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

Taller Frontend

Autor:

Jeason Steven Gelpud Cadena

Docente:

Vicente Aux Revelo

Universidad de Nariño

Facultad de Ingeniería

Ingeniería de Sistemas

Ipiales, Colombia

A continuación, se realiza el desarrollo del Taller frontend

1. Creación de Componentes en Angular

Para crear un componente en Angular, se utiliza el siguiente comando en la terminal:

```
ng generate component nombre-del-componente
```

cada componente tiene su archivo de conponent.ts, conponent.html conponent.css y el de pruebas unitarias conponent.spec.ts

nombre-del-componente.component.ts:

- Este archivo contiene la lógica del componente. Aquí se define la clase del componente, sus propiedades y métodos, y también se pueden incluir decoradores como @Component que configuran cómo se comporta el componente.

nombre-del-componente.component.html:

- Este archivo contiene la plantilla HTML del componente, donde se define la estructura y el contenido que se mostrará en la interfaz de usuario. Es el lugar donde se puede usar la sintaxis de Angular para enlazar datos y manejar eventos.

nombre-del-componente.component.css

- Este archivo contiene los estilos CSS específicos para ese componente. Aquí se puede aplicar estilos que afecten solo a este componente, lo que ayuda a mantener el código más organizado y modular.

nombre-del-componente.component.spec.ts:

- Este archivo se utiliza para las pruebas unitarias del componente. Donde Angular crea una configuración básica que se puede utilizar para verificar que el componente funciona correctamente.

Ya creado los módulos podemos empezar a implementarlos de esta manera empezamos a utilizar los parámetros que dicta el taller que deben ser utilizados.

1.1 Uso de ngModel

`ngModel` es una directiva que permite la vinculación de datos entre el componente y la vista. Se utilizo principalmente en los formularios para mantener el estado de los inputs.

Ejemplo de uso de ngModel:

En cada uno de los modelos se utiliza ngModel, esto depende según el componte ya que se lo utiliza de diferentes maneras, en este caso se usa [(ngModel)]="searchQuery" para

enlazar el valor del input con la variable searchQuery en el componente ***lista-usuarios.compnen.html***

```
<form class="form-inline justify-content-between mb-3
    <fieldset class="form-group flex-grow-1">
        <input
            type="text"
            class="form-control"
            placeholder="Buscar usuario..."
            aria-label="Buscar"
            [(ngModel)]="searchQuery"
            name="search"
            (input)="search()">
    </fieldset>
    <fieldset class="form-group">
```

Otro ejemplo del implementación de ngModel lo podemos encontra en el componente ***editar-usuarios.componet.html***, utiliza la directiva ngModel para crear o editar un usuario, haciendo un **data binding** bidireccional, conectando el modelo de datos (usuario) con los inputs del formulario, lo que asegura que cualquier cambio en el input se refleje inmediatamente en el modelo y viceversa.

En este caso, los campos de nombre, email, contraseña, teléfono, dirección, fecha de registro y rol están vinculados a las propiedades del objeto usuario.

```
<div class="container">
    <h2 *ngIf="idUsuario; else nuevoUsuario">Editar Usuario<
    <ng-template #nuevoUsuario><h2>Nuevo Usuario</h2></ng-te

    <form (ngSubmit)="onSubmit()">
        <div class="form-group">
            <label for="nombre">Nombre</label>
            <input
                type="text"
                id="nombre"
                class="form-control"
                [(ngModel)]="usuario.nombre"
                name="nombre"
                required />
        </div>

        <div class="form-group">
            <label for="email">Email</label>
            <input
                type="email"
                id="email"
                class="form-control"
                [(ngModel)]="usuario.email"
                name="email"
                required />
        </div>

        <div class="form-group">
            <label for="password">Contraseña</label>
            <input
                type="password"
                id="password"
                class="form-control"
                [(ngModel)]="usuario.password"
```

ngModel fue clave por varias razones, ya que proporciona una forma eficiente y sencilla de enlazar los datos entre el modelo (datos de la aplicación) y la vista (interfaz de usuario). Como lo es el caso de este proyecto, que involucra la gestión de usuarios, adopciones, mascotas y solicitudes, ngModel.

1.3 RouterLink

`RouterLink` es una directiva que se utiliza para navegar entre diferentes rutas dentro de la aplicación Angular. Proporciona una forma declarativa de establecer enlaces.

Ejemplo de uso de RouterLink:

```
<fieldset class="form-group">
    <a type="button" class="btn btn-primary" [routerLink]="['/usuarios/agregar']"
        <i class="fas fa-plus-circle"></i> Nuevo Usuario
    </a>
    <a type="button" class="btn btn-danger" [routerLink]="['/inicio']">
        <i class="fas fa-plus-circle"></i> Salir
    </a>
</fieldset>
```

En este caso, se emplean botones para redirigir al usuario a la página de agregar un nuevo usuario y a la página de inicio. La directiva RouterLink se utiliza para definir la ruta correspondiente a cada acción, permitiendo una navegación fluida dentro de la aplicación.

Para que `routerLink` funcione correctamente, se configuro el sistema de enrutamiento de la siguiente manera en `app-routing.module.ts`.

```
{ path: 'inicio', component: PrincipalMascotasComponent },
{ path: 'mascotas', component: ListaMascotasComponent },
{ path: 'solicitudesAd', component: ListaSolicitudesComponentAd },
{ path: 'solicitudes', component: ListaSolicitudesComponent },
{ path: 'solicitudes/editar/:idSolicitud', component: EditarSolicitudesCompor
{ path: 'solicitudes/agregar', component: EditarSolicitudesComponent},
{ path: 'mascotas/editar/:idMascota', component: EditarMascotasComponent },
{ path: 'mascotas/agregar', component: EditarMascotasComponent },
{ path: 'usuarios', component: ListaUsuariosComponent },
{ path: 'usuarios/editar/:idUsuario', component: EditarUsuariosComponent },
{ path: 'usuarios/agregar', component: EditarUsuariosComponent },
{ path: 'adopciones', component: ListaAdopcionesComponent }, // Nuevo
{ path: 'adopciones/editar/:idAdopcion', component: EditarAdopcionesComponent
{ path: 'adopciones/agregar', component: EditarAdopcionesComponent }, // Nuev
```

1.4 Servicios

En el proyecto el uso de estos servicios fueron fundamental ya que es así como se consume el **backend** creado en el anterior taller.

