

Universidad de Nariño

Ingeniería de Sistemas

Diplomado de actualización en nuevas tecnologías para el desarrollo de  
Software

Taller Final (Backend&Frontend)

Jhonattan Gabriel Benavides Concha

Código: 219034046

5 de enero de 2024

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

Como continuación del TallerUnidad3Frontend, se añadirá primero un campo buscador de filtrado por nombre o raza, que se actualiza dinámicamente según el usuario escribe en el campo de búsqueda.

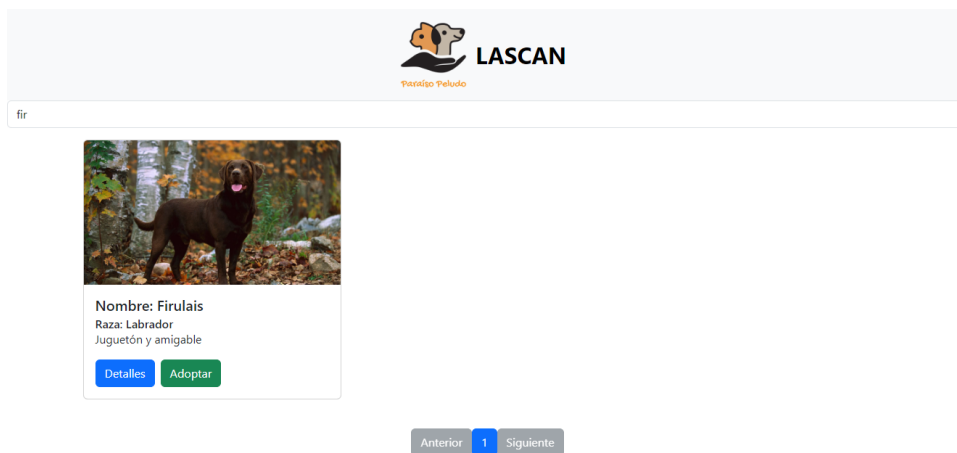
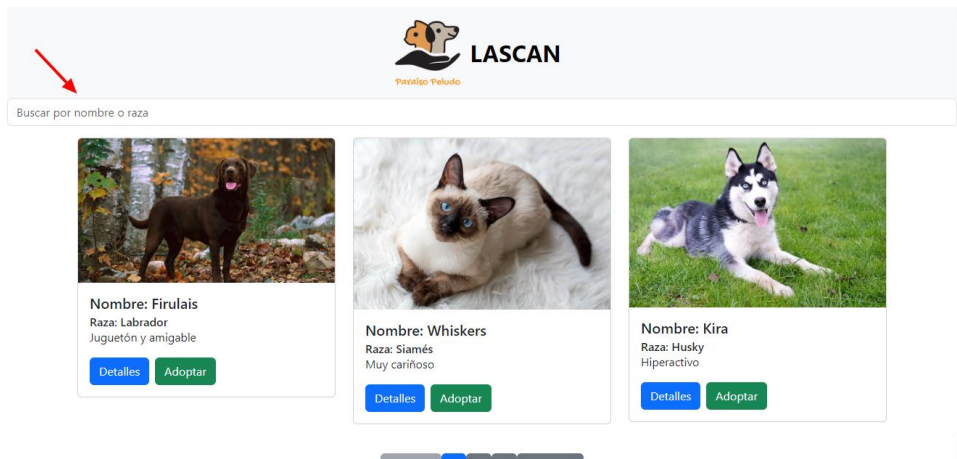
```
33      /*
34       searchTerm es el estado que almacena el término de búsqueda actual.
35       setSearchTerm es la función que se utiliza para actualizar el valor de searchTerm.
36      */
37      const [searchTerm, setSearchTerm] = useState("");
38  
```

```
39      /*
40       useCallback es un gancho de React que memoriza una función para evitar que se vuelva a crear en cada renderizado, a
41       menos que sus dependencias cambien.
42       La función getMascotas realiza una solicitud a la API para obtener la lista de mascotas.
43       El array [searchTerm] en el segundo argumento de useCallback especifica que la función depende del valor de
44       searchTerm. Si searchTerm cambia, se creará una nueva versión de la función.
45      */
46      const getMascotas = useCallback(async () => {
47          const respuesta = await axios.get(`${urlMascotas}/buscar`);
48          const mascotasFiltradas = respuesta.data.mascotas.filter(
49              (mascota) =>
50                  mascota.nombre.toLowerCase().includes(searchTerm.toLowerCase()) ||
51                  mascota.raza.toLowerCase().includes(searchTerm.toLowerCase())
52          );
53          setMascotas(mascotasFiltradas);
54      }, [searchTerm]);
  
```

```
56      /*
57       Efecto de carga inicial para obtener las mascotas al renderizar el componente
58       Este efecto se dispara cada vez que getMascotas cambia. En este caso, solo cambia si searchTerm cambia.
59      */
60      useEffect(() => {
61          getMascotas();
62      }, [getMascotas]);
  
