Universidad de Nariño.

Ingeniería de Sistemas.

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

Estudiante: David Alejandro Rodríguez Acosta

Taller Unidad 3 Frontend.

Desarrollar una aplicación Frontend en React que haga uso del Backend desarrollado en el taller anterior (Taller Unidad 2 Backend), elaborar un informe que detalle el proceso de construcción y la implementación del aplicativo.

Para el desarrollo de esta actividad primero creamos el proyecto Frontend asi:

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend> npx create-react-app unidad-03-mascotas
```

Ahora instalamos las dependencias que vamos a necesitar para poder desarrollar nuestra aplicación.

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend\unidad-03-mascotas> npm i axios
```

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend\unidad-03-mascotas> npm i bootstrap
```

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend\unidad-03-mascotas> npm i fortawesome
```

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend\unidad-03-mascotas> npm i react-router-dom
```

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend\unidad-03-mascotas> npm i @fortawesome/fontawesome-free
```

```
PS C:\Users\DAVID\Downloads\Diplomado\Talleres Frontend\TallerUnidad03Frontend\unidad-03-mascotas> npm i sweetalert2-react-content
```

Retomando todo lo realizado en clase, empezamos a añadir las nuevas características que va a tener el proyecto, adicionalmente en nuestro proyecto de backend instalamos una dependencia adicional:

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

Posteriormente editamos el archivo **app.js** de nuestro proyecto Backend adicionando esto:

app.js M

src > app.js > catch() callback

```
1   import express from "express";
2   import { routerMascotas,routerSolicitud } from "../rutas/mascotasRouter.js";
3   import {db} from "../database/conexion.js";
4   import cors from "cors";
5
6   //Crear Instancia de Express
7   const app = express();
8
9   //Middleware
10  app.use(cors());
11  app.use(express.json());
```

De esta forma vamos a poder consumir el api de nuestro proyecto Backend ya creado anteriormente.

Ahora vamos a editar el archivo **MascotasComponent.js** el cual fue creado en clases, para poder registrar una mascota con los campos requeridos por la base de datos creada en el proyecto de Backend vamos a necesitar agregar el campo para especie y el campo disponible así:

Definimos las variables

```
//CUERPO COMPONENTE
const MascotasComponent = () => {
  const url = "http://localhost:8000/mascotas";
  const [mascotas, setMascotas] = useState([]);
  const [id, setId] = useState("");
  const [nombre, setNombre] = useState("");
  const [especie, setEspecie] = useState("");
  const [edad, setEdad] = useState("");
  const [disponible, setDisponible] = useState("");
  const [operacion, setOperacion] = useState("");
  const [titulo,setTitulo]=useState("");
```

Y dentro de la función que abre el cuadro para el registro y actualización las añadimos asi:

```
const openModal =(opcion, id, nombre, especie, edad, disponible)=>{
  setId('');
  setNombre('');
  setEspecie('Especie');
  setEdad('');
  setDisponible(true);
  setOperacion(opcion);
  if(opcion === 1){
      setTitulo("Registrar Mascota");
  }
  else if(opcion===2){
      setTitulo("Editar Mascota");
      setId(id);
      setNombre(nombre);
      setEspecie(especie);
      setEdad(edad);
      setDisponible(disponible);
  }
};
```

En los “set” de especie y disponible se puede observar que hemos realizado la asignación de valores por defecto para que nos facilite el uso del formulario modal

Posteriormente validamos estos valores para que no haya errores en la base de datos así:

```
const validar = ()=>{
  let parametros;
  let metodo;
  if(nombre.trim()===''){
      console.log("Debe escribir un Nombre");
      mostrarAlerta("Debe escribir un Nombre");
  }
  else if(edad.trim()===''){
      console.log("Debe escribir una Edad");
      mostrarAlerta("Debe escribir una Edad");
  }
  else if(especie.trim()===''){
      console.log("Debe escribir una Especie");
      mostrarAlerta("Debe escribir una Especie");
  }
  else{
      if(operacion===1){
          parametros={
              urlExt: `${url}/crear`,
              nombre: nombre.trim(),
```

En caso de ser correcto entonces guardamos los datos en la base de datos según sea el caso así:

```
else{
    if(operacion===1){
        parametros={
            urlExt: `${url}/crear`,
            nombre: nombre.trim(),
            especie: especie.toString(),
            edad: edad.trim(),
            disponible: disponible.toString()
        };
        metodo="POST";
    }
    else{
        parametros={
            urlExt: `${url}/actualizar/${id}`,
            nombre: nombre.trim(),
            especie: especie.toString(),
            edad: edad.trim(),
            disponible: disponible.toString()
        };
        metodo="PUT";
    }
    enviarSolicitud(metodo, parametros);
```

Colocamos la palabra reservada “toString()” debido a que es necesario convertir las entradas del formulario a cadenas de caracteres para poder ser guardadas en la base de datos.