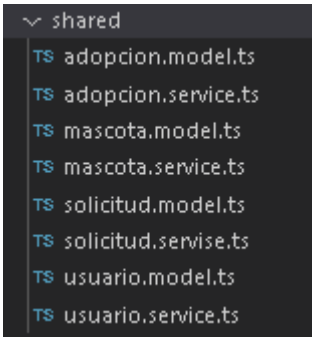
Por otra parte, los servicios nos permiten la interacción con la lógica de negocio relacionada con los usuarios, adopciones, mascotas, solicitudes. Permitiendo que al importarlos que cada uno de los componentes realice sus respectivas operaciones:

```
import { UsuarioService } from '../shared/usuario.service';
```

En este proyecto se creó la carpeta llamada ***shared*** en la cual se crean los archivos de cada uno de los servicios, en esta también se crearon los modelos, al igual que los servicios los modelos son utilizados por los componentes para manejar datos.

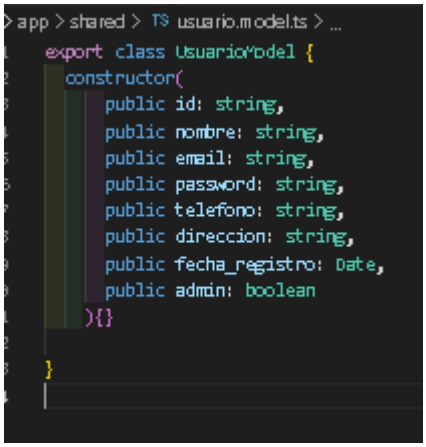
```
import { UsuarioModel } from '../shared/usuario.model';
```

Ejemplo de un servicio:



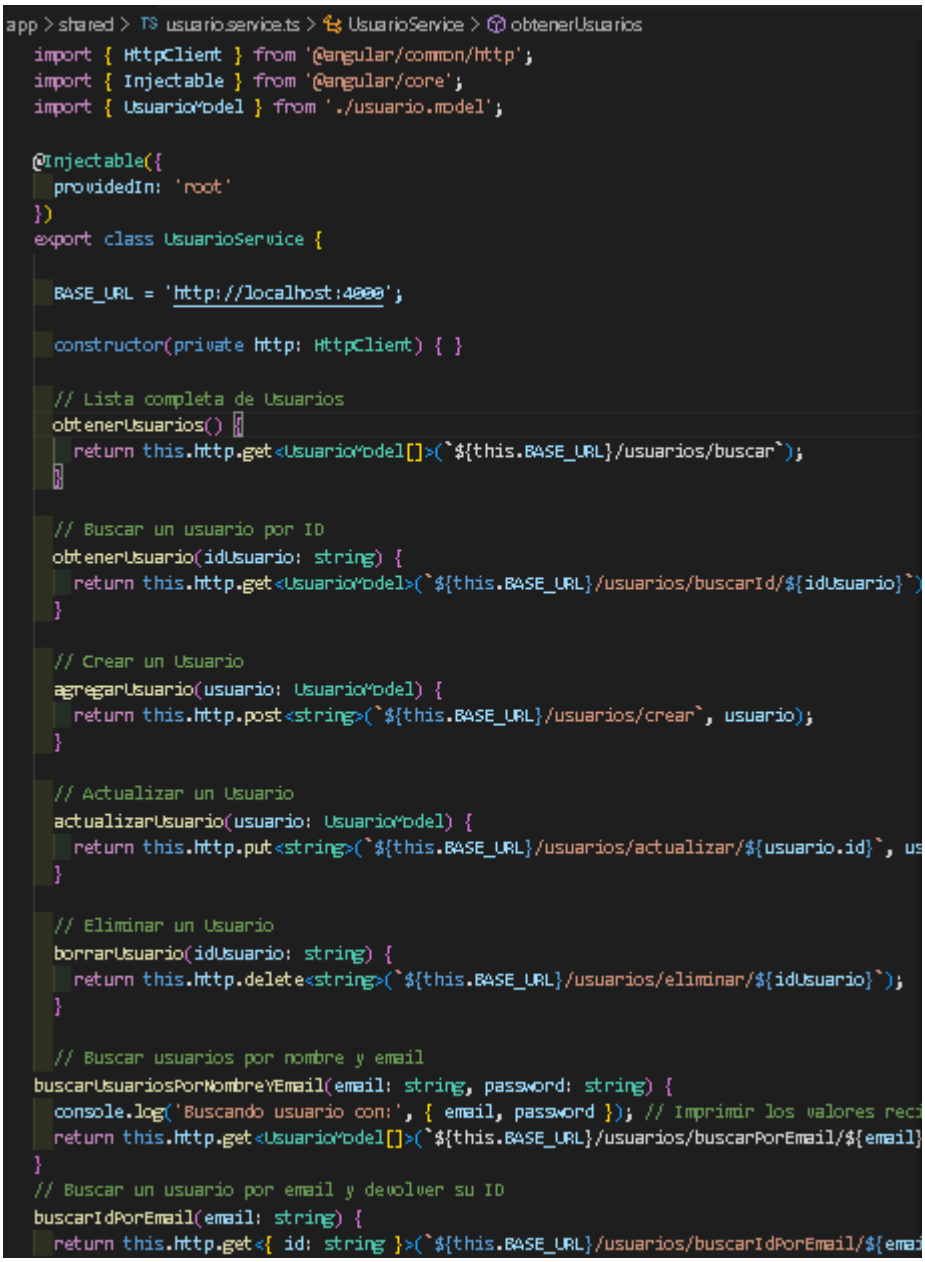
Modelos:

- Estos son una representación estructurada de los datos que una aplicación utiliza, esto incluye los campos o propiedades que los datos tendrán, así como sus tipos y relaciones entre diferentes tipos de datos.