```

```
99      /*
100     handleSearch es una función que se ejecuta cuando el valor del campo de búsqueda cambia.
101     Actualiza el estado searchTerm con el nuevo valor del campo de búsqueda.
102     */
103     const handleSearch = (e) => {
104         setSearchTerm(e.target.value);
105     };
  
```

```
148     { /* Campo de búsqueda */ }
149     <Form.Control
150         type="text"
151         placeholder="Buscar por nombre o raza"
152         value={searchTerm}
153         onChange={handleSearch}
154     />
155  
```



Finalmente se añadirá un login para que el administrador se autentique, con su usuario y contraseña. Además, este administrador tendrá las funcionalidades de añadir, editar y eliminar mascotas.

Antes de seguir con el proceso de construcción de lo expuesto anteriormente se debe crear el modelo Administradores en el backend.

```
mascotasModel.js  administradoresModel.js U X
1  import Sequelize from "sequelize";
2  import { db } from "../database/conexion.js";
3
4  const Administradores = db.define('Administradores', {
5      usuario: {
6          type: Sequelize.STRING,
7          allowNull: false
8      },
9      contraseña: {
10         type: Sequelize.STRING,
11         allowNull: false
12     }
13 }, {
14     tableName: 'Administradores',
15     timestamps: false
16 });
17
18 export { Administradores };
19
```

Se necesitará el controlador para verificar si existe el administrador al momento de ingresar.

```
mascotasModel.js  administradoresModel.js U  mascotasController.js  administradoresController.js U X
1  import { Administradores } from '../modelos/administradoresModel.js';
2
3  const obtenerTodosLosAdministradores = (req, res) => {
4      Administradores.findAll()
5          .then(resultados => {
6              if (resultados.length === 0) {
7                  res.status(404).json({
8                      mensaje: "No se encontraron administradores."
9                  });
10             } else {
11                 res.status(200).json({
12                     mensaje: "Búsqueda exitosa",
13                     administradores: resultados
14                 });
15             }
16         })
17         .catch(err => {
18             res.status(500).json({
19                 mensaje: `Error al realizar la búsqueda: ${err}`
20             });
21         });
22     };
23
24     export {
25         obtenerTodosLosAdministradores,
26     };
27
```

Del mismo modo la ruta para obtener los administradores existentes.

```
administradoresModel.js U  mascotasController.js  administradoresController.js U  mascotasRouter.js  administradoresRouter.js U X
1  import express from 'express';
2  import { obtenerTodosLosAdministradores, } from '../controladores/administradoresController.js';
3  const routerAdministradores = express.Router();
4
5  routerAdministradores.get("/buscar", (req, res) => {
6      obtenerTodosLosAdministradores(req, res);
7  });
8
9  export { routerAdministradores };
10
```

```

1 import express from "express";
2 import bodyParser from "body-parser";
3 import cors from "cors";
4
5 import { routerMascotas } from "../rutas/mascotasRouter.js";
6 import { routerSolicitudesAdopcion } from "../rutas/solicitudesAdopcionRouter.js";
7 import { routerAdministradores } from "../rutas/administradoresRouter.js";
8 import {db} from "../database/conexion.js";
9
25 app.use("/mascotas", routerMascotas);
26 app.use("/solicitudesAdopcion", routerSolicitudesAdopcion);
27 app.use("/administradores", routerAdministradores);

```

Se comprueba por thunder client si se obtienen los administradores correctamente

```

GET http://localhost:8000/administradores/buscar
Send
Status: 200 OK Size: 108 Bytes Time: 395 ms
Response
1 {
2   "mensaje": "Búsqueda exitosa",
3   "administradores": [
4     {
5       "id": 1,
6       "usuario": "jbenavides",
7       "contraseña": "12345678"
8     }
9   ]
10  }

```

Ahora bien, ya realizado se procede con la parte del frontend. Primeramente, se añade el model login para el inicio de sesión del administrador que dará paso a las funcionalidades permitidas para este (añadir, editar y eliminar mascota).

Para el icono de login se instaló la siguiente dependencia.

```

$ npm install --save @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons @fortawesome/react-fontawesome
added 4 packages, and audited 1585 packages in 52s
257 packages are looking for funding
  run `npm fund` for details
9 vulnerabilities (1 low, 2 moderate, 6 high)
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.

6 import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
7 import { faUser } from '@fortawesome/free-solid-svg-icons';

```

Al ser necesario verificar si existe el administrador se tuvo que traer los administradores existentes.

```

14 // Definición de la URL de la API para obtener información de administradores
15 const urlAdministradores = "http://localhost:8000/administradores";
16
20 // Estado para almacenar la lista de administradores obtenidos de la API
21 const [administradores, setAdministradores] = useState([]);
22

