


PROYECTO REACT

1. Paso – Creación del Backend

1.2 Configuración de la Base de Datos (config.js):

```
 Credenciales de la base de datos
const config = {
  db: {
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'bdgestion',
    port: "3300",
    connectTimeout: 60000
  },
};

// Exportar configuración para su uso en otros archivos
module.exports = config;
```

Este archivo configura las credenciales de la base de datos, como el host, usuario, contraseña, nombre de la base de datos y puerto.

1.2 2. Funciones de Utilidad (helper.js):

```
// Maneja el caso en el que la base de datos no devuelve nada:
function emptyOrRows(rows) {
  if (!rows) {
    // Si no hay filas devueltas, retorna un array vacío []
    return [];
  }
  // Si hay filas devueltas, retorna las filas.
  return rows;
}

// Exportar función para su uso en otros archivos
module.exports = {
  ...
  emptyOrRows
};
```

Utilizamos una función para manejar en caso de que la bd no devuelva nada o devuelva los datos correspondientes.

```
// Conexión a la base de datos
const mysql = require('mysql2/promise');
const config = require('../config');

// Función para realizar consultas a la base de datos
async function query(sql, params) {
  // Crear una conexión a la base de datos
  const connection = await mysql.createConnection(config.db);

  // Ejecutar la consulta y obtener los resultados
  const [results] = await connection.execute(sql, params);

  // Cerrar la conexión y devolver los resultados
  return results;
}

// Función para probar la conexión a la base de datos
async function testDatabaseConnection() {
  try {
    // Crear una conexión de prueba
    const connection = await mysql.createConnection(config.db);

    // Mostrar mensaje de conexión exitosa con detalles
    console.log('\nConexión a la base de datos exitosa, \n|puerto: ' + config.db.port,
      '|usuario: ' + config.db.user, '|base de datos: ' + config.db.database);

    // Cierra la conexión de prueba
    connection.end();
  } catch (error) {
    // Manejar errores al conectar a la base de datos
    console.error('Error al conectar a la base de datos:', error);
  }
}

// Exportar funciones para su uso en otros archivos
module.exports = {
  query,
  testDatabaseConnection,
}
```

1.3 - Conexión y Prueba de la Base de Datos (db.js):

Este código tiene dos funciones:

- **query:** Función para realizar consultas a la base de datos. Crea una conexión, ejecuta la consulta y devuelve los resultados.
- **testDatabaseConnection:** Función para probar la conexión a la base de datos. Crea una conexión de prueba, imprime detalles de conexión y cierra la conexión.

1.4 - Autenticación del Usuario (login.js):

```
// Importar módulos necesarios
const db = require('./db');
const helper = require('./helper');
const config = require('./config');

// Función para obtener datos de usuario por login y contraseña
async function getUserData(login, password) {
  try {
    // Realizar consulta a la base de datos
    const rows = await db.query(
      `SELECT nombre, rol FROM usuarios WHERE login = ? AND password = ?`, [login, password]
    );

    // Procesar los resultados y devolver los datos del usuario
    const data = helper.emptyOrRows(rows[0]);
    return {
      data
    };
  } catch (error) {
    // Manejar errores al obtener datos del usuario
    console.error(`Error while getting data `, error.message);
    return {
      error: 'Error en el servidor'
    };
  }
}

// Exportar función para su uso en otros archivos
module.exports = {
  getUserData
};
```

Función para obtener datos de usuario por login y contraseña. Realiza una consulta SQL y devuelve los datos del usuario o un error en caso de fallo.