Tomaremos de **ejemplo el servicio** de usuarios:

El UsuarioService esta diseñado para manejar las operaciones relacionadas con los usuarios en la aplicación, utilizando el módulo HttpClient para realizar solicitudes HTTP.



Funciones Principales

- **obtenerUsuarios**: Recupera una lista de todos los usuarios registrados.
- **obtenerUsuario**: Busca un usuario específico por su ID.
- **agregarUsuario**: Crea un nuevo usuario en la base de datos.
- **actualizarUsuario**: Actualiza los datos de un usuario existente.
- **borrarUsuario**: Elimina un usuario por su ID.
- **buscarUsuariosPorNombreYEmail**: Permite buscar usuarios utilizando su email y contraseña.
- **buscarIdPorEmail**: Devuelve el ID de un usuario dado su email.

Características

- Utiliza la URL base de la API (BASE_URL) para realizar las solicitudes.
- Proporciona una estructura clara y modular, lo que facilita la integración y mantenimiento.
- Se sugiere implementar manejo de errores y pruebas unitarias para asegurar su correcto funcionamiento.

2. Uso de HTML5 y JavaScript

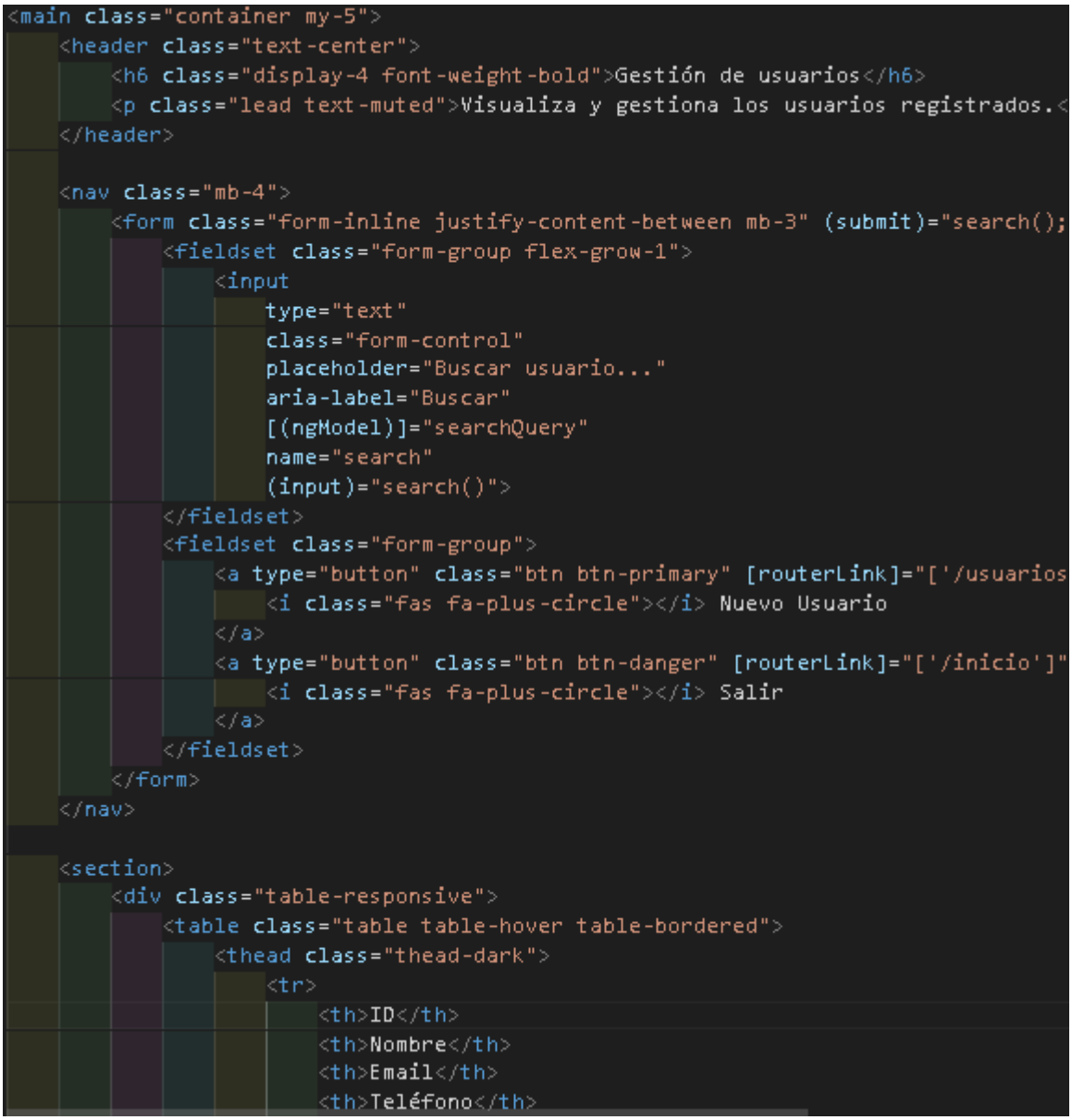
2.1 Estructura Ordenada

Es importante estructurar el HTML de manera que sea semántico y fácil de leer. Utilizar etiquetas HTML5 como `<header>`, `<nav>`, `<main>`, `<footer>` mejora la accesibilidad y SEO.