```

```

47      /*
48       Estado para controlar la visibilidad del modal de inicio de sesión de administrador.
49      */
50      const [showAdminLoginModal, setShowAdminLoginModal] = useState(false);
51
52      /*
53       Estado para almacenar las credenciales de administrador (nombre de usuario y contraseña).
54      */
55      const [adminCredentials, setAdminCredentials] = useState({
56        username: "",
57        password: "",
58      });
59
60      /*
61       Estado para comprobar si el administrador ha iniciado sesión o no.
62      */
63      const [isAdminLoggedIn, setIsAdminLoggedIn] = useState(false);
64

```

```

126      /*
127      Función para obtener la lista de administradores desde el servidor.
128      La respuesta se almacena en la variable 'respuesta'.
129      Actualiza el estado 'administradores' con la lista de administradores obtenida del servidor.
130      */
131      const getAdministradores = async () => {
132        const respuesta = await axios.get(`${urlAdministradores}/buscar`);
133        setAdministradores(respuesta.data.administradores);
134      };

```

```

144      /*
145       Efecto de carga para obtener los administradores al renderizar el componente
146       Este efecto se dispara una vez debido al array de dependencias vacío ([]).
147      */
148      useEffect(()=>{
149        getAdministradores();
150      }, []);

```

```

195      /*
196       Función para mostrar el modal de inicio de sesión de administrador.
197      */
198      const handleAdminLogin = () => {
199        setShowAdminLoginModal(true);
200      };
201
202      /*
203       Función para cerrar el modal de inicio de sesión de administrador.
204      */
205      const closeAdminLoginModal = () => {
206        setShowAdminLoginModal(false);
207      };
208
209      /*
210       Función para manejar los cambios en los campos de entrada del inicio de sesión de administrador.
211       Actualiza el estado de las credenciales de administrador.
212      */
213      const handleAdminLoginInputChange = (e) => {
214        const { name, value } = e.target;
215        setAdminCredentials({ ...adminCredentials, [name]: value });
216      };
217

```

```

219     Función que verifica las credenciales y muestra mensajes de éxito o error utilizando la librería Swal.
220     Actualiza el estado de inicio de sesión de administrador (esto para poder visualizar o no las funcionalidades
221     de añadir, editar y eliminar mascotas).
222     Cierra el modal de inicio de sesión independientemente del resultado.
223     */
224     const handleAdminLoginSubmit = async (e) => {
225         e.preventDefault();
226         const administradorValido = administradores.find(
227             (admin) => admin.username === adminCredentials.username && admin.password === adminCredentials.password
228         );
229
230         if (administradorValido) {
231             Swal.fire({
232                 icon: 'success',
233                 title: 'Inicio de sesión exitoso!',
234                 showCancelButton: false,
235                 showConfirmButton: true,
236             });
237
238             setIsAdminLoggedIn(true);
239         } else {
240             Swal.fire({
241                 icon: 'error',
242                 title: 'Inicio de sesión fallido!',
243                 showCancelButton: false,
244                 showConfirmButton: true,
245             });
246
247             setIsAdminLoggedIn(false);
248         }
249
250         closeAdminLoginModal();
251     };