1.5 Configuración del Servidor Express (index.js):

```
// Importar módulos necesarios
const express = require('express');
const cors = require('cors');
const { insertData, getData, deleteData } = require('./services/item');
const { testDatabaseConnection } = require('./services/db');
const { getUserData } = require('./services/login');

// Configuración del puerto y creación de la aplicación Express
const port = 3030;
const app = express();

// Configuración de middleware para el manejo de JSON, datos de formulario y CORS
app.use(express.json());
app.use(
  express.urlencoded({
    extended: true
  })
);
app.use(cors());

// Ruta de bienvenida
app.get('/', function (req, res) {
  res.json({ message: 'Hola Mundo!' });
});

// Ruta de autenticación de usuario
app.get('/login', async function (req, res, next) {
  const { login, password } = req.query;
  const result = await getUserData(login, password);
  if (result.error) {
    res.status(500).json({ error: result.error });
  } else {
    res.json(result);
  }
});

// Configuración del servidor y prueba de conexión a la base de datos
app.listen(port, async () => {
  console.log(`\nAPI escuchando en el puerto ${port}`);
  await testDatabaseConnection();
});

// Middleware para manejo de errores
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
});
```

- Configura un servidor Express que escucha en el puerto 3030.
- Define una ruta de prueba ("/") y una ruta de autenticación ("/login").
- Utiliza las funciones de getUserData y testDatabaseConnection.

2. Frontend - Paso a Paso:

2.1. Punto de Entrada del Frontend (index.js):

```
// Este archivo es el punto de entrada principal del frontend.  
// Se encarga de renderizar la aplicación React en el elemento con el id 'root'.  
  
import React from 'react';  
import { createRoot } from 'react-dom';  
import ReactDOM from 'react-dom/client';  
import { Provider } from 'react-redux';  
import App from './App'; // Importa el componente principal de la aplicación.  
import store from './redux/index'; // Importa la store de Redux configurada.  
  
// Crea un punto de entrada para la aplicación React en el elemento con id 'root'.  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(  
  // Envuelve la aplicación en el componente Provider de Redux para que tenga acceso a la store.  
  <React.StrictMode>  
    <Provider store={store}>  
      <App />  
    </Provider>  
  </React.StrictMode>,  
);
```

- Configura el punto de entrada principal del frontend.
- Crea un punto de entrada para la aplicación React y envuelve la aplicación en el componente
- Provider de Redux para tener acceso a la store.

2.2 Configuración de Rutas (app.js):

```
import React from 'react';
import { createBrowserRouter, RouterProvider, Route } from 'react-router-dom';
import Home from './components/Home';
import Login from './components/Login';
import Informe from './components/Informe';

const router = createBrowserRouter([
  {
    path: '/',
    children: [
      {
        index: true,
        element: <Login />,
      },
      {
        path: 'Home',
        element: <Home />,
      },
    ],
  },
  {
    path: 'Informe',
    element: <Informe />,
  },
]);

function App() {
  return (
    <RouterProvider router={router}>
      <Route />
    </RouterProvider>
  );
}

export default App;
```

- Configura las rutas de la aplicación utilizando react-router-dom.
- Define rutas para la página de inicio (/), la página de inicio de sesión (/Home), y la página de informes (/Informe).

2.3 Configuración del redux

Redux:

- Redux es un contenedor de estado predecible para aplicaciones JavaScript. Su objetivo principal es gestionar el estado de la aplicación de manera centralizada y predecible. Esto es especialmente útil en aplicaciones complejas con muchos componentes que necesitan compartir y responder a cambios en el estado.

Index.js

```
// Este archivo configura y exporta la store de Redux.  
// Importa el reducer de autenticación (loginReducer)  
// y utiliza configureStore de '@reduxjs/toolkit' para crear la store.  
  
import loginReducer from './storelogin'; // Importa el reducer de autenticación.  
  
import { configureStore } from '@reduxjs/toolkit';  
  
// Configura la store de Redux con el reducer de autenticación.  
const store = configureStore({  
  reducer: {  
    login: loginReducer,  
  },  
});  
  
// Exporta la store configurada para su uso en la aplicación.  
export default store;
```

- **Este archivo configura y exporta la tienda (store) de Redux, que actúa como un contenedor centralizado para el estado de toda la aplicación**

Función:

Configura una tienda Redux que manejará el estado global de la aplicación. Utiliza el reductor de autenticación (loginReducer) para gestionar el estado relacionado con la autenticación.