Ejemplo de estructura HTML5 de lista-usuarios.component.html:

Como lo había mencionado anteriormente al crear los componentes se crean los archivos conponent.ts, conponent.html conponent.css y el de pruebas unitarias conponent.spec.ts

Para manejar la estructura se utiliza el archivo **component.html** ya que este permite construir la interfaz visual de tu aplicación utilizando etiquetas HTML5. Esto incluye la organización del contenido en secciones, encabezados, párrafos, listas, tablas, etc. Por ejemplo:



```
<main class="container my-5">
    <header class="text-center">
        <h6 class="display-4 font-weight-bold">Gestión de usuarios</h6>
        <p class="lead text-muted">Visualiza y gestiona los usuarios registrados.<
    </header>

    <nav class="mb-4">
        <form class="form-inline justify-content-between mb-3" (submit)="search();
            <fieldset class="form-group flex-grow-1">
                <input
                    type="text"
                    class="form-control"
                    placeholder="Buscar usuario..."
                    aria-label="Buscar"
                    [(ngModel)]="searchQuery"
                    name="search"
                    (input)="search()">
            </fieldset>
            <fieldset class="form-group">
                <a type="button" class="btn btn-primary" [routerLink]="['/usuarios
                    <i class="fas fa-plus-circle"></i> Nuevo Usuario
                </a>
                <a type="button" class="btn btn-danger" [routerLink]="['/inicio']"
                    <i class="fas fa-plus-circle"></i> Salir
                </a>
            </fieldset>
        </form>
    </nav>

    <section>
        <div class="table-responsive">
            <table class="table table-hover table-bordered">
                <thead class="thead-dark">
                    <tr>
                        <th>ID</th>
                        <th>Nombre</th>
                        <th>Email</th>
                        <th>Teléfono</th>
```

HTML5

1. **Estructura Semántica**:
    - Se utilizo <main>, <header>, <nav> y <section> para organizar el contenido de manera semántica, lo que mejora la accesibilidad y la SEO.
2. **Formularios**:
    - Se empleo un formulario con un campo de entrada (<input>) para buscar usuarios.
3. **Tablas**:
    - Se implemento una tabla (<table>) para mostrar los datos de los usuarios, utilizando <thead> para los encabezados y <tbody> para los registros. Esto permite una presentación clara y estructurada de la información.

4. **Accesibilidad**:
    - Se utilizaron atributos como aria-label para mejorar la accesibilidad del campo de búsqueda, permitiendo que los lectores de pantalla describan su función.

JavaScript

1. **Interactividad**:
    - Se utilizo Angular para manejar eventos, como la búsqueda de usuarios y la navegación entre diferentes rutas, lo que proporciona una experiencia interactiva y fluida.
2. **Two-Way Data Binding**:
    - Se empleo [(ngModel)] para enlazar el valor del campo de búsqueda (searchQuery) con el modelo de datos en Angular, lo que permite actualizaciones automáticas en la interfaz de usuario al cambiar el valor.
3. **Manejo de Eventos**:
    - Se utilizaron eventos como (submit) y (input) para manejar la lógica de búsqueda de usuarios, mejorando la interactividad y la respuesta de la aplicación.
4. **Navegación Dinámica**:
    - A través de [routerLink], se implemento la navegación entre componentes de Angular, facilitando la gestión de usuarios y permitiendo acciones como agregar, editar o eliminar usuarios de manera intuitiva.

3. Estilos CSS (Uso de Bootstrap)

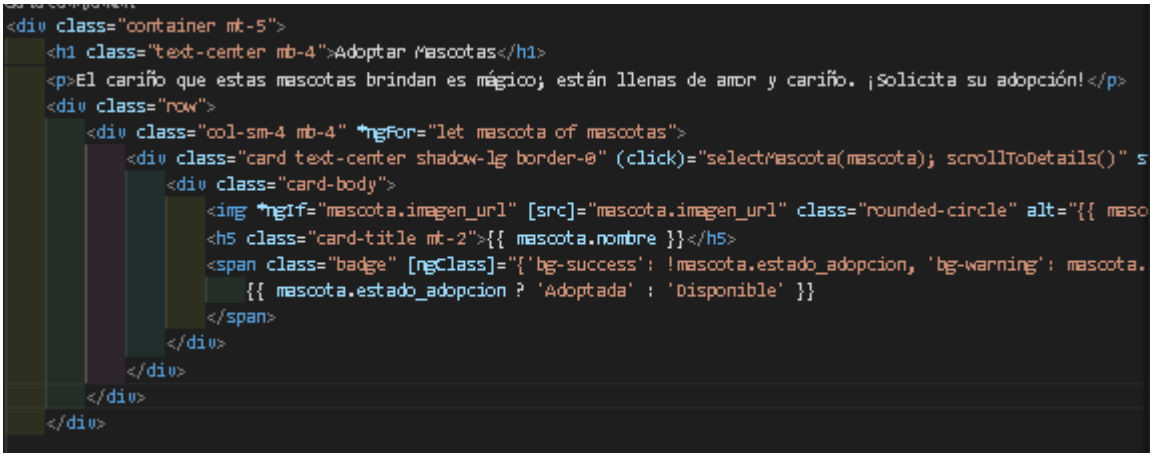
3.1 Implementación de Bootstrap

ng add @ng-bootstrap/ng-bootstrap se utiliza en Angular para instalar el paquete de Angular Bootstrap, que permite integrar los componentes de Bootstrap en el proyecto.

3.2 Generación de una Interfaz Ordenada

Utilizando las clases de Bootstrap, se puede estructurar una interfaz ordenada y atractiva.

