Universidad de Nariño.

Ingeniería de Sistemas.

Diplomado de actualización en nuevas tecnologías para el desarrollo de Software.

Estudiante: David Alejandro Rodríguez Acosta

Taller Final (Backend y Frontend).

Desarrollar los componentes Backend (Node, Express) y Frontend (React) orientados a desarrollar una aplicación que soporte un hogar de paso de mascotas.

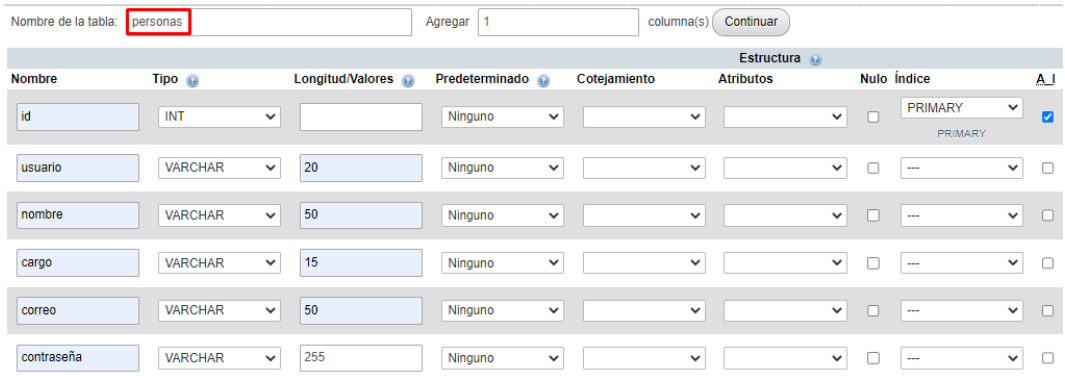
Nota: A diferencia del Taller de Frontend se debe integrar una búsqueda sencilla y se debe permitir autenticar al administrador del sitio para editar o eliminar mascotas.

Desarrollo.

A. Backend

1. En nuestro proyecto Taller Unidad 02 Backend ya tenemos creada la base de datos por eso en esa misma vamos a crear la nueva tabla que nos permitirá hacer uso de la autenticación a la cual llamaremos **“personas”** y tendrá 6 atributos.

   Tabla personas:



   Y con esto tendríamos creada la tabla **“personas”** en nuestra base de datos que permita la autenticación.

2. Posteriormente instalamos las dependencias necesarias:

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Backend\TallerUnidad02Backend> npm i bcrypt
```

Una vez ejecutado el comando revisamos el archivo **package.json** debería quedar de la siguiente manera

package.json > ...

```
{
  "name": "tallerunidad02backend",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "type": "module",
  Debug
  "scripts": {
    "start": "nodemon ./src/app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "nodemon": "^3.0.2"
  },
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mysql2": "^3.6.5",
    "sequelize": "^6.35.2"
  }
}
```

Posteriormente definimos el modelo de datos, para ello dentro de la carpeta **modelos** creamos un archivo llamado **personasModelo.js** para la tabla personas.

modelos > personasModelo.js > ...

```
import Sequelize  from "sequelize";
import {db} from "../database/conexion.js";
import bcrypt from "bcrypt";

const personas = db.define("personas",{
    id:{
        type:Sequelize.INTEGER,
        allowNull: false,
        autoIncrement: true,
        primaryKey: true
    },
    usuario:{
        type: Sequelize.STRING,
        allowNull: false
    },
    nombre:{
        type: Sequelize.STRING,
        allowNull: false
    },
    cargo:{
        type: Sequelize.STRING,
        allowNull:false
    },
    correo:{
        type: Sequelize.STRING,
        allowNull: false,
        unique: true
    },
    contraseña:{
        type: Sequelize.STRING,
        allowNull: false
    }
},{
    hooks: {
        beforeCreate: async (persona) => {
            const hashedContraseña = await bcrypt.hash(persona.contraseña, 10);
            persona.contraseña = hashedContraseña;
        }
    },
    timestamps: false,
    createAt: false,
    updatedAt: false
});

export {personas}
```

Usamos los **"hooks"** de **Sequelize** para ejecutar código antes de crear un nuevo usuario. En este caso, utilizamos **Bcrypt** para hashear la contraseña antes de almacenarla en la base de datos.

Ahora crearemos todos los métodos, para ello en la carpeta **controladores** creamos un archivo llamado **personasController.js** para la tabla personas

controladores > personasController.js > ...