```

```

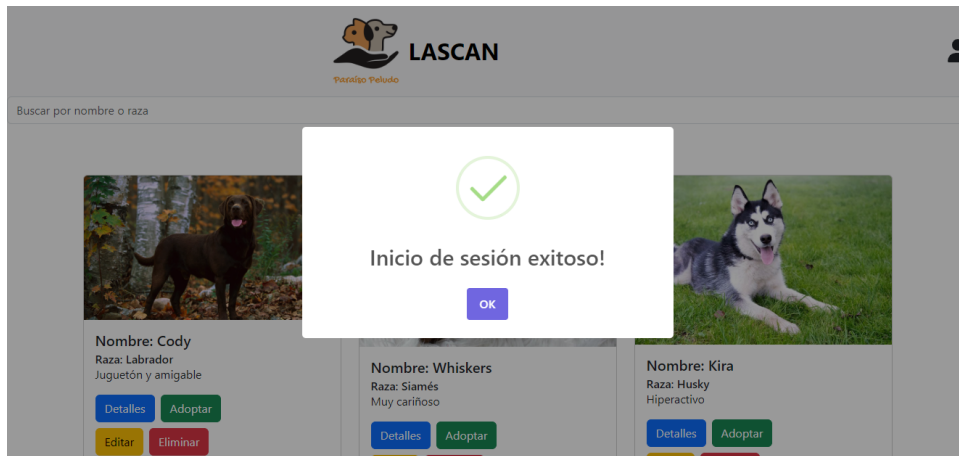
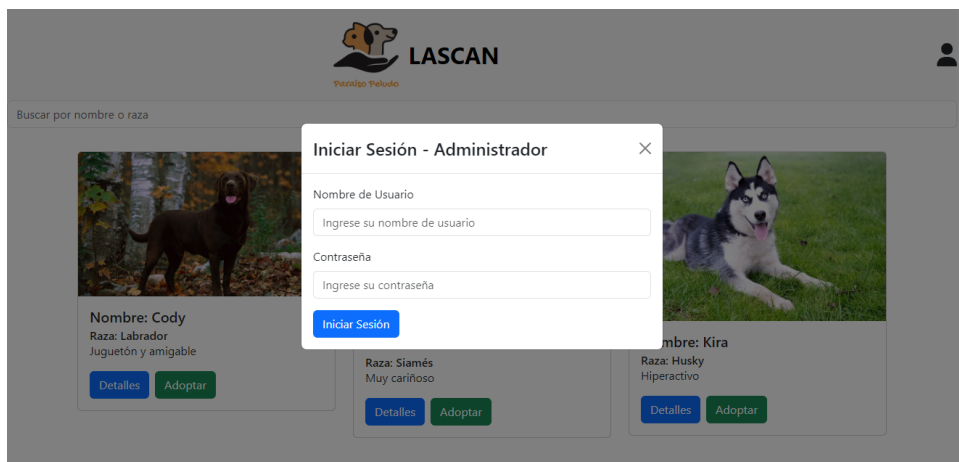
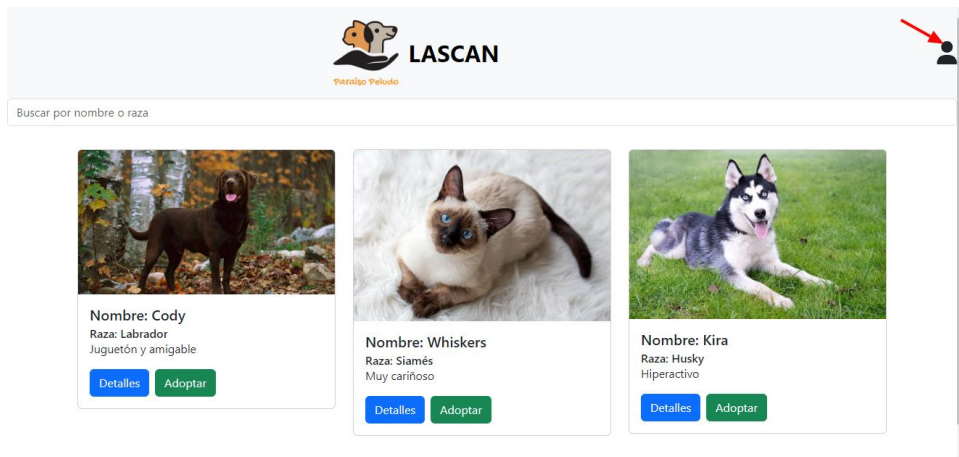
384     <Navbar.Collapse className="justify-content-end">
385         <FontAwesomeIcon icon={faUser} size="2x" onClick={() => handleAdminLogin()} style={{ cursor: 'pointer' }}/>
386     </Navbar.Collapse>

```

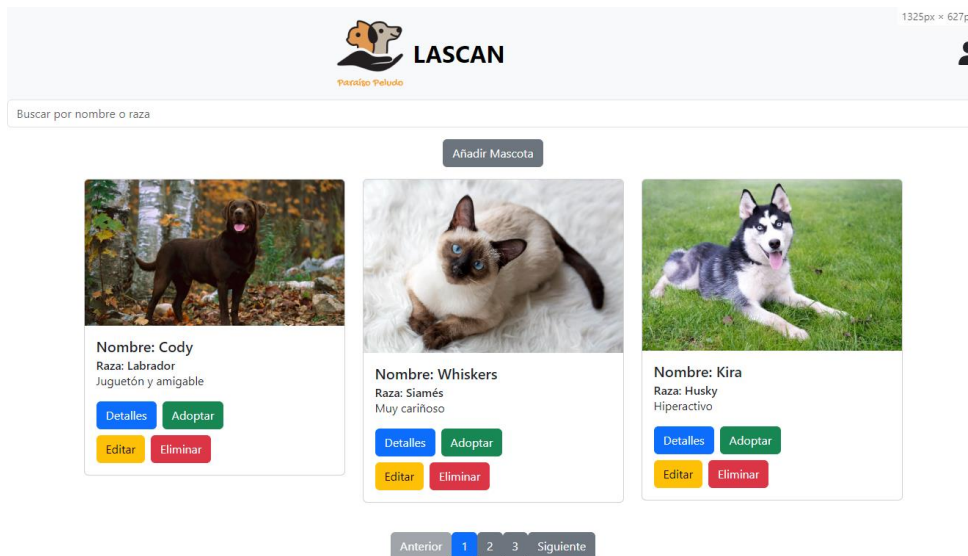
```

529     {/* Login Administrador Modal */}
530     <Modal show={showAdminLoginModal} onHide={closeAdminLoginModal} centered>
531         <Modal.Header closeButton>
532             <Modal.Title>Iniciar Sesión - Administrador</Modal.Title>
533         </Modal.Header>
534         <Modal.Body>
535             <Form>
536                 <Form.Group className="mb-3" controlId="formUsername">
537                     <Form.Label>Nombre de Usuario</Form.Label>
538                     <Form.Control
539                         type="text"
540                         placeholder="Ingresa su nombre de usuario"
541                         name="username"
542                         value={adminCredentials.username}
543                         onChange={handleAdminLoginInputChange}
544                     />
545                 </Form.Group>
546                 <Form.Group className="mb-3" controlId="formPassword">
547                     <Form.Label>Contraseña</Form.Label>
548                     <Form.Control
549                         type="password"
550                         placeholder="Ingresa su contraseña"
551                         name="password"
552                         value={adminCredentials.password}
553                         onChange={handleAdminLoginInputChange}
554                     />
555                 </Form.Group>
556                 <Button variant="primary" type="submit" onClick={handleAdminLoginSubmit}>
557                     Iniciar Sesión
558                 </Button>
559             </Form>
560         </Modal.Body>
561     </Modal>