Ejemplo de uso de Bootstrap:



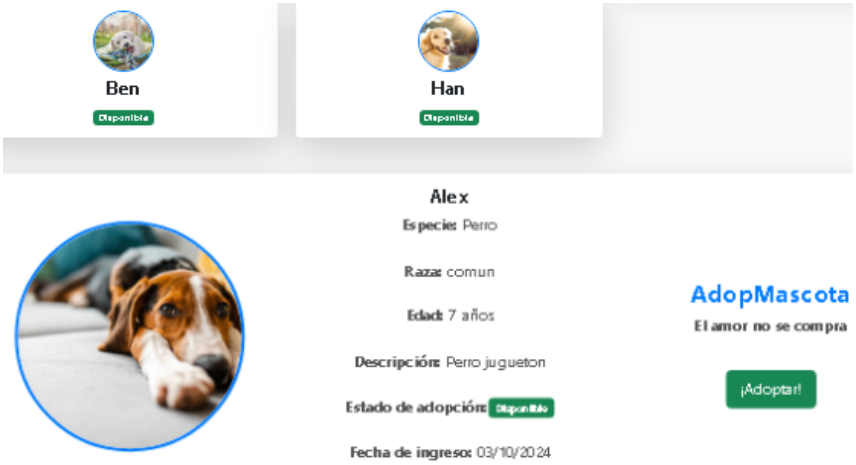
```
<div class="container mt-5">
    <h1 class="text-center mb-4">Adoptar Mascotas</h1>
    <p>El cariño que estas mascotas brindan es mágico; están llenas de amor y cariño. ¡Solicita su adopción!</p>
    <div class="row">
        <div class="col-sm-4 mb-4" *ngFor="let mascota of mascotas">
            <div class="card text-center shadow-lg border-0" (click)="selectMascota(mascota); scrollToDetails()" s
                <div class="card-body">
                    <img *ngIf="mascota.imagen_url" [src]="mascota.imagen_url" class="rounded-circle" alt="{{ masc
                    <h5 class="card-title mt-2">{{ mascota.nombre }}</h5>
                    <span class="badge" [ngClass]="{'bg-success': !mascota.estado_adopcion, 'bg-warning': mascota.
                        {{ mascota.estado_adopcion ? 'Adoptada' : 'Disponible' }}
                    </span>
                </div>
            </div>
        </div>
    </div>
```

- **`text-center`**: Centra el texto.
- **`mb-4`**: Añade un margen inferior de 4 unidades.
- **`row`**: Crea una fila en el sistema de grid.
- **`col-sm-4`**: Define una columna que ocupa 4 de 12 columnas en pantallas pequeñas y superiores.
- **`card`**: Crea un contenedor de tarjeta para agrupar contenido.
- **`shadow-lg`**: Agrega una sombra grande a la tarjeta.
- **`border-0`**: Elimina el borde de la tarjeta.
- **`card-body`**: Contenedor para el contenido de la tarjeta.
- **`rounded-circle`**: Hace que la imagen sea circular.
- **`mt-2`**: Añade un margen superior de 2 unidades al título.
- **`badge`**: Estiliza un elemento como etiqueta.

- **`bg-success`** y **`bg-warning`**: Aplica colores de fondo verde (disponible) y amarillo (adoptada) a la etiqueta.

3.3 Resultado Visual

Y al seleccionar la mascota muestra su información completa



Conclusión

La combinación de componentes en Angular, el uso de HTML5 y JavaScript, y la aplicación de estilos CSS con Bootstrap el proyecto se desarrolla satisfactoriamente, dando como resultado una página web bien estructuradas y visualmente atractivas desde mi punto de vista.

Ejecución de la página web de *AdopMasxotas*

Es importante recordar que nuestra base de datos creada se llama **ad_mascotas,** esta nos permitirá almacenar y gestionar los datos de: mascotas, usuarios, adopciones y solicitudes.

Comencemos

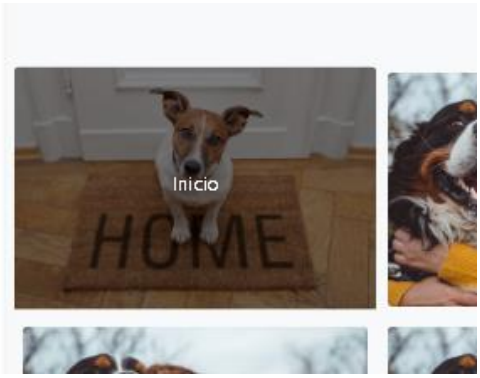
Tenemos un inicio de sesión el cual nos permitirá visualizar las diferentes ventanas según su rol:

Esta es la pagina de inicio que tiene un inicio de sesión y información crucial de la empresa **AdopMascotas:**

Como lo había mencionado anteriormente si el usuario es un administrador o un usuario podrá ver opciones diferentes

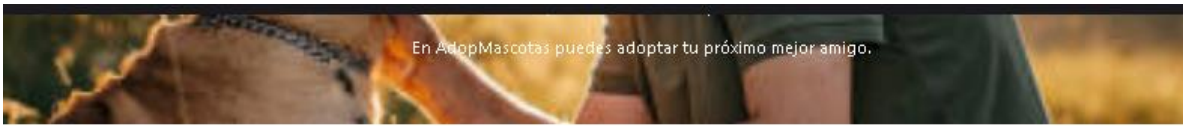
Administrador

El administrador tiene las opciones de Incio, usuarios, solicitudes, gestor de solicitudes, adopciones, mascotas. El nombre de la tarjeta aparece cuando se pasa el cursor sobre él.



