TEMA: Integración del frontend con el backend

Actividad: Paso a Paso

**Objetivo**

**Implementar Autenticación y Autorización Seguras:**

* **Objetivo Específico:** Implementar autenticación mediante tokens OAuth OpenID en el frontend y utilizar tokens JWT con roles de usuario para la autorización en el backend.
* **Resultado Esperado:** Los estudiantes podrán diseñar e implementar un sistema de autenticación y autorización seguro y eficiente, utilizando tecnologías de tokens para controlar el acceso a recursos y funcionalidades específicas.

**Requisitos**

1. Descargar del portafolio del alumno el proyecto backend
2. Descargar del portafolio del alumno el proyecto frontend
3. Levantar keycloak con docker (docker run -d --name keycloak -e KEYCLOAK\_USER=admin -e KEYCLOAK\_PASSWORD=admin -p 8080:8080 jboss/keycloak)
4. Haber realizado la configuración del IDP:

-Creación de reino “comercio”

-Generación de roles: “admin” y “read\_only”

-Generación de 2 usuarios (administrador – rol admin+read\_only, lector – rol read\_only)

**Consigna**

1. El proyecto frontend debe disponibilizar tres páginas: Pública, Administradores/Lector, Lectores
2. El proyecto frontend debe delegar la autenticación en el proveedor de identidad y redirigir al frontend.
3. El proyecto frontend debe mostrar opciones Nuevo Producto y Editar solo a los usuarios autenticados con rol “admin”.
4. El proyecto frontend debe mostrar opciones Lista Productos a los usuarios con el rol autenticados con rol “admin” (el rol “admin” contiene al rol “read\_only”).
5. El proyecto frontend debe generar una sesión de usuario.
6. El proyecto backend debe verificar la autorización del usuario autenticado.
7. El código debe integrar una aplicación web que utiliza React para el frontend y Express.js para el backend.
8. La autenticación y la autorización se gestionan a través de Keycloak, lo que permite controlar el acceso a las rutas basándose en los roles de usuario. Las rutas relacionadas con los productos (/productos) están protegidas y solo son accesibles para usuarios autenticados con roles específicos.

**Desarrollo**

Recuerda que para instalar las dependencias definidas debes posicionarte en la raíz del proyecto y abrir una terminal.

1. Ejecutar npm install
2. Ejecutar npm start

**FRONT END**

Vamos a trabajar con otras dependencias:

|  |
| --- |
| Package.json |
| {    "name": "frontend\_producto\_dds",    "version": "0.1.0",    "private": true,    "dependencies": {      "axios": "^1.5.1",      "bootstrap": "^5.3.2",      "keycloak-js": "^22.0.4",      "react": "^18.2.0",      "react-bootstrap": "^2.9.0",      "react-dom": "^18.2.0",      "react-hook-form": "^7.47.0",      "react-router-bootstrap": "^0.26.2",      "react-router-dom": "^6.16.0",      },   } |

-React (import React from "react";):

* React es la biblioteca principal para construir interfaces de usuario en aplicaciones web. Es esencial para crear componentes y manejar el estado y el ciclo de vida de la aplicación.
* useForm (import { useForm } from "react-hook-form";):
* useForm es un hook de React que proporciona funcionalidades para manejar formularios de manera eficiente. Permite registrar campos del formulario, realizar validaciones y gestionar el estado del formulario.

-axios (import axios from "axios";):

* Axios es una biblioteca de JavaScript que facilita la realización de solicitudes HTTP desde el navegador o Node.js. Se utiliza para interactuar con el backend de la aplicación, enviar datos, y recibir respuestas del servidor.

-Form, Button, Container (import { Form, Button, Container } from "react-bootstrap";):

* Estos son componentes de la biblioteca React Bootstrap, que proporciona componentes preestilizados para el diseño de la interfaz de usuario. En este caso, se utilizan para crear un formulario con campos de entrada, un botón de envío y un contenedor para el formulario.

-useKeycloak (import { useKeycloak } from "@react-keycloak/web";):

* useKeycloak es un hook proporcionado por @react-keycloak/web que permite acceder a las funcionalidades de autenticación y autorización proporcionadas por Keycloak. Permite gestionar el estado de autenticación del usuario y acceder a su token de acceso.