```







A continuación, se presenta el código correspondiente a las funcionalidades del administrador (añadir, editar y eliminar mascotas)

```

65  /*
66  Estado para controlar la visibilidad del modal de edición de mascotas.
67  */
68  const [showEditModal, setShowEditModal] = useState(false);
69
70  /*
71  Estado para almacenar los datos del formulario de mascotas, que se utilizan al agregar o editar una mascota.
72  */
73  const [formData, setFormData] = useState({
74    nombre: "",
75    tipo: "",
76    raza: "",
77    edad: "",
78    descripcion: "",
79    detalle: "",
80    foto: ""
81  });
82
83  /*
84  Estado para determinar si el formulario está en modo de edición o no.
85  */
86  const [editMode, setEditMode] = useState(false);
87

```

```

89  /*
90  Función que se llama al abrir el modal de edición de mascotas.
91  Recibe la información de una mascota y actualiza el estado del formulario con esos datos.
92  Establece el modo de edición en verdadero y muestra el modal de edición.
93  */
94  const openEditModal = (mascota) => {
95    setEditMode(true);
96    setFormData({
97      id: mascota.id,
98      nombre: mascota.nombre,
99      tipo: mascota.tipo,
100     raza: mascota.raza,
101     edad: mascota.edad,
102     descripcion: mascota.descripcion,
103     detalle: mascota.detalle,
104     foto: mascota.foto,
105   });
106   setShowEditModal(true);
107 };

```

```

291  /*
292     Función para manejar los cambios en los campos de entrada del formulario de mascotas.
293     Convierte el valor de 'edad' a un número entero si el campo es 'edad'.
294     Actualiza el estado del formulario de mascotas.
295  */
296  const handleFormInputChange = (e) => {
297    const { name, value } = e.target;
298    const valorParseado = name === 'edad' ? parseInt(value, 10) : value;
299    setFormData({ ...formData, [name]: valorParseado });
300  };
301
302  /*
303     Función para cerrar el modal de edición de mascotas.
304  */
305  const closeEditModal = () => {
306    setShowEditModal(false);
307  };

```

```

309  /*
310     Función que actualiza la lista de mascotas después de una adición exitosa.
311  */
312  const submitAddForm = async (e) => {
313    e.preventDefault();
314
315    try {
316      await axios.post(`${urlMascotas}/crear`, formData);
317
318      getMascotas();
319
320      Swal.fire({
321        icon: 'success',
322        title: 'Mascota añadida exitosamente!'
323      });
324    } catch (error) {
325      Swal.fire({
326        icon: 'error',
327        title: 'Error al añadir la mascota!',
328        text: 'Por favor, inténtalo de nuevo.'
329      });
330    }
331    closeEditModal();
332  };

```

```

334  /*
335     Función que actualiza la lista de mascotas después de una edición exitosa.
336  */
337  const submitEditForm = async (e) => {
338    e.preventDefault();
339
340    try {
341      await axios.put(`${urlMascotas}/actualizar/${formData.id}`, formData);
342
343      getMascotas();
344
345      Swal.fire({
346        icon: 'success',
347        title: 'Mascota editada exitosamente!'
348      });
349    } catch (error) {
350      Swal.fire({
351        icon: 'error',
352        title: 'Error al editar la mascota!',
353        text: 'Por favor, inténtalo de nuevo.'
354      });
355    }
356    closeEditModal();
357  };

```

```

267 // Función para manejar la eliminación de una mascota
268 const handleDeleteMascota = async (mascotaId) => {
269   try {
270     console.log("Eliminando mascota:", mascotaId);
271
272     await axios.delete(`${urlMascotas}/eliminar/${mascotaId}`);
273
274     getMascotas();
275
276     Swal.fire({
277       icon: 'success',
278       title: 'Mascota eliminada exitosamente!'
279     });
280   } catch (error) {
281     console.error("Error al eliminar la mascota:", error);
282
283     Swal.fire({
284       icon: 'error',
285       title: 'Error al eliminar la mascota!',
286       text: 'Por favor, inténtalo de nuevo.'
287     });
288   }
289 };