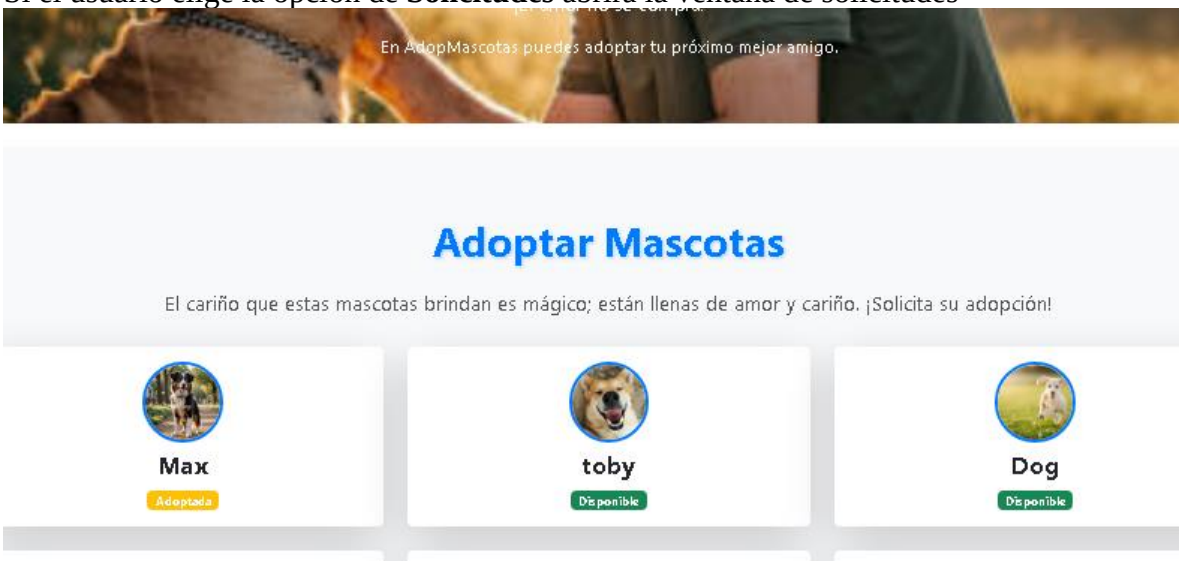
Usuario

El usuario solo podrá acceder a estas dos opciones:

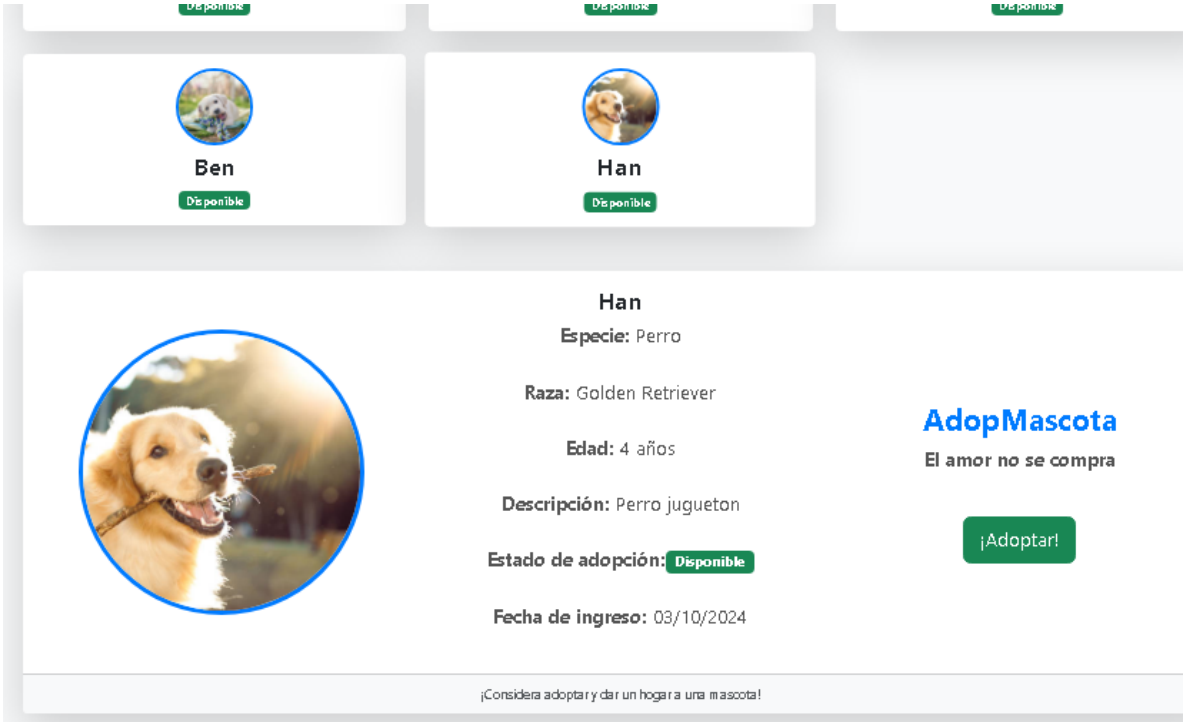


Continuemos con un rol de Usuario

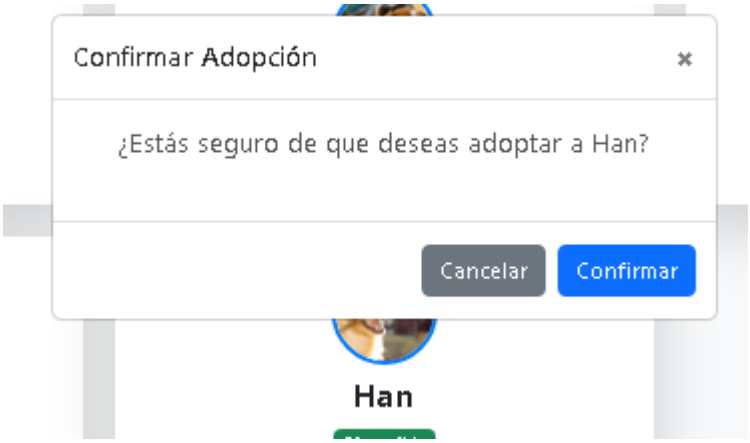
Si el usuario elige la opción de **Solicitudes** abrirá la ventana de solicitudes