A continuación, para poder visualizar todos los atributos que están disponibles en la base de datos hacemos lo siguiente:

```
151                 <table className="table table-bordered">
152                   <thead>
153                     <tr>
154                       <th>#</th>
155                       <th>NOMBRE</th>
156                       <th>ESPECIE</th>
157                       <th>EDAD</th>
158                       <th>DISPONIBLE</th>
159                     </tr>
160                   </thead>
161                   <tbody className="table-group-divider">
162                     {mascotas.map((mascota, i) => (
163                       <tr key={mascota.id}>
164                         <td>{mascota.id}</td>
165                         <td>{mascota.nombre}</td>
166                         <td>{mascota.especie}</td>
167                         <td>{mascota.edad}</td>
168                         <td>{mascota.disponible ? 'Sí' : 'No'}</td>
```

Hemos añadido los títulos a la tabla y posteriormente mostramos los dos atributos faltantes de la tabla, como el atributo **disponible** es de tipo **buleano** le pusimos esa condición para que pueda ser legible de manera más clara.

Ahora en el botón editar debemos pasar estos atributos para que puedan ser mostrados en el cuadro flotante de registro/actualización por lo tanto lo hacemos lo siguiente:

```
<td>
  <button
    onClick={()=>openModal(2,mascota.id,mascota.nombre,mascota.especie,mascota.edad,mascota.disponible)}
    className="btn btn-warning"
    data-bs-toggle="modal"
    data-bs-target="#modalMascotas"
  >
    <i className="fa-solid fa-edit"></i>
  </button>
</td>
```

Ahora necesitamos editar el cuadro en donde se va a registrar/actualizar las mascotas, es decir tenemos que agregar estos dos atributos faltantes.

Nos dirigimos hacia donde está el modal:

```
<div id="modalMascotas" className="modal fade" aria-hidden="true">
```

Dentro de él añadimos lo siguiente:

Para el campo especie:

```
<div className="input-group mb-3">
  <span className="input-group-text">
    <i className="fa-solid fa-gift"></i>
  </span>
  <select
    id="especie"
    className="form-control"
    placeholder="Especie"
    onChange={(e)=>setEspecie(e.target.value)}

  >
    <option value={especie}>{especie}</option>
    <option value={'Perro'}>Perro</option>
    <option value={'Gato'}>Gato</option>
  </select>
</div>
```

Para el campo disponible:

```
<div className="input-group mb-3">
  <span className="input-group-text">
    <i className="fa-solid fa-gift"></i>
  </span>
  <input
    type="text"
    id="disponible"
    className="form-control"
    value={disponible === true ? "Sí" : "No"}
    readOnly
  ></input>
</div>
```

Al momento de registrar a la mascota, esta entra inmediatamente a estar disponible para su adopción, adicionalmente hemos puesto el valor por defecto **Especie** para que pueda ser visualizado como primera opción en el formulario modal. por lo tanto, en la parte de la función que abre el cuadro modal, modificamos lo siguiente:

```
const openModal =(opcion, id, nombre, especie, edad, disponible)=>{
  setId('');
  setNombre('');
  setEspecie('Especie');
  setEdad('');
  setDisponible(true);
  setOperacion(opcion);
  if(opcion === 1){
      setTitulo("Registrar Mascota");
  }
  else if(opcion===2){
      setTitulo("Editar Mascota");
      setId(id);
      setNombre(nombre);
      setEspecie(especie);
      setEdad(edad);
      setDisponible(disponible);
  }
};
```

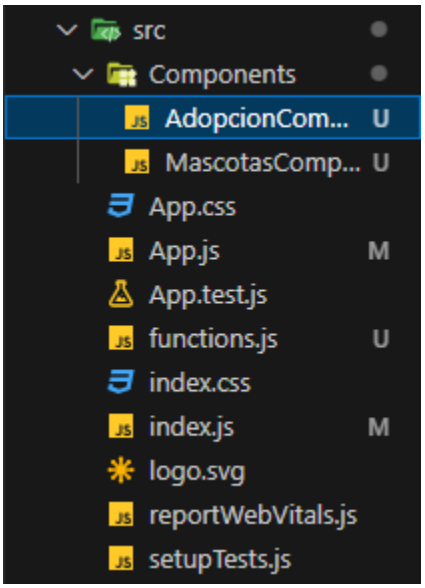
Se le añade el **true** puesto que la mascota debe estar disponible hasta cuando la adopten ahí debería cambiar su estado, además nadie puede editar su estado por lo mencionado anteriormente.