-useNavigate (import { useNavigate } from "react-router-dom";):

* useNavigate es un hook proporcionado por React Router, que se utiliza para realizar la navegación programática en la aplicación. Permite redirigir al usuario a diferentes rutas de la aplicación sin necesidad de un enlace.

|  |
| --- |
| index.js |
| const express = require('express');  const { keycloak } = require('./config/keycloak');  const productosRoutes = require('./routes/productosRoute');  const sequelize = require('./config/database');  const cors = require('cors');  require('dotenv').config();  const app = express();  app.use(cors());  // Habilita CORS para todas las rutas  // Middleware para parsear el cuerpo de las solicitudes JSON  app.use(express.json());  // Middleware de Keycloak  app.use(keycloak.middleware());  // Configuración de Sequelize  sequelize.sync()      .then(() => console.log('Database synced'))      .catch(err => console.error('Database sync failed:', err));  // Configuración de las rutas  app.use('/productos', productosRoutes);  // Middleware de manejo de errores personalizado  app.use((err, req, res, next) => {      console.error(err.stack);      res.status(500).send({ error: 'Something went wrong!' });  });  const PORT = process.env.PORT || 4000;  app.listen(PORT, () => {      console.log(`Server is running on port ${PORT}`);  });  module.exports = app; |

-Configuración de *keycloakProviderInitConfig*:

* Define una configuración para el proveedor de autenticación Keycloak. onLoad: 'none' especifica que Keycloak no se cargará automáticamente cuando la página se cargue.
* *checkLoginIframe*: false indica que el proveedor de Keycloak no debe intentar comprobar el estado de la sesión a través de un iframe.
* *redirectUri* especifica la URL a la que Keycloak redirigirá después de la autenticación.

-Inicialización de Keycloak:

* Importa la instancia de Keycloak configurada desde keycloak.js.
* Envuelve a la aplicación con ReactKeycloakProvider:

-Utiliza ReactKeycloakProvider para proporcionar el contexto de Keycloak a la aplicación. Esto permite que los componentes accedan a las funcionalidades de autenticación de Keycloak.

|  |
| --- |
| App.js |
| import React from "react";  import { BrowserRouter as Router, Routes, Route } from "react-router-dom";  import ProtectedRoute from "./components/ProtectedRoute";  import Login from "./components/Login";  import PublicPage from "./components/PublicPage";  import ProductoForm from "./components/ProductoForm";  import AppNavbar from "./components/AppNavbar";  import Callback from "./components/Callback";  import "bootstrap/dist/css/bootstrap.min.css";  import ProductosLista from "./components/ProductosLista";  const App = () => (    <Router>      <AppNavbar />      <Routes>        <Route path="/" element={<PublicPage />} />        <Route path="/login" element={<Login />} />        <Route path="/callback" element={<Callback />} />        <Route          path="/productos"          element={            <ProtectedRoute roles={["read\_only"]}>              <ProductosLista />            </ProtectedRoute>          }        />        <Route          path="/productos/editar"          element={            <ProtectedRoute roles={["admin"]}>              <ProductoForm />            </ProtectedRoute>          }        />      </Routes>    </Router>  );  export default App; |

-Configuración de Rutas:

* Define rutas para diferentes páginas de la aplicación, como la página pública, la página de inicio de sesión (/login), la página de devolución de llamada (/callback), la lista de productos (/productos), y el formulario de edición de productos (/productos/editar).
* Usa el componente ProtectedRoute para proteger las rutas que requieren roles específicos (read\_only para /productos y admin para /productos/editar).

|  |
| --- |
| Keycloak.js |
| // src/keycloak.js  import Keycloak from 'keycloak-js';  const keycloakConfig = {    url: 'http://localhost:8080/auth',  // URL de tu instancia de Keycloak    realm: 'comercio',  // Tu realm    clientId: 'frontend',  // Tu client ID  };  const keycloak = new Keycloak(keycloakConfig);  export default keycloak; |

-Configuración de Keycloak:

* Define la configuración de Keycloak, incluyendo la URL de la instancia de Keycloak, el reino y el ID de cliente.