```

```

393 /* Botón para añadir mascota visible solo para administradores */
394 <div className="d-flex justify-content-center mt-3">
395   {isAdminLoggedIn && (
396     <Button variant="secondary" onClick={() => { setEditMode(false); setFormData({}); setShowEditModal(true); }}>
397       Añadir Mascota
398     </Button>
399   )}
400 </div>

```

```

424 {isAdminLoggedIn && (
425   <div>
426     <Button variant="warning" className="mt-2" onClick={() => {openEditModal(mascota);}}>
427       Editar
428     </Button>
429     <Button variant="danger" className="mt-2 ms-2" onClick={() => handleDeleteMascota(mascota.id)}>
430       Eliminar
431     </Button>
432   </div>
433 )}

```

```

559 /* Adición y Edición de Mascotas Modal */
560 <Modal show={showEditModal} onHide={closeEditModal} centered>
561   <Modal.Header closeButton>
562     <Modal.Title>{editMode ? "Editar Mascota" : "Añadir Mascota"}</Modal.Title>
563   </Modal.Header>
564   <Modal.Body>
565     <Form onSubmit={editMode ? submitEditForm : submitAddForm}>
566       <Form.Group className="mb-3" controlId="formNombre">
567         <Form.Label>Nombre</Form.Label>
568         <Form.Control
569           type="text"
570           placeholder="Ingresa el nombre"
571           name="nombre"
572           value={formData.nombre || ''}
573           onChange={handleFormInputChange}
574         />
575       </Form.Group>
576       <Form.Group className="mb-3" controlId="formTipo">
577         <Form.Label>Tipo</Form.Label>
578         <Form.Control
579           as="select"
580           name="tipo"
581           value={formData.tipo}
582           onChange={handleFormInputChange}
583         >
584           <option value="">---</option>
585           <option value="Perro">Perro</option>
586           <option value="Gato">Gato</option>
587         </Form.Control>
588       </Form.Group>