Configuración del componente Storelogin.js

```
// Este archivo define el slice de autenticación utilizando createSlice de '@reduxjs/toolkit'.
// Define el estado inicial, las acciones de login y logout, y exporta el slice y las acciones.

import { createSlice } from '@reduxjs/toolkit';

// Define el estado inicial para la autenticación.
const initialAuthState = {
  isAuthenticated: false,
  userName: '',
  userRol: '',
};

// Crea el slice de autenticación con el nombre 'authentication'.
const authSlice = createSlice({
  name: 'authentication',
  initialState: initialAuthState,
  reducers: {
    login: (state, action) => {
      // Actualiza el estado con la información del usuario al hacer login.
      const userData = action.payload;
      state.isAuthenticated = true;
      state.userName = userData.name;
      state.userRol = userData.rol;
    },
    logout: (state) => {
      // Restablece el estado al hacer logout.
      state.isAuthenticated = false;
      state.userName = '';
      state.userRol = '';
    },
  },
});

// Exporta las acciones de login y logout.
export const loginActions = authSlice.actions;
// Exporta el reducer del slice para su uso en la store.
export default authSlice.reducer;
```

Este archivo define el slice (porción) de autenticación utilizando la función createSlice de @reduxjs/toolkit. También define las acciones de login y logout y exporta tanto las acciones como el reductor.

- `import { createSlice } from '@reduxjs/toolkit';`: Importa la función createSlice de @reduxjs/toolkit para crear slices de Redux.
- `const initialAuthState = { isAuthenticated: false, userName: '', userRol: '', };`: Define el estado inicial para la autenticación.
- `const authSlice = createSlice({ /* ... */ });`: Crea un slice llamado authentication utilizando createSlice, que incluye el nombre del slice, el estado inicial y las acciones de login y logout.

- `export const loginActions = authSlice.actions;` Exporta las acciones de login y logout para que puedan ser utilizadas en otros lugares de la aplicación.
- `export default authSlice.reducer;` Exporta el reductor del slice para que pueda ser utilizado por la tienda de Redux.

Función:

Define el estado inicial y las acciones relacionadas con la autenticación.

Un slice en Redux incluye el reductor y las acciones relacionadas con una porción específica del estado de la aplicación. En este caso, se centra en la autenticación del usuario.

Configuración del componente de inicio sesión (Login.jsx)

Explicación :

1. State y Manejadores de Cambio:

- `useState` se usa para crear estados locales para el nombre de usuario (login) y la contraseña (password).
- `handleUserChange` y `handlePasswordChange` son funciones que actualizan los estados cuando los campos de entrada cambian.

2. Envío del Formulario y Autenticación:

- `handleSubmit` es llamada al hacer clic en el botón de inicio de sesión.
- Realiza una solicitud al servidor utilizando `fetch` con los valores del nombre de usuario y la contraseña.
- Maneja la respuesta del servidor, actualiza el estado de Redux y navega a la página principal si la autenticación es exitosa.

3. Estructura Visual:

- Utiliza componentes de Material-UI como `Container`, `Paper`, `Avatar`, `Typography`, `TextField`, y `Button` para crear una interfaz visual limpia y agradable.

4. *Redux:*

- Utiliza useDispatch de React-Redux para obtener la función de despacho de Redux.
- Despacha una acción de Redux (loginActions.login) con los datos del usuario cuando la autenticación es exitosa.

5. *React Router y Navegación:*

- Utiliza useNavigate de react-router-dom para navegar a la página principal ('/home') después de una autenticación exitosa.
- Este componente es parte fundamental de la aplicación, manejando la interfaz de inicio de sesión y conectándose con el backend para la autenticación del usuario.

Configuración componente Home.jsx

- Este archivo Home.js es un componente de React que representa la página principal de la aplicación después de iniciar sesión.