-Inicialización de la Instancia de Keycloak:

* Crea una nueva instancia de Keycloak utilizando la configuración proporcionada.

|  |
| --- |
| productosroute.js |
| import { useEffect } from "react";  import { useNavigate } from "react-router-dom";  import { useKeycloak } from "@react-keycloak/web";  const ProtectedRoute = ({ roles, children }) => {    const { keycloak } = useKeycloak();    const navigate = useNavigate();    useEffect(() => {      if (        !keycloak.authenticated ||        (roles && !roles.some((role) => keycloak.hasRealmRole(role)))      ) {        navigate("/login");      }    }, [keycloak, navigate, roles]);    return children;  };  export default ProtectedRoute; |

-Configuración de Rutas de Productos:

* Define rutas para las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de los productos.
* Utiliza el middleware keycloak.protect para proteger las rutas, asegurando que solo los usuarios autenticados con roles específicos (read\_only y admin) puedan acceder a estas rutas.

|  |
| --- |
| ProductoForm.js |
| import React from "react";  import { useForm } from "react-hook-form";  import axios from "axios";  import { Form, Button, Container } from "react-bootstrap";  import { useKeycloak } from "@react-keycloak/web";  import { useNavigate } from "react-router-dom";  const ProductForm = () => {  const { keycloak } = useKeycloak();  const navigate = useNavigate();  const {      register,      handleSubmit,      formState: { errors },    } = useForm();  const onSubmit = async (data) => {      try {        await axios.post("http://localhost:4000/productos",data, {          headers: {            Authorization: `Bearer ${keycloak.token}`, // Usa el token de acceso de Keycloak          },        });        navigate("/productos")      } catch (error) {        console.error("Hubo un error al agregar el producto:", error);      }    };    return (      <Container className="mt-5" style={{ maxWidth: "500px" }}>        <Form onSubmit={handleSubmit(onSubmit)}>          <Form.Group className="mb-3">            <Form.Label>Nombre</Form.Label>            <Form.Control              type="text"              {...register("Nombre", { required: true })}              isInvalid={!!errors.Nombre}            />            <Form.Control.Feedback type="invalid">              Este campo es requerido            </Form.Control.Feedback>          </Form.Group>          <Form.Group className="mb-3">            <Form.Label>Precio</Form.Label>            <Form.Control              type="number"              {...register("Precio", { required: true })}              isInvalid={!!errors.Precio}            />            <Form.Control.Feedback type="invalid">              Este campo es requerido            </Form.Control.Feedback>          </Form.Group>          <Form.Group className="mb-3">            <Form.Label>Código de Barra</Form.Label>            <Form.Control              type="text"              {...register("CodigoDeBarra", { required: true })}              isInvalid={!!errors.CodigoDeBarra}            />            <Form.Control.Feedback type="invalid">              Este campo es requerido            </Form.Control.Feedback>          </Form.Group>          <Form.Group className="mb-3">            <Form.Label>Stock</Form.Label>            <Form.Control              type="number"              {...register("Stock", { required: true })}              isInvalid={!!errors.Stock}            />            <Form.Control.Feedback type="invalid">              Este campo es requerido            </Form.Control.Feedback>          </Form.Group>          <Form.Group className="mb-3">            <Form.Label>Fecha de Alta</Form.Label>            <Form.Control              type="date"              {...register("FechaAlta", { required: true })}              isInvalid={!!errors.FechaAlta}            />            <Form.Control.Feedback type="invalid">              Este campo es requerido            </Form.Control.Feedback>          </Form.Group>          <Form.Group className="mb-3">            <Form.Check type="checkbox" label="Activo" {...register("Activo")} />          </Form.Group>          <Button variant="primary" type="submit">            Agregar Producto          </Button>        </Form>      </Container>    );  };  export default ProductForm; |

Este componente representa un formulario para agregar productos. Cuando el formulario es enviado, los datos son enviados al backend a través de una solicitud HTTP POST. La autorización para esta acción se asegura mediante el token de acceso de Keycloak. Si la operación es exitosa, el usuario es redirigido a la página de lista de productos.