```

```

589     <Form.Group className="mb-3" controlId="formRaza">
590       <Form.Label>Raza</Form.Label>
591       <Form.Control
592         type="text"
593         placeholder="Ingrese la raza"
594         name="raza"
595         value={formData.raza || ''}
596         onChange={handleFormInputChange}
597       />
598     </Form.Group>
599     <Form.Group className="mb-3" controlId="formEdad">
600       <Form.Label>Edad</Form.Label>
601       <Form.Control
602         type="text"
603         placeholder="Ingrese la edad"
604         name="edad"
605         value={formData.edad || ''}
606         onChange={handleFormInputChange}
607       />
608     </Form.Group>
609     <Form.Group className="mb-3" controlId="formDescripcion">
610       <Form.Label>Descripción</Form.Label>
611       <Form.Control
612         type="textarea"
613         placeholder="Ingrese la descripción corta"
614         name="descripcion"
615         value={formData.descripcion || ''}
616         onChange={handleFormInputChange}
617       />
618     </Form.Group>

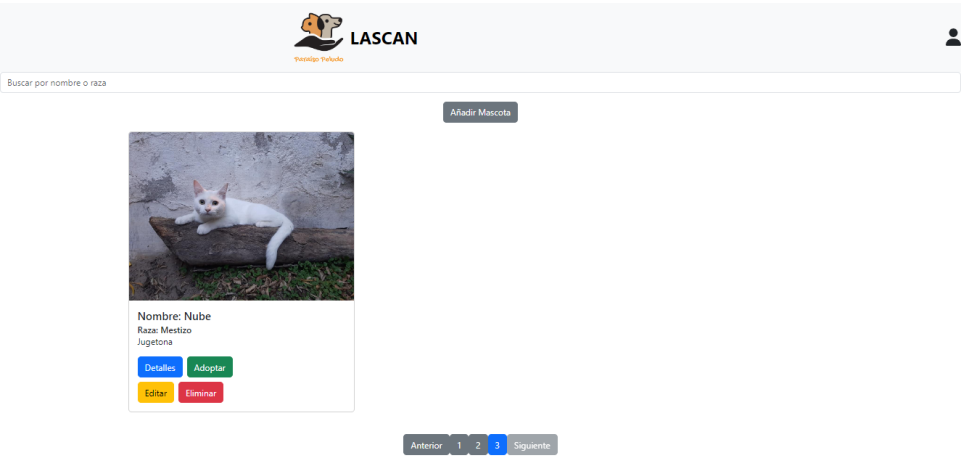
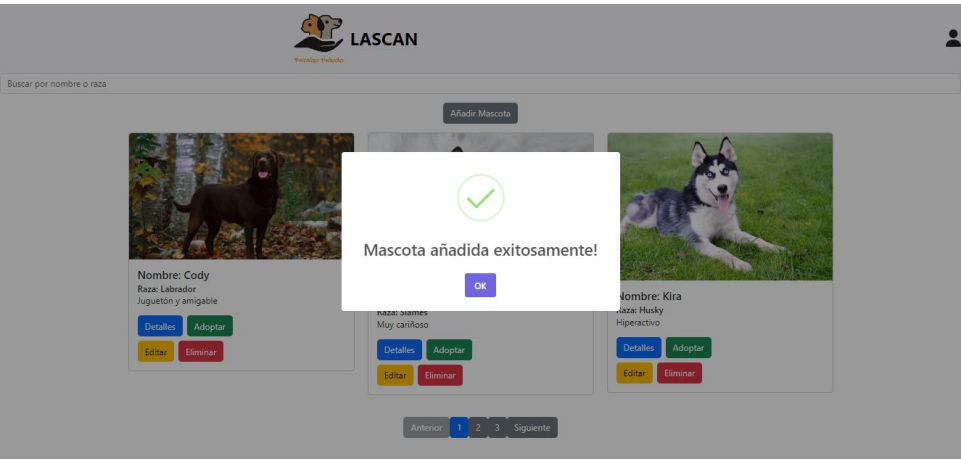
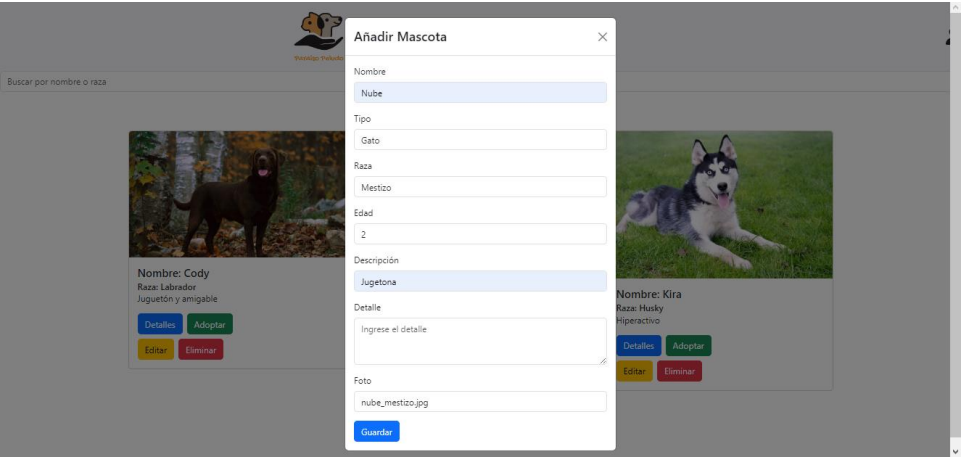
```

```

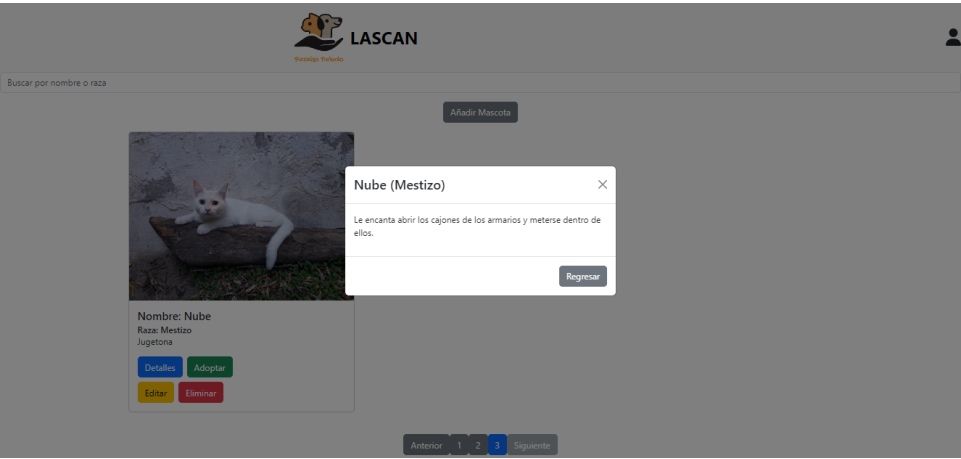
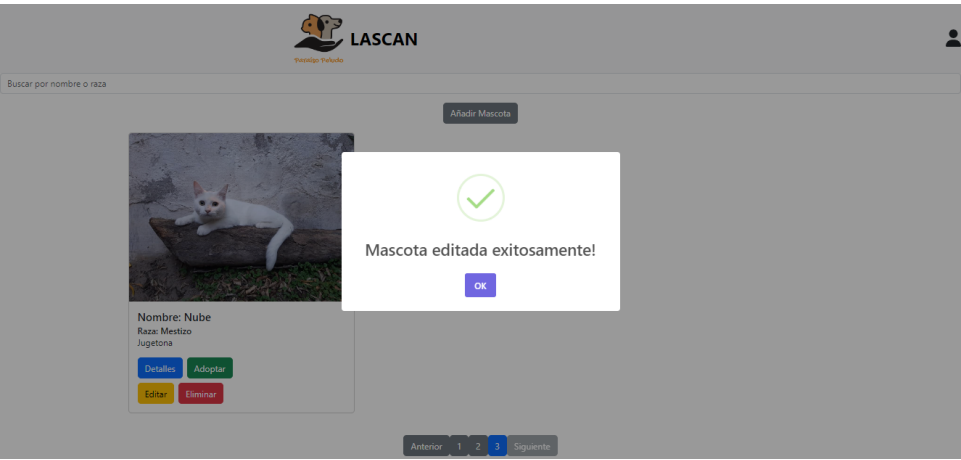
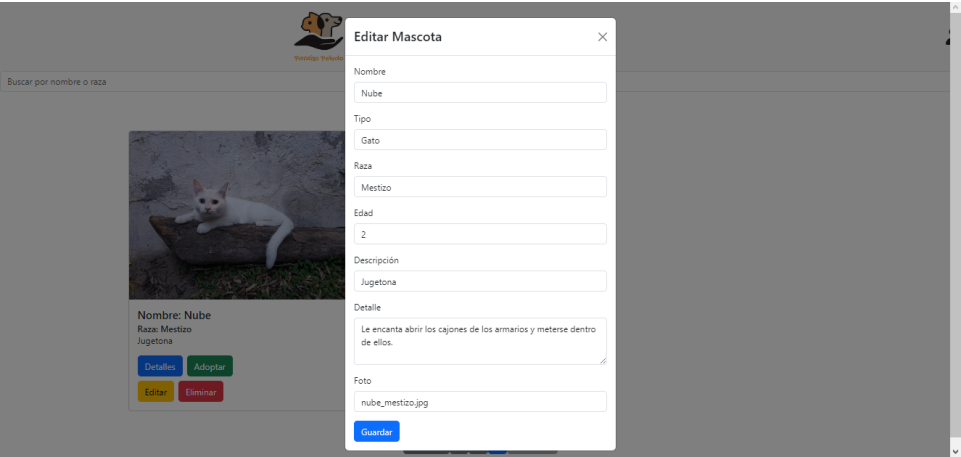
619     <Form.Group className="mb-3" controlId="formDetalle">
620       <Form.Label>Detalle</Form.Label>
621       <Form.Control
622         as="textarea"
623         rows={3}
624         placeholder="Ingrese el detalle"
625         name="detalle"
626         value={formData.detalle || ''}
627         onChange={handleFormInputChange}
628       />
629     </Form.Group>
630     <Form.Group className="mb-3" controlId="formFoto">
631       <Form.Label>Foto</Form.Label>
632       <Form.Control
633         type="text"
634         placeholder="Ingrese la URL de la foto"
635         name="foto"
636         value={formData.foto || ''}
637         onChange={handleFormInputChange}
638       />
639     </Form.Group>
640     <Button variant="primary" type="submit">
641       Guardar
642     </Button>
643   </Form>
644 </Modal.Body>
645 </Modal>