Lógica de Autenticación:

- Utiliza useSelector para obtener el estado de autenticación desde Redux.
- Utiliza useEffect para redirigir al usuario a la página de inicio de sesión si no está autenticado.

Barra de Navegación (AppBar):

- Utiliza componentes de Material-UI para crear una barra de navegación.
- Muestra el nombre del usuario autenticado y enlaces a diferentes secciones de la aplicación.

Formulario de Inserción de Datos:

- Utiliza un formulario (Box) con campos (TextField) para insertar datos.
- Utiliza el estado local formData para rastrear los valores del formulario.

Tabla de Registros:

- Utiliza una tabla (TableContainer, Table, etc.) para mostrar los registros.
- Mapea los datos (data) para mostrar cada fila en la tabla.
- Cada fila tiene un botón para eliminar el registro correspondiente.

Funciones Asíncronicas:

- handleInsertData: Inserta datos en el servidor utilizando la API fetch.
- handleDelete: Elimina registros utilizando la API fetch.
- handleShowRecords: Muestra registros llamando a fetchData.

Funciones Adicionales:

- fetchData: Obtiene los registros del servidor utilizando la API fetch.
- handleLogout: Desconecta al usuario, actualiza el estado Redux y redirige a la página de inicio de sesión.

Condición de Renderizado:

- Renderiza el contenido solo si el usuario está autenticado.

Configuración del Componente Home.jsx

- El archivo Home.js es un componente de React que representa la página principal de la aplicación después de iniciar sesión.

Lógica de Autenticación:

- Utiliza useSelector para obtener el estado de autenticación desde Redux.
- Utiliza useEffect para redirigir al usuario a la página de inicio de sesión si no está autenticado.

Barra de Navegación (AppBar):

- Utiliza componentes de Material-UI para crear una barra de navegación.
- Muestra el nombre del usuario autenticado y enlaces a diferentes secciones de la aplicación.

Formulario de Inserción de Datos:

- Utiliza un formulario (Box) con campos (TextField) para insertar datos.

- Utiliza el estado local `formData` para rastrear los valores del formulario.

Tabla de Registros:

- Utiliza una tabla (`TableContainer`, `Table`, etc.) para mostrar los registros.
- Mapea los datos (`data`) para mostrar cada fila en la tabla.
- Cada fila tiene un botón para eliminar el registro correspondiente.

Funciones Asincrónicas:

- **`handleInsertData`:** Inserta datos en el servidor utilizando la API `fetch`.
- **`handleDelete`:** Elimina registros utilizando la API `fetch`.
- **`handleShowRecords`:** Muestra registros llamando a `fetchData`.
-

Funciones Adicionales:

- **`fetchData`:** Obtiene los registros del servidor utilizando la API `fetch`.
- **`handleLogout`:** Desconecta al usuario, actualiza el estado `Redux` y redirige a la página de inicio de sesión.

Condicional de Renderizado:

- Renderiza el contenido solo si el usuario está autenticado.

Configuración del componente `Topbar.jsx`

- El archivo `Topbar.jsx` define un componente de barra superior que se utiliza para la navegación y visualización del nombre de usuario.

Configuración del componente `Informe.jsx`

El archivo `Informe.jsx` representa la página de informes. Utiliza el componente `Topbar` y muestra un informe de colección utilizando el componente `InformeColeccion`.

Configuración del componente InformeColeccion.jsx

- El componente InformeColeccion.jsx muestra un informe de colección utilizando la biblioteca @material-table/core. Se obtienen los datos del servidor utilizando la función getItems y se utilizan para configurar la tabla.

Configuración del componente Ayuda.jsx

- El archivo Ayuda.jsx es un componente que muestra información de ayuda. Utiliza el componente Topbar y tiene botones para descargar manuales de usuario y guías rápidas
- . Los botones utilizan la función handleDownloadPDF para abrir los archivos PDF en una nueva ventana y también incluye un botón para cerrar sesión.