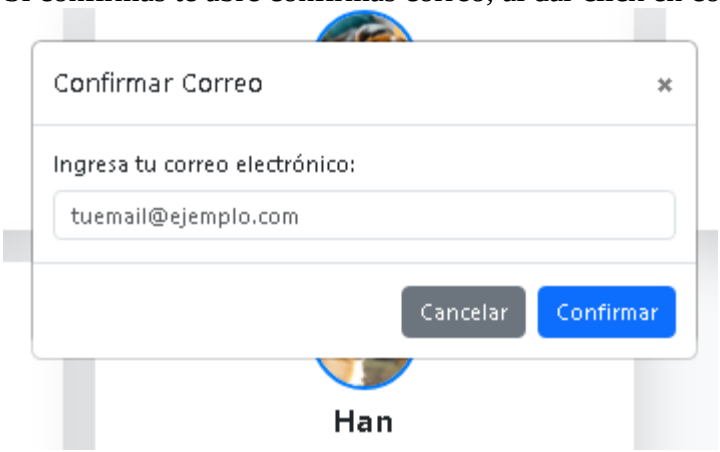
Al hacer click en uno de ellos podrá ve toda su información: en este caso se selecciona a **Han** y me muestra la información completa de la mascota



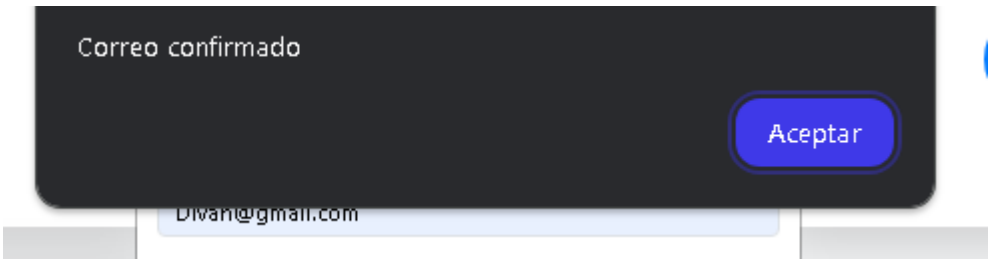
Si el usuario quiere adoptar esa solicitud presiona el botón de **adoptar**, este nos abrirá esta confirmación de adopción



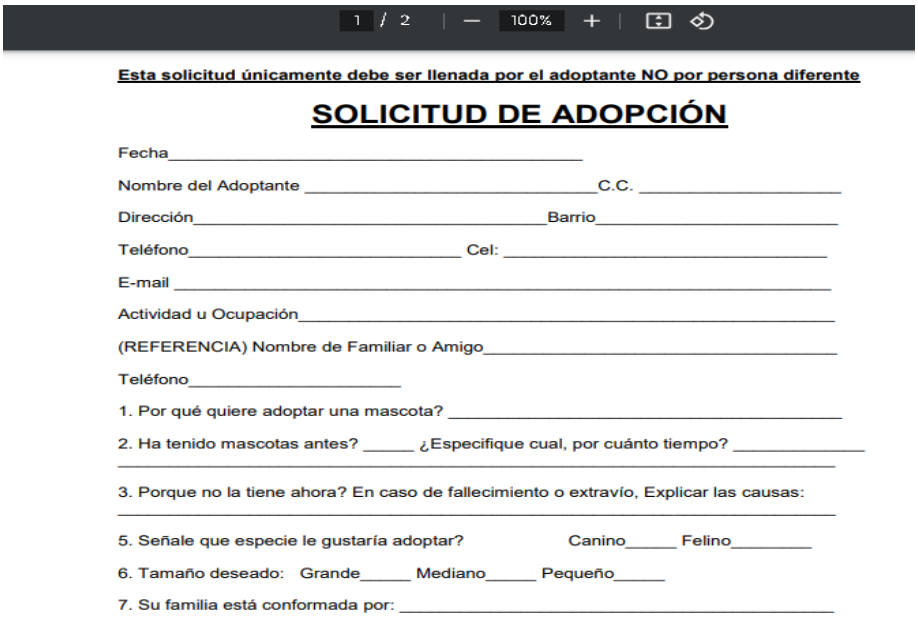
Si confirmas te abre confirmas correo, al dar click en confirmar se verifica el correo



Se despliega una alerta si es encontrado



Le damos **Aceptar** y abrira un formato de solicitud para que este lo diligencie



1 / 2 | 100%

Esta solicitud únicamente debe ser llenada por el adoptante NO por persona diferente

SOLICITUD DE ADOPCIÓN

Fecha_____

Nombre del Adoptante _____C.C. _____

Dirección_____Barrio_____

Teléfono_____ Cel: _____

E-mail _____

Actividad u Ocupación_____

(REFERENCIA) Nombre de Familiar o Amigo_____

Teléfono_____

1. Por qué quiere adoptar una mascota? _____

2. Ha tenido mascotas antes? _____ ¿Especifique cual, por cuánto tiempo? _____
_____

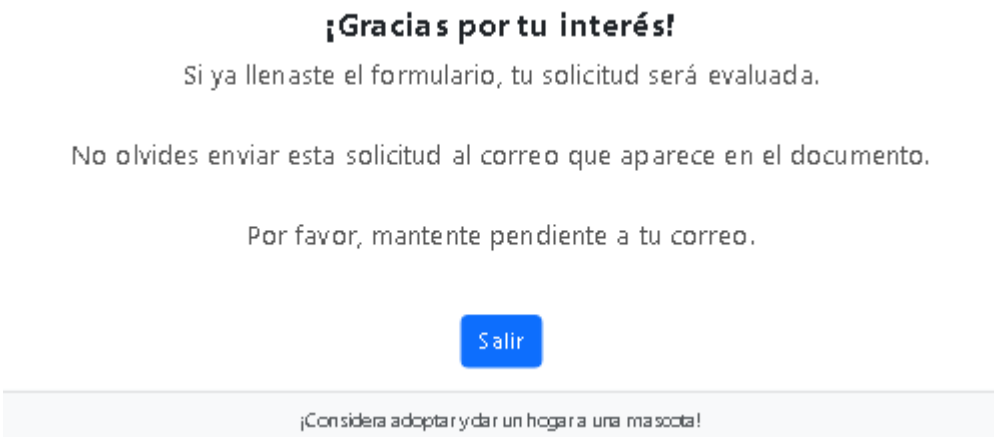
3. Porque no la tiene ahora? En caso de fallecimiento o extravío, Explicar las causas:
_____