Ahora vamos a permitirle al usuario adoptar la mascota que solamente este disponible de lo contrario no debería aparecer, para ello vamos a crear una nueva ruta:

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

```
1  import {BrowserRouter,Routes,Route} from 'react-router-dom';
2  import MascotasComponent from './Components/MascotasComponent';
3  import './App.css';
4
5  function App() {
6    return (
7      <BrowserRouter>
8        <Routes>
9          <Route path = '/' element = {<MascotasComponent></MascotasComponent>}></Route>
10         <Route path = '/adoptar' element = {<AdopcionComponent></AdopcionComponent>}></Route> {/*Nueva ruta
11       </Routes>
12     </BrowserRouter>
13   );
14 }
15
16 export default App;
```

A continuación, vamos a crear dentro de la carpeta **Components** el archivo **AdopcionComponent.js**



Dentro del archivo **AdopciónComponent.js** vamos a definir las variables para la tabla solicitudes_adopcion

```
//CUERPO COMPONENTE
const AdopcionComponent = () => {
  const url = "http://localhost:8000/mascotas";
  const [mascotas, setMascotas] = useState([]);
  const [id, setId] = useState("");
  const [nombre, setNombre] = useState("");
  const [especie, setEspecie] = useState("");
  const [edad, setEdad] = useState("");
  const [disponible, setDisponible] = useState("");
  const [operacion, setOperacion] = useState("");
  const [titulo,setTitulo]=useState("");
  //Tabla solicitudes_adopcion
  const urlS = "http://localhost:8000/solicitudes";
  const [solicitudes, setSolicitudes] = useState([]);
  const [idS, setIdS] = useState("");
  const [nombreS, setNombreS] = useState("");
  const [idMascota, setIdMascota] = useState("");
  const [estado, setEstado] = useState("");
```

Posteriormente creamos un nuevo formulario modal para el registro de las solicitudes de adopción:

```
const openModal =(opcion, idMascota, nombre, nombreS, idS, estado)=>{
  setIdS('');
  setNombreS('');
  setIdMascota(idMascota);
  setEstado('Pendiente');
  setOperacion(opcion);
  if(opcion === 1){
      setTitulo("Registar Solicitud");
      setDisponible(false) //Establecer valor default
      setNombre(nombre) //Para mostrar nombre en el modal
  }
  else if(opcion===2){
      setTitulo("Editar Solicitud");
      setIdS(idS);
      setNombreS(nombreS);
      setIdMascota(idMascota);
      setEstado(estado);
  }
};
```

Posteriormente en la función **"validar()"** vamos a validar los datos que se deben ingresar por teclado y vamos a crear una nueva función que nos permita cambiar el atributo disponible de la tabla mascotas

```
const validar = async ()=>{
  let parametros;
  let metodo;
  if(nombreS.trim()===''){
      console.log("Debe escribir un Nombre");
      mostrarAlerta("Debe escribir un Nombre");
  }
  else{
      if(operacion===1){
          parametros={
              urlExt: `${urlS}/crear`,
              nombre_solicitante: nombreS.trim(),
              id_mascota: idMascota.toString(),
              estado: estado.trim()
          };
          metodo="POST";

          mascotasDisponible(); //Actualizar atributo de disponible
      }
      else{
```

```
const mascotasDisponible = async ()=>{
  let parametros;
  let metodo
  parametros={
      urlExt: `${url}/actualizar/${idMascota}`,
      disponible: disponible.toString() //Asignación del valor default para guardarlo en la base de datos
  };
  metodo="PUT";

  await enviarSolicitud(metodo, parametros);
};
```

Posteriormente crearemos un aviso utilizando **SweetAlert2** para que nos muestre cierta información de acuerdo a la especie si es perro o gato según corresponda

```
const infoMascota=(nombre,especie,edad)=>{
  const MySwal = withReactContent(Swal);
  let textoEspecie = ''; //Inicializar variable para guardar el texto
  //Condicional para que segun la especie se muestre cierto tipo de mensaje en el boton info
  if (especie === 'Perro') {
      textoEspecie = 'Son mascotas muy leales, protectores y juguetones. Los perros son animales soci
  }
  else {
    textoEspecie = 'Son muy independientes, curiosos y cariñosos. Los gatos son animales sociales y n
  }
  MySwal.fire({
      title: `${nombre} | ${edad} años`,
      icon: 'info',
      text: textoEspecie,
      showCloseButton: true,
      confirmButtonText: `<i class="fa fa-thumbs-up"></i> Entendido!`
  })
}
```

