I.E.S. POLITÉCNICO HERMENEGILDO LANZ

DEPARTAMENTO DE INFORMÁTICA

FACE DOG

10 / junio / 2022 **Cristian Pérez López**

1 IDEAS PRINCIPALES

La idea de FaceDog surgió para cubrir la necesidad de espacio virtual dedicado única y exclusivamente a la temática de animales domésticos, dado que cada vez es un sentimiento más fuerte por parte de la sociedad.

También poder poner en contacto a usuarios con veterinarios con un solo click.

Facedog es una red social dedicada al espacio de las mascotas, en ella podrás encontrar a tus amigos y compartir contenido con ellos además de interactuar mediante chats y comentarios con los demás usuarios.
También existe la posibilidad de interactuar con perfiles de veterinarios, para consultar dudas sobre la salud de tu mascota o hablar con ellos mediante un chat.

2 BASE DE DATOS

2.1 TECNOLOGIA

