results;
}




module.exports = { query,querywithoutclose };

```

./db/config.js

Hará un enlace con nuestra base de datos los cuales ubicara en donde se encuentra la base de datos, el usuario, la contraseña y en el puerto que se está ejecutando.



```
const config = {
    db: {
        host: process.env.DB_HOST,
        user: process.env.DB_USER,
        password: process.env.DB_PASS,
        port: process.env.DB_PORT
    }
}

module.exports = config;
```