Después creamos un filtro que nos permita visualizar solamente las mascotas que están disponibles para la adopción

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

Ahora en la parte de la vista lo utilizamos para generar las cartas de solo las mascotas disponibles

```
<div class="row">
    {mascotasDisponibles.map((mascota, i) => (
        <div class="col">
            <br></br>
            <div class="card" style={{width: '18rem'}}>
                <img src="https://img.freepik.com/vector-gratis/ilustracion-dibujos-animados-lindo-perro-g
                <div class="card-body">
                    <h5 class="card-title">{mascota.nombre} | {mascota.edad} años</h5>
                    <h6 class="card-subtitle mb-2 text-body-secondary">{mascota.especie}</h6>
                    <p class="card-text">{mascota.especie === 'Perro' ? 'Son mascotas muy leales, protec..
                    <div class="row">
                        <div class="col">
                            <button
                                onClick={()=>infoMascota(mascota.nombre,mascota.especie,mascota.edad)}
                                className="btn btn-info"
                            >
                                <i className="fa-solid fa-circle-info"></i> Detalles
                            </button>
                        </div>
                        <div class="col">
                            <button
                                onClick={()=>openModal(1,mascota.id, mascota.nombre)}
```

Adicionalmente en el modalSolicitudes hacemos lo siguiente

```
<div className="input-group mb-3">
  <span className="input-group-text">
    <i className="fa-solid fa-gift"></i>
  </span>
  <input
    type="text"
    id="idMascota"
    className="form-control"
    //Mostramos el nombre de la mascota en el form pero no guardamos en la base de datos
    value={nombre}
    readOnly
  ></input>
</div>
```

Puesto que no guardamos el nombre en la base de datos sino el id de la mascota por esta razón lo asignamos en la función openModal dentro del set y lo establecemos como valor predeterminado según la carta que el usuario escoja.

El id de la mascota se guarda en la variable idMascota asi:

```
<div class="col">
  <button
      onClick={()=>openModal(1,mascota.id, mascota.nombre)}
      className="btn btn-outline-success"
      data-bs-toggle="modal"
      data-bs-target="#modalSolicitudes"
  >
      <i className="fa-solid fa-square-plus"></i> Adoptar
  </button>
</div>
```

Podemos ver que estamos pasando el id de mascota y el nombre, como es importante mantener el orden al momento de pasar valores en las funciones por esta razón podemos ver en el modal el orden en que se cargó las variables

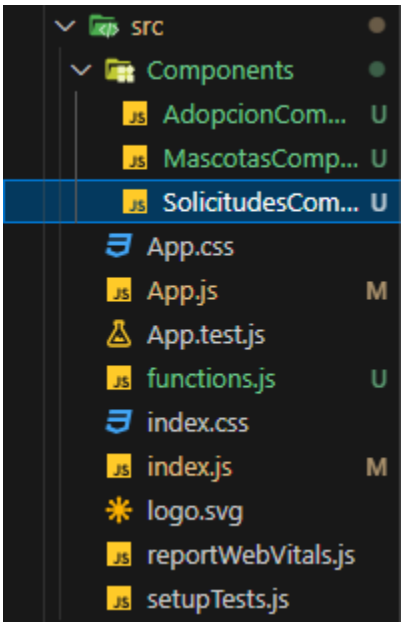
```
const openModal =(opcion, idMascota, nombre, nombreS, idS, estado)=>{
```

Observemos el orden en que las variables fueron cargadas, la variable nombre es la variable designada para nombre de la mascota mientras que la variable nombreS es la variable designada a guardar el nombre del solicitante.

```
const AdopcionComponent = () => {
  const url = "http://localhost:8000/mascotas";
  const [mascotas, setMascotas] = useState([]);
  const [id, setId] = useState("");
  const [nombre, setNombre] = useState("");
  const [especie, setEspecie] = useState("");
  const [edad, setEdad] = useState("");
  const [disponible, setDisponible] = useState("");
  const [operacion, setOperacion] = useState("");
  const [titulo,setTitulo]=useState("");
  //Tabla solicitudes_adopcion
  const urlS = "http://localhost:8000/solicitudes";
  const [solicitudes, setSolicitudes] = useState([]);
  const [idS, setIdS] = useState("");
  const [nombreS, setNombreS] = useState("");
  const [idMascota, setIdMascota] = useState("");
  const [estado, setEstado] = useState("");
```

Ahora creamos un nuevo componente para poder visualizar las solicitudes de adopción realizadas.