-Importación de Dependencias:

* Se importan diversas dependencias como React para crear componentes, useForm de react-hook-form para gestionar formularios de manera eficiente, axios para realizar solicitudes HTTP, y algunos componentes de react-bootstrap para el diseño del formulario.

-Inicialización de Variables y Hooks:

* Se inicializan variables y hooks, como keycloak (para acceder a las funcionalidades de autenticación de Keycloak), navigate (para la navegación programática), y useForm para configurar y gestionar el estado del formulario.

-Definición de la Función onSubmit:

* onSubmit es una función asincrónica que se activa cuando el formulario es enviado.
* La función realiza una solicitud POST a la URL http://localhost:4000/productos con los datos del formulario.
* La autorización para esta solicitud se maneja enviando el token de acceso de Keycloak en el encabezado de la solicitud.
* Si la solicitud es exitosa, el usuario es redirigido a la página de lista de productos (/productos).
* Si hay un error en la solicitud, se muestra un mensaje de error en la consola del navegador.

-Renderizado del Formulario:

* Se renderiza un formulario usando componentes de react-bootstrap.
* Cada campo del formulario (Nombre, Precio, CodigoDeBarra, Stock, FechaAlta, Activo) tiene su propio campo de entrada y validación para asegurar que los datos sean proporcionados correctamente.
* Se utiliza register de react-hook-form para vincular cada campo de entrada con su correspondiente propiedad en el objeto de datos del formulario.
* Cuando el usuario envía el formulario, se activa la función onSubmit.

,

**BACKEND**

Vamos a trabajar con otras dependencias:

|  |
| --- |
| Package.json |
| "dependencies": {  "cors": "^2.8.5",  "keycloak-connect": "^22.0.4",  } |

-La dependencia cors (Cross-Origin Resource Sharing) permite controlar las políticas de acceso desde diferentes orígenes en una aplicación web. Es útil cuando se necesita permitir que recursos en una página web sean solicitados desde otro dominio distinto al que sirve la página web.

-La dependencia keycloak-connect es una biblioteca para Node.js que proporciona integración con el servidor de autenticación y autorización Keycloak. Permite proteger rutas y recursos en una aplicación Express.js para que solo los usuarios autenticados y autorizados puedan acceder a ellos.

|  |
| --- |
| Index.js |
| const express = require('express');  const { keycloak } = require('./config/keycloak');  const productosRoutes = require('./routes/productosRoute');  const cors = require('cors');  const app = express();  app.ue(cors());  // Habilita CORS para todas las rutas  // Middleware para parsear el cuerpo de las solicitudes JSON  app.use(express.json());  // Middleware de Keycloak  app.use(keycloak.middleware()); |

app.use(cors());: Habilita CORS para todas las rutas en la aplicación, permitiendo solicitudes desde cualquier origen.

const { keycloak } = require('./config/keycloak');: Importa la configuración y el objeto keycloak necesario para proteger rutas y gestionar la autenticación y la autorización.

app.use(keycloak.middleware());: Aplica el middleware de Keycloak en la aplicación Express, lo que protege las rutas definidas en productosRoute.js.

|  |
| --- |
| productosRoute.js |
| const express = require('express');  const { keycloak } = require('../config/keycloak');  const productosController = require('../controllers/productosController');  const router = express.Router();  router.get('/', keycloak.protect('realm:read\_only'), productosController.getAllproductos);  router.get('/:id', keycloak.protect('realm:read\_only'), productosController.getProductById);  router.post('/', keycloak.protect('realm:admin'), productosController.createProduct);  router.put('/:id', keycloak.protect('realm:admin'), productosController.updateProduct);  router.delete('/:id', keycloak.protect('realm:admin'), productosController.deleteProduct);  module.exports = router; |

-Las rutas especificadas en productosRoute.js están protegidas por roles y solamente los usuarios con roles específicos (read\_only para las rutas GET y admin para las rutas POST, PUT, DELETE) pueden acceder a ellas.