```


Funcionalidad “Añadir Mascota”



Funcionalidad “Editar”



Funcionalidad “Eliminar”




LASCAN

Protegiendo tu futuro

Buscar por nombre o raza

Añadir Mascota




Nombre: Kira  
Raza: Husky  
Hiperactivo

Detalles

Adoptar

Editar

Eliminar




Nombre: Kira  
Raza: Husky  
Hiperactivo

Detalles

Adoptar

Editar

Eliminar



Nombre: Rocky  
Raza: Pitbull  
Agresivo

Detalles

Adoptar

Editar

Eliminar


Anterior

1

2

3

Siguiente




LASCAN

Protegiendo tu futuro

Buscar por nombre o raza

Añadir Mascota



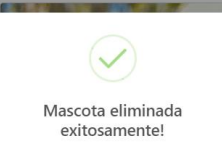
Nombre: Kira  
Raza: Husky  
Hiperactivo

Detalles

Adoptar

Editar

Eliminar




Nombre: Kira  
Raza: Husky  
Hiperactivo

Detalles

Adoptar

Editar

Eliminar




Nombre: Rocky  
Raza: Pitbull  
Agresivo

Detalles

Adoptar

Editar

Eliminar



Nombre: Nube  
Raza: Mestizo  
Juguetona

Detalles

Adoptar

Editar

Eliminar

Anterior

1


2

Siguiente

✓

Mascota eliminada exitosamente!

OK




LASCAN

Protegiendo tu futuro

Buscar por nombre o raza

Añadir Mascota




Nombre: Kira  
Raza: Husky  
Hiperactivo

Detalles

Adoptar

Editar

Eliminar




Nombre: Rocky  
Raza: Pitbull  
Agresivo

Detalles

Adoptar

Editar

Eliminar



Nombre: Nube  
Raza: Mestizo  
Juguetona

Detalles

Adoptar

Editar

Eliminar

Anterior

1

2

Siguiente