Dentro de la carpeta **Components** creamos el archivo **SolicitudesComponent.js**



Ahora vamos a permitirle al usuario visualizar las solicitudes que solamente estén en estado **Pendiente** de lo contrario no debería aparecer, para ello vamos a crear una nueva ruta:

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

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


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

export default App;
```

Ahora en la parte de los import queda igual

```
//IMPORT
import React, { useEffect, useState } from "react";
import axios from "axios";
import { mostrarAlerta } from "../functions.js";
import Swal from 'sweetalert2';
import withReactContent from "sweetalert2-react-content";
```

Posteriormente vamos a definir las variables para poder guardar los datos que vienen desde el Backend

```
//CUERPO COMPONENTE
const SolicitudesComponent = () => {
  const url = "http://localhost:8000/solicitudes";
  const [solicitudes, setSolicitudes] = useState([]);
  const [id, setId] = useState("");
  const [nombre_solicitante, setNombre] = useState("");
  const [id_mascota, setIdMascota] = useState("");
  const [estado, setEstado] = useState("");
  const [operacion, setOperacion] = useState("");
  const [titulo,setTitulo]=useState("");
```

Ahora realizamos las funciones que nos permitirán visualizar las solicitudes de adopción

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

const getSolicitudes = async () => {
  const respuesta = await axios.get(`${url}/buscar`);
  console.log(respuesta.data);
  setSolicitudes(respuesta.data);
};
```

Después realizamos la función que nos va permitir enviar la solicitud y cambiar su estado en la base de datos.

```
const enviarSolicitud = async (metodo, parametros)=>{
  await axios({method: metodo, url: parametros.urlExt, data: parametros })
  .then((respuesta)=>{
      let tipo= respuesta.data.tipo;
      let mensaje = respuesta.data.mensaje;
      mostrarAlerta(mensaje,tipo);
      if(tipo ==="success"){
          document.getElementById("btnCerrarModal").click();
          getSolicitudes();
      }
  })
  .catch((error)=>{
      mostrarAlerta(`Error en la solicitud`,error)
  });
};
```

Ahora creamos la función que nos permitirá cambiar el estado de **Pendiente** por **Aprobado**

```
const estadoAprobar=(id,nombre_solicitante)=>{
  const MySwal = withReactContent(Swal);
  MySwal.fire({
      title: `Esta seguro de aprobar la solicitud de ${nombre_solicitante} ?`,
      icon: 'question',
      text: 'Tenga en cuenta de haber hecho el estudio anteriormente para que la mascota quede en buenas manos',
      showCancelButton: true,
      confirmButtonText: 'Si, aprobar',
      cancelButtonText: 'Cancelar'
  }).then((result)=>{
      if(result.isConfirmed){
          enviarSolicitud("PUT",{urlExt: `${url}/actualizar/${id}`,estado:'Aprobado'})
      }
      else{
          mostrarAlerta("No se pudo aprobar la solicitud","info");
      }

  })
}
```

Ahora realizamos lo siguiente

```
<table className="table table-bordered">
  <thead>
    <tr>
      <th>#</th>
      <th>NOMBRE</th>
      <th>MASCOTA</th>
      <th>ESTADO</th>
    </tr>
  </thead>
  <tbody className="table-group-divider">
    {solicitudes.map((solicitud, i) => (
      <tr key={solicitud.id}>
        <td>{solicitud.id}</td>
        <td>{solicitud.nombre_solicitante}</td>
        <td>{solicitud.id_mascota}</td>
        <td>{solicitud.estado}</td>
        <td>
          <button
            onClick={()=>estadoAprobar(solicitud.id, solicitud.nombre_solicitante)}
            className="btn btn-success"
            disabled={solicitud.estado === 'Aprobado'}
          >
            <i className="fa-solid fa-check"></i>
          </button>
        </td>
      </tr>
    ))}
  </tbody>
```

La sección en rojo es la definición de los títulos de la tabla, la sección amarilla es todo el cuerpo de la tabla en donde vamos a mostrar los datos de cada solicitud de adopción, lo blanco es donde llamamos a la función para actualizar el estado de cada registro; y la flecha de color verde nos indica la condición que permite deshabilitar el botón para aprobar las solicitudes, siempre y cuando el estado sea **Aprobado**.

Observación: Debido a que la aplicación trabaja de modo asíncrono es necesario recargar el navegador para poder observar los cambios que se han efectuando.