5. Señale que especie le gustaría adoptar? Canino_____ Felino_____

6. Tamaño deseado: Grande_____ Mediano_____ Pequeño_____

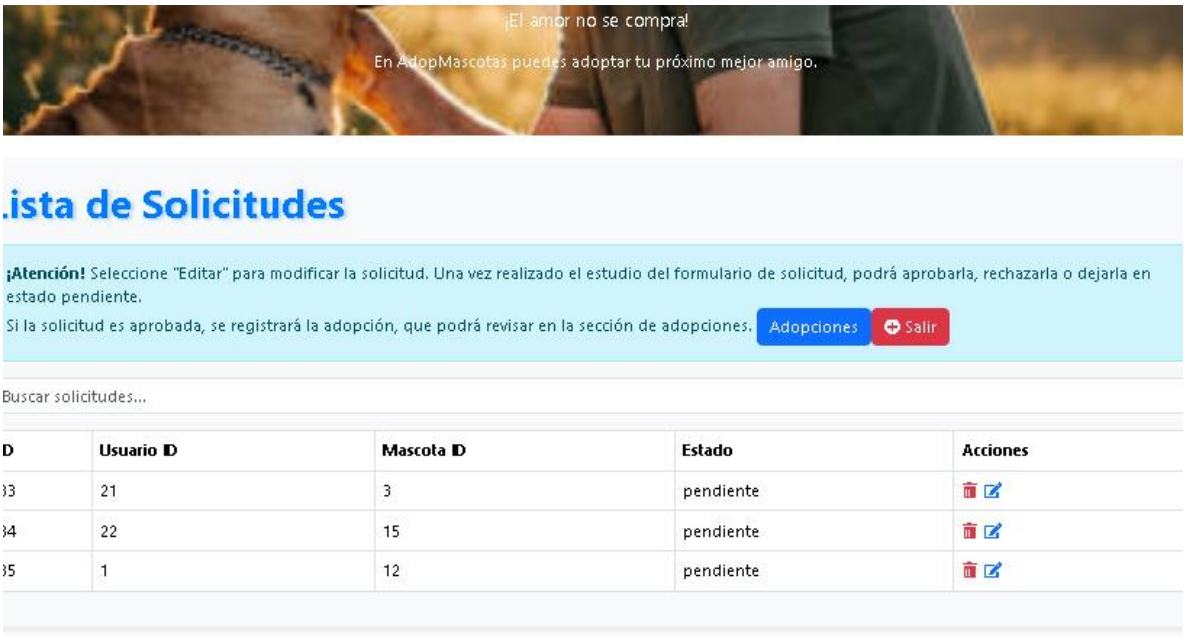
7. Su familia está conformada por: _____

Muestra un mensaje en pantalla

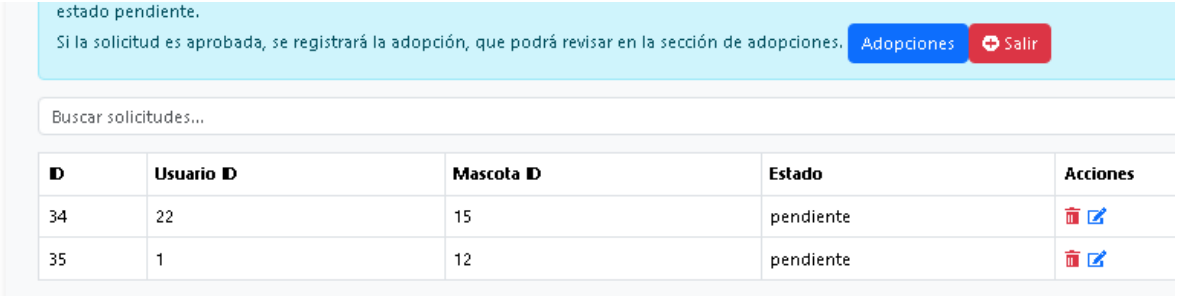


Administrador

Este será que una vez evaluada la solicitud la apruebe o rechace, seleccionando el botón de editar.



Sí es aprobada se registrará una nueva adopción y se eliminara de la tabla de solicitudes



Verificamos en adopciones

Independientemente de estas operaciones el usuario podrá editar,eliminar, buscar mascotas,

editar,eliminar, buscar usuarios, editar,eliminar, buscar solicitudes, editar,eliminar, buscar adopciones.

Gestión de mascotas

Gestión de usuarios

ID	Nombre	Email	Teléfono	Dirección	Fecha de Registro	Rol	Acciones
1	Jeason	jeisongelpud48@gmail.com	3174505358	ipiales	30/09/2024	Administrador	
8	Juan	juan@example.com	No disponible	No disponible	05/10/2024	Usuario	
13	steven	jeisonsy8@gmail.com	3174505358	ipiales	05/10/2024	Usuario	
14	Diego S	Diego48@gmail.com	3174505358	ipiales	05/10/2024	Usuario	
15	Rene	rene@gmail.com	3174505358	ipiales	05/10/2024	Usuario	
16	Carlos	carlos48@gmail.com	3174505358	ipiales	05/10/2024	Usuario	
18	juana	juana@gmail.com	3174505358	ipiales	05/10/2024	Usuario	

Y las adopciones y solicitudes que se implemento en el registro de la solicitud y la solicitud.

Observación final:

El proceso de implementar tanto el backend como el frontend en Angular fue muy enriquecedor e interesante. Combinar la lógica del backend con una interfaz moderna permitió un flujo de trabajo dinámico y altamente interactivo. La integración de componentes en Angular facilitó la reutilización de código, mientras que la creación de rutas seguras y el manejo del estado mejoró la experiencia del usuario. Desarrollar un sistema donde ambos mundos se complementan, ofreciendo una experiencia fluida, fue un reto gratificante que mostró la capacidad de Angular para manejar aplicaciones complejas de manera eficiente.