```js
import { personas } from "../modelos/personasModelo.js";
import bcrypt from "bcrypt";

const registrarPersona = async (req, res) => {
    try {
        const { usuario, nombre, cargo, correo, contraseña } = req.body;

        // Verificar si la persona ya existe
        const personaExistente = await personas.findOne({ where: { correo } });
        if (personaExistente) {
            return res.status(400).json({ mensaje: "La persona ya existe." });
        }

        // Crear nueva persona
        const nuevaPersona = await personas.create({ usuario, nombre, cargo, correo, contraseña });

        res.status(201).json({ mensaje: "Persona registrada con éxito.", persona: nuevaPersona });
    } catch (error) {
        console.error(error);
        res.status(500).json({ mensaje: "Error en el servidor." });
    }
};

const autenticarPersona = async (req, res) => {
    try {
        const { correo, contraseña } = req.body;

        // Verificar si la persona existe
        const persona = await personas.findOne({ where: { correo } });
        if (!persona) {
            return res.status(401).json({ mensaje: "Credenciales inválidas." });
        }

        // Verificar la contraseña
        const contraseñaValida = await bcrypt.compare(contraseña, persona.contraseña);
        if (!contraseñaValida) {
            return res.status(401).json({ mensaje: "Credenciales inválidas." });
        }

        res.status(200).json({ mensaje: "Autenticación exitosa." });
    } catch (error) {
        console.error(error);
        res.status(500).json({ mensaje: "Error en el servidor." });
    }
};

export { registrarPersona, autenticarPersona };
```

Ahora dentro de la carpeta **rutas** crearemos el archivo **personasRouter.js**

rutas > personasRouter.js > ...

```js
import express from "express";
import { registrarPersona,autenticarPersona } from "../controladores/personasController.js";

const routerPersonas = express.Router();

// Rutas de autenticación
routerPersonas.post("/registro", (req,res)=>{
    registrarPersona(req,res);
});
routerPersonas.post("/autenticar", (req,res)=>{
    autenticarPersona(req,res);
});

export { routerPersonas };
```

Por último, utilizamos las rutas en la aplicación principal

src > app.js > ...

```
1   import express from "express";
2   import { routerMascotas,routerSolicitud } from "../rutas/mascotasRouter.js";
3   import { routerPersonas } from "../rutas/personasRouter.js";
4   import {db} from "../database/conexion.js";
5   import cors from "cors";
6
7   //Crear Instancia de Express
8   const app = express();
9
10  //Middleware
11  app.use(cors());
12  app.use(express.json());
13  //Verificar Conexion a Base de Datos
14  db.authenticate().then(()=>{
15      console.log(`Base de Datos conectada de manera exitosa`);
16  }).catch(err=>{
17      console.log(`Error al conectarse a la Base de Datos ::: ${err}`);
18  })
19
20  //Definir Rutas
21  app.get("/",(req,res)=>{
22      res.send("Hola Backend Mysql");
23  });
24
25  //Rutas
26  app.use("/mascotas",routerMascotas);
27  app.use("/solicitudes",routerSolicitud);
28  app.use("/autenticacion",routerPersonas);
```

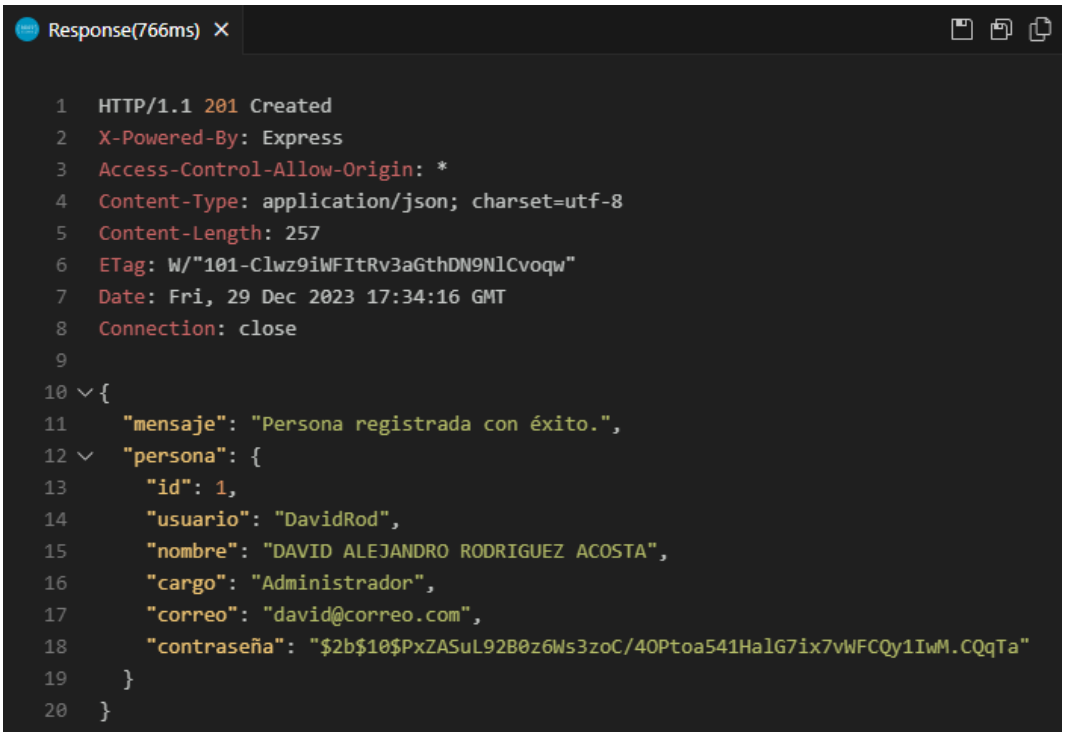
3. A continuación, en el archivo **requests.txt** creamos el usuario administrador para que pueda iniciar sesión en la aplicación Frontend

   Petición realizada:

```
###
Send Request
POST http://localhost:8000/autenticacion/registro HTTP/1.1
Content-Type: application/json

{
    "usuario": "DavidRod",
    "nombre": "DAVID ALEJANDRO RODRIGUEZ ACOSTA",
    "cargo": "Administrador",
    "correo": "david@correo.com",
    "contraseña": "david123"
}
```

Respuesta recibida:

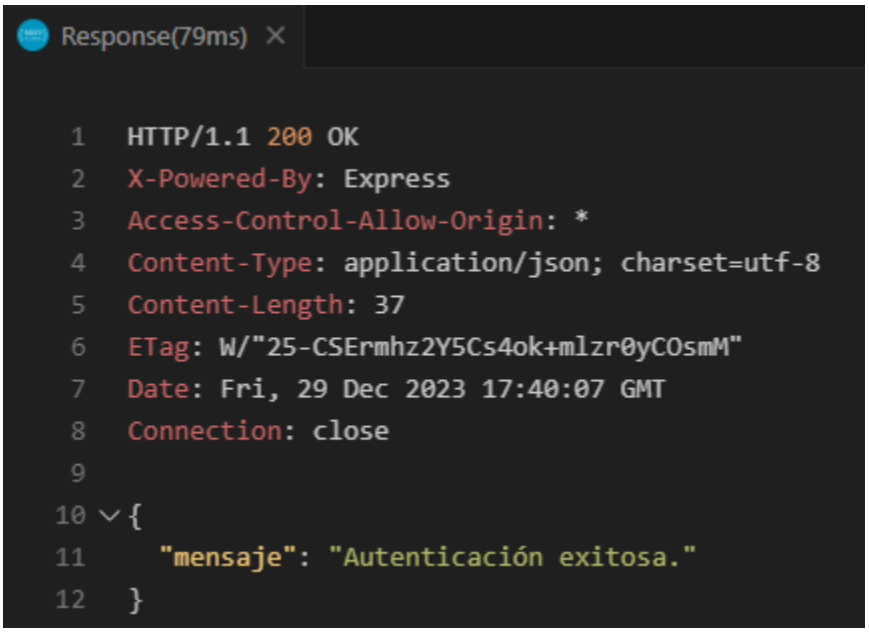


Podemos ver que el usuario ha sido creado y la contraseña ha sido encriptada para una mayor seguridad de los datos.

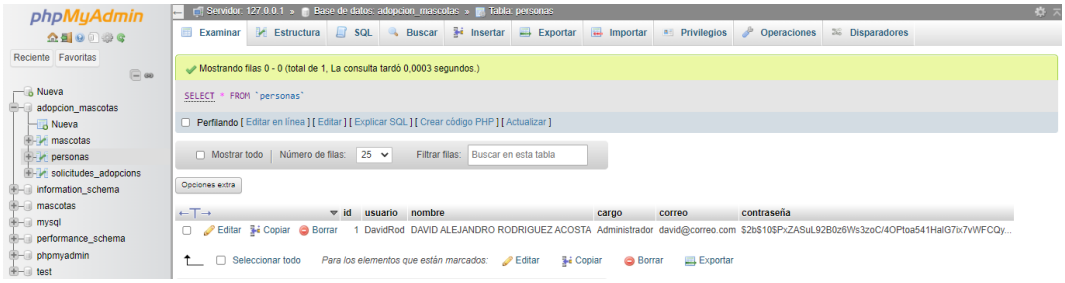
Ahora vamos a probar si el usuario puede autenticarse.

Petición realizada:

Respuesta recibida:



Y ahora verifiquemos si en realidad el usuario existe en la base de datos.



Efectivamente el dato existe en nuestra base de datos.

B. Frontend

Para permitir la autenticación del usuario administrador mediante la vista vamos a necesitar crear un nuevo componente el cual llamaremos **PersonasComponent.js**

Importamos los recursos necesarios

unidad-03-mascotas > src > Components > PersonasComponent.js > PersonasComponent

```
1   //IMPORT
2   import React, { useEffect, useState } from "react";
3   import axios from "axios";
4   import { mostrarAlerta } from "../functions.js";
5   import '../estilos.css'
```

Ahora vamos a necesitar variables que nos permita manejar la lógica de inicio de sesión.

```
//CUERPO COMPONENTE
const PersonasComponent = () => {
  const url = "http://localhost:8000/autenticacion";
  const [correo, setCorreo] = useState("");
  const [contraseña, setPassword] = useState("");
  const [autenticado, setAutenticado] = useState(false);
```

Podemos ver que la variable **autenticado** no hace parte de la base de datos sin embargo esta variable nos va a permitir manejar la lógica de la autenticación guardando la autenticación en un almacenamiento local.

```
14    useEffect(() => {
15      const storedAuth = localStorage.getItem("autenticado");
16      if (storedAuth) {
17        setAutenticado(JSON.parse(storedAuth));
18      }
19    }, []);
```

localStorage.getItem("autenticado") obtiene el valor almacenado en el **localStorage** bajo la clave **"autenticado"**. **localStorage** es una forma de almacenar datos en el navegador web de manera persistente y devuelve null si no hay ningún valor almacenado bajo esa clave.

Adicionalmente, el **“if (storedAuth)”** verifica si storedAuth (el valor obtenido del localStorage) existe y no es null. Si existe, entonces **“setAutenticado(JSON.parse(storedAuth))”** significa que hay un valor almacenado. **JSON.parse** se utiliza para convertir el valor almacenado (que normalmente es una cadena) en un objeto o valor JavaScript, y luego **setAutenticado** se llama para actualizar el estado del componente con ese valor.

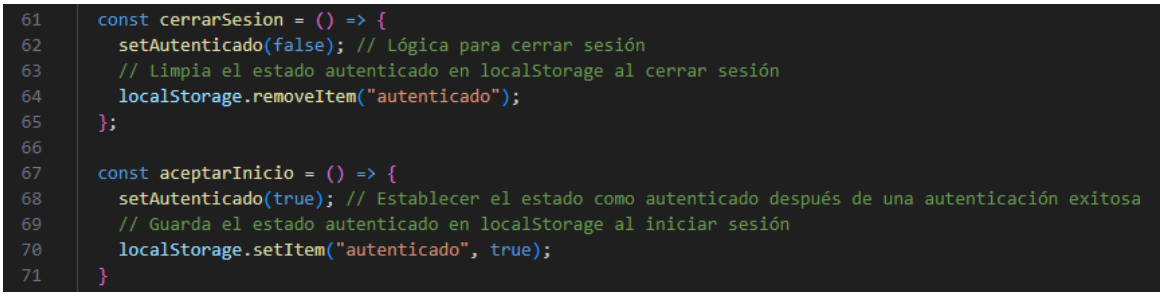
Posteriormente copiamos la función de enviar solicitud que fue realizada en clases

```
21  const enviarSolicitud = async (metodo, parametros)=>{
22    await axios({method: metodo, url: parametros.urlExt, data: parametros })
23    .then((respuesta)=>{
24        let tipo= respuesta.data.tipo;
25        let mensaje = respuesta.data.mensaje;
26        mostrarAlerta(mensaje,tipo);
27        if(tipo === "success"){
28            aceptarInicio();
29        }
30    })
31    .catch((error)=>{
32        mostrarAlerta(`Error en la solicitud`,error)
33    });
34  };
```

Cabe resaltar que después de que la solicitud sea exitosa llama a la función **aceptarInicio()** debido a que necesitamos establecer la variable **"autenticado"** en true para poder mostrar contenido de pendiendo del caso

```
36  const validar = ()=>{
37    let parametros;
38    let metodo;
39    if(correo.trim()===''){
40        console.log("Debe escribir un Correo");
41        mostrarAlerta("Debe escribir un Correo");
42    }
43    else if(contraseña.trim()===''){
44        console.log("Debe escribir una Contraseña");
45        mostrarAlerta("Debe escribir una Contraseña");
46    }
47    else{
48        parametros={
49            urlExt: `${url}/autenticar`,
50            correo: correo.trim(),
51            contraseña: contraseña.trim(),
52        };
53        metodo="POST";
54
55        enviarSolicitud(metodo, parametros);
56
57    }
58
59  };
```

En la función validar simplemente validamos los campos que no esten vacíos para evitar errores en el servidor, adicionalmente si no hay errores pues enviamos la solicitud con los parámetros, podemos ver que en url está el “End Point” de nuestro Backend



```
61  const cerrarSesion = () => {
62    setAutenticado(false); // Lógica para cerrar sesión
63    // Limpia el estado autenticado en localStorage al cerrar sesión
64    localStorage.removeItem("autenticado");
65  };
66
67  const aceptarInicio = () => {
68    setAutenticado(true); // Establecer el estado como autenticado después de una autenticación exitosa
69    // Guarda el estado autenticado en localStorage al iniciar sesión
70    localStorage.setItem("autenticado", true);
71  }
```

Como se puede observar en la imagen anterior están las funciones **cerrarSesion** que la vamos a utilizar en las demás vistas, y también está la función que utilizábamos en la función **enviarSolicitud** después de que la autenticación fuera exitosa.

```
88                  {autenticado ? (
89                      <li class="nav-item dropdown">
90                      <a class="nav-link dropdown-toggle" role="button" data-bs-toggle="dropdown" aria-e
91                          Mascota
92                      </a>
93                      <ul class="dropdown-menu">
94                          <li><a class="dropdown-item" href="/mascotas">Registrar</a></li>
95                          <li><a class="dropdown-item" href="/">Adoptar</a></li>
96                          <li><hr class="dropdown-divider"></hr></li>
97                          <li><a class="dropdown-item" href="/visualizar">Solicitudes</a></li>
98                      </ul>
99                      </li>
100                 ):(
101                     <li class="nav-item">
102                         <a class="nav-link" aria-current="page" href="/">Adoptar</a>
103                     </li>
104                 )}
```

De esta manera podemos mostrar el contenido de acuerdo si el usuario esta registrado o no, lo que quiere decir es que si el usuario esta autenticado (autenticado es true) entonces va a mostrar en la barra de navegación las opciones de registrar mascotas, adoptar mascotas y ver las solicitudes de las adopciones. Pero si no ha iniciado sesión (autenticado es false) entonces solo mostrara al usuario el botón de adoptar.

De la misma forma pasa con la imagen del usuario

```
{autenticado ? (
    <ul className="navbar-nav">
        <li className="nav-item dropdown">
            <a className="nav-link dropdown-toggle" role="button" data-bs-toggle="dropdown" aria-expanded="false">
                <div className="d-flex align-items-center">
                    <img src="https://cdn-icons-png.flaticon.com/512/236/236831.png" alt="Logo" height="50" />
                </div>
            </a>
            <ul className="dropdown-menu dropdown-menu-end">
                <li><a className="dropdown-item" href="/login">Mi Perfil</a></li>
                <li><a className="dropdown-item" href="#">Configuraciones</a></li>
                <li><hr className="dropdown-divider"></hr></li>
                <li><a className="dropdown-item" href="#" onClick={() => cerrarSesion()}>Cerrar Sesión</a></li>
            </ul>
        </li>
    </ul>
) : (
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link" href="/login">
                <img src="https://cdn-icons-png.flaticon.com/512/1177/1177568.png" alt="Logo" height="50" />
            </a>
        </li>
    </ul>
)}
```

Con respecto al contenido es igual

```
{autenticado ? (
    <div>
        {/* Contenido para usuarios autenticados */}
        <br></br>
        <h2 style={{ textAlign: "center" }} >Bienvenido(a)</h2>
        <br></br>
        <br></br>
        <br></br>
        <button
        type="button"
        className="btn btn-danger btn-block mb-4"
        onClick={() => cerrarSesion()}
        >
        Cerrar Sesión
        </button>
    </div>
) : (
    <section class=" text-center text-lg-start">
    {/* Contenido para usuarios NO autenticados */}
    <div class="card mb-3">
        <div class="row g-0 d-flex align-items-center">
            <div class="col-lg-4 d-none d-lg-flex">
                <img src="https://img.freepik.com/fotos-premium/retr
                className="w-100 rounded-t-5 rounded-tr-lg-0 rounded
            </div>
```

Cabe resaltar que para utilizar esta manera de controlar lo que se puede mostrar al usuario se necesita empaquetar todo el contenido, en este caso esta todo empaquetado en un **div** en el otro caso esta empaquetado en un **section** y en otros casos esta empaquetado en un **li**

Ahora para poder recuperar si el usuario esta autenticado o no en los demás componentes hacemos lo siguiente

Definimos la variable

```
const [autenticado, setAutenticado] = useState(false);
```

Luego llamamos a la función que nos permitirá verificar si esta autenticado

```
23    useEffect(() => {
24      getMascotas();
25      checkAuthentication();
26    }, []);
```

La función a la cual se llama es esta

```
34  const checkAuthentication = () => {
35    const storedAuth = localStorage.getItem("autenticado"); //Recuperar el estado de la variable segun sea el caso
36    if (!storedAuth) { //Condicional para saber si esta autenticado o no
37      console.log("Usuario no autenticado en MascotasComponent");
38      setAutenticado(false);
39      navigate('/login'); //Redirigir al usuario no autenticado al login
40    }
41  };
```

Puesto que debemos proteger las rutas hemos puesto un navigate esto con el fin de que el inicio de sesión sirva para algo

Para lograr esto primero importamos el recurso

```
7  import { useNavigate } from 'react-router-dom';
```

Posteriormente definimos la variable que va a utilizar este recurso

```
21    const navigate = useNavigate();
```

Y después la llamamos dentro de la función anterior, ahora utilizamos la misma función de cerrar sesión

```
const cerrarSesion = () => {
  setAutenticado(false); // Lógica para cerrar sesión
  // Limpia el estado autenticado en localStorage al cerrar sesión
  localStorage.removeItem("autenticado");
  checkAuthentication();
};
```

Pero en este caso dentro de ella llamamos a **checkAuthentication** con el fin de que cuando este viendo este componente y cierre sesión dentro de este mismo no se quede como intruso, sino que sea redireccionado al login inmediatamente después de cerrar la sesión.

Cabe aclarar que el componente MascotasComponent y SolicitudesComponent están protegidos por medio de las rutas, por esta razón no se utiliza lógica para mostrar contenido de la foto o enlaces de la barra de navegación.

Ahora en el AdopcionComponent si necesitamos utilizar lógica de inicio de sesión debido a que a este componente puede ingresar un usuario cualquiera por lo tanto hacemos lo mismo que los demás componentes

Importamos los recursos necesarios

```
7  import { useNavigate } from 'react-router-dom';
```

Definimos las variables para el control de autenticación

```
27    //Inicio de sesion
28    const [autenticado, setAutenticado] = useState(false);
29    const navigate = useNavigate();
```

Llamamos a la función de verificación de inicio de sesión

```
31    useEffect(() => {
32      getMascotas();
33      checkAuthentication();
34    }, []);
```

Creamos la función de verificación de inicio de sesion

```
42    const checkAuthentication = () => {
43      const storedAuth = localStorage.getItem("autenticado");
44      if (!storedAuth) {
45        console.log("Usuario no autenticado en MascotasComponent");
46        setAutenticado(false);
47      }else{
48        setAutenticado(true);
49      }
50    };
```

Podemos ver que la función ahora tiene un **else** pues este es para poder mostrar el contenido dependiendo sea el caso

```
155     const cerrarSesion = () => {
156       setAutenticado(false); // Lógica para cerrar sesión
157       // Limpia el estado autenticado en localStorage al cerrar sesión
158       localStorage.removeItem("autenticado");
159       navigate('/');
160     };
```

Adicionalmente creamos la función **cerrarSesion** pero podemos ver que ahora esta redirecciona al inicio, pues el (‘/’) es la ruta definida para el componente AdopcionComponent, es decir va actualizar la pagina con el fin de mostrar el contenido según sea el caso.

```
177         {autenticado ? (
178             <li class="nav-item dropdown">
179             <a class="nav-link dropdown-toggle" role="button" data-bs-toggle="dropdown" aria-expanded="false">
180                 Mascota
181             </a>
182             <ul class="dropdown-menu">
183                 <li><a class="dropdown-item" href="/mascotas">Registrar</a></li>
184                 <li><a class="dropdown-item" href="/">Adoptar</a></li>
185                 <li><hr class="dropdown-divider"></hr></li>
186                 <li><a class="dropdown-item" href="/visualizar">Solicitudes</a></li>
187             </ul>
188             </li>
189         ):(
190             <li class="nav-item">
191                 <a class="nav-link" aria-current="page" href="/">Adoptar</a>
192             </li>
193         )}
```

Y manejamos el contenido de la misma manera que en el componente PersonasComponent

```
201   {autenticado ? (
202       <ul className="navbar-nav">
203           <li className="nav-item dropdown">
204               <a className="nav-link dropdown-toggle" role="button" data-bs-toggle="dropdown" aria-expanded="false">
205                   <div className="d-flex align-items-center">
206                       <img src="https://cdn-icons-png.flaticon.com/512/236/236831.png" alt="Logo" height="50" />
207                   </div>
208               </a>
209               <ul className="dropdown-menu dropdown-menu-end">
210                   <li><a className="dropdown-item" href="/login">Mi Perfil</a></li>
211                   <li><a className="dropdown-item" href="#">Configuraciones</a></li>
212                   <li><hr className="dropdown-divider"></hr></li>
213                   <li><a className="dropdown-item" href="#" onClick={() => cerrarSesion()}>Cerrar Sesión</a></li>
214               </ul>
215           </li>
216       </ul>
217   ) : (
218       <ul class="navbar-nav">
219           <li class="nav-item">
220               <a class="nav-link" href="/login">
221                   <img src="https://cdn-icons-png.flaticon.com/512/1177/1177568.png" alt="Logo" height="50" />
222               </a>
223           </li>
224       </ul>
225   )}
```

Y así manejamos la foto que se muestra en la barra de navegación según sea el caso.

Por último, especificamos las rutas en el archivo App.js

unidad-03-mascotas > src > App.js > App

```
import {BrowserRouter,Routes,Route} from 'react-router-dom';
import MascotasComponent from './Components/MascotasComponent';
import AdopcionComponent from './Components/AdopcionComponent';
import SolicitudesComponent from './Components/SolicitudesComponent';
import PersonasComponent from './Components/PersonasComponent';
import './App.css';



function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path = '/mascotas' element = {<MascotasComponent></MascotasComponent>}></Route>
        <Route path = '/' element = {<AdopcionComponent></AdopcionComponent>}></Route> {/*Nueva ruta para la a
        <Route path = '/visualizar' element = {<SolicitudesComponent></SolicitudesComponent>}></Route> {/*Nueva
        <Route path = '/login' element = {<PersonasComponent></PersonasComponent>}></Route> {/*Nueva ruta para
      </Routes>
    </BrowserRouter>
  );
}

export default App;

```

Implementación de la barra de búsqueda.

1. **AdopcionComponent.js**

   Para esto vamos a definir una variable que nos permita cambiar el estado

```
//Barra de búsqueda
const [filtro, setFiltro] = useState("");
```

   Ahora vamos a modificar la función de renderizado para mostrar solo las mascotas que coincidan con el término de búsqueda y que obviamente estén disponibles

```
//Filtro para mostrar mascotas de acuerdo a la disponibilidad, se muestran los que estan disponibles (true)
const mascotasDisponibles = mascotas.filter(mascota => mascota.disponible);

//Fitro para mostrar mascotas de acuerdo a la disponibilidad y de acuerdo a la barra de busqueda
const mascotasFiltradas = mascotasDisponibles.filter(
  (mascota) =>
    mascota.nombre.toLowerCase().includes(filtro.toLowerCase()) ||
    mascota.especie.toLowerCase().includes(filtro.toLowerCase())
);
```

   Cabe aclarar que la barra de búsqueda solo busca por nombre y especie, pero se puede adicionar más filtros

En el input de la búsqueda agregamos lo siguiente

```
<div class="d-flex justify-content-center">
    <form class="d-flex" role="search">
        <input
        class="form-control me-2"
        type="search"
        placeholder="Search"
        aria-label="Search"
        value={filtro}
        onChange={(e) => setFiltro(e.target.value)}
        ></input>
        <button class="btn btn-outline-success" type="submit">Search</button>
    </form>
</div>
```

Después actualizamos el mapeo de las mascotas con **mascotasFiltradas** en la sección del contenedor de mascotas ya que anteriormente estaba utilizando **mascotasDisponibles** porque solo necesitaba ver mascotas disponibles.

```
{mascotasFiltradas.map((mascota, i) => (
```

2. MascotasComponent.js

Para esto vamos a definir una variable que nos permita cambiar el estado

```
//Barra de búsqueda
const [filtro, setFiltro] = useState("");
```

Ahora vamos a crear la función de renderizado para mostrar solo las mascotas que coincidan con el término de búsqueda

```
const mascotasFiltradas = mascotas.filter(
  (mascota) =>
    mascota.nombre.toLowerCase().includes(filtro.toLowerCase()) ||
    mascota.especie.toLowerCase().includes(filtro.toLowerCase())
);
```

Aquí también la barra de búsqueda filtra por nombre y especie, pero también podríamos agregar más filtros con el condicional OR (“**||**”)

En el input de la búsqueda agregamos lo siguiente

```
<div class="d-flex justify-content-center">
    <form class="d-flex" role="search">
        <input
        className="form-control me-2"
        type="search"
        placeholder="Search"
        aria-label="Search"
        value={filtro}
        onChange={(e) => setFiltro(e.target.value)}
        ></input>
        <button class="btn btn-outline-success" type="submit">Search</button>
    </form>
</div>
```

Y modificamos el mapeo ya que estaba utilizando **mascotas** lo reemplazamos por la función de renderizado

```
247                    {mascotasFiltradas.map((mascota, i) => (
```

3. **SolicitudesComponent.js**

Definimos una variable que nos permita cambiar el estado

```
22      //Barra de búsqueda
23      const [filtro, setFiltro] = useState("");
```

Ahora vamos a crear la función de renderizado para mostrar solo las mascotas que coincidan con el término de búsqueda

```
87    const solicitudesFiltradas = solicitudes.filter(
88      (solicitud) =>
89        solicitud.nombre_solicitante.toLowerCase().includes(filtro.toLowerCase()) ||
90        solicitud.estado.toLowerCase().includes(filtro.toLowerCase())
91    );
```

Aquí la barra de búsqueda filtra por nombre de solicitante y estado de la solicitud, pero también podríamos agregar más filtros con el condicional OR (“||”)

En el input de la búsqueda agregamos lo siguiente

```
120                 <div class="d-flex justify-content-center">
121                     <form class="d-flex" role="search">
122                         <input
123                         className="form-control me-2"
124                         type="search"
125                         placeholder="Search"
126                         aria-label="Search"
127                         value={filtro}
128                         onChange={(e) => setFiltro(e.target.value)}
129                         ></input>
130                         <button class="btn btn-outline-success" type="submit">Search</button>
131                     </form>
132                 </div>
```

Por último, modificamos el mapeo ya que estaba utilizando **solicitudes** lo cual debemos reemplazamos por la función de renderizado en este caso **solicitudesFiltradas**

```
165                    {solicitudesFiltradas.map((solicitud, i) => (
```

Observación: El botón **search** no tiene funcionalidad debido a que a medida que se va escribiendo en el input automáticamente la aplicación va aplicando los filtros debido al evento **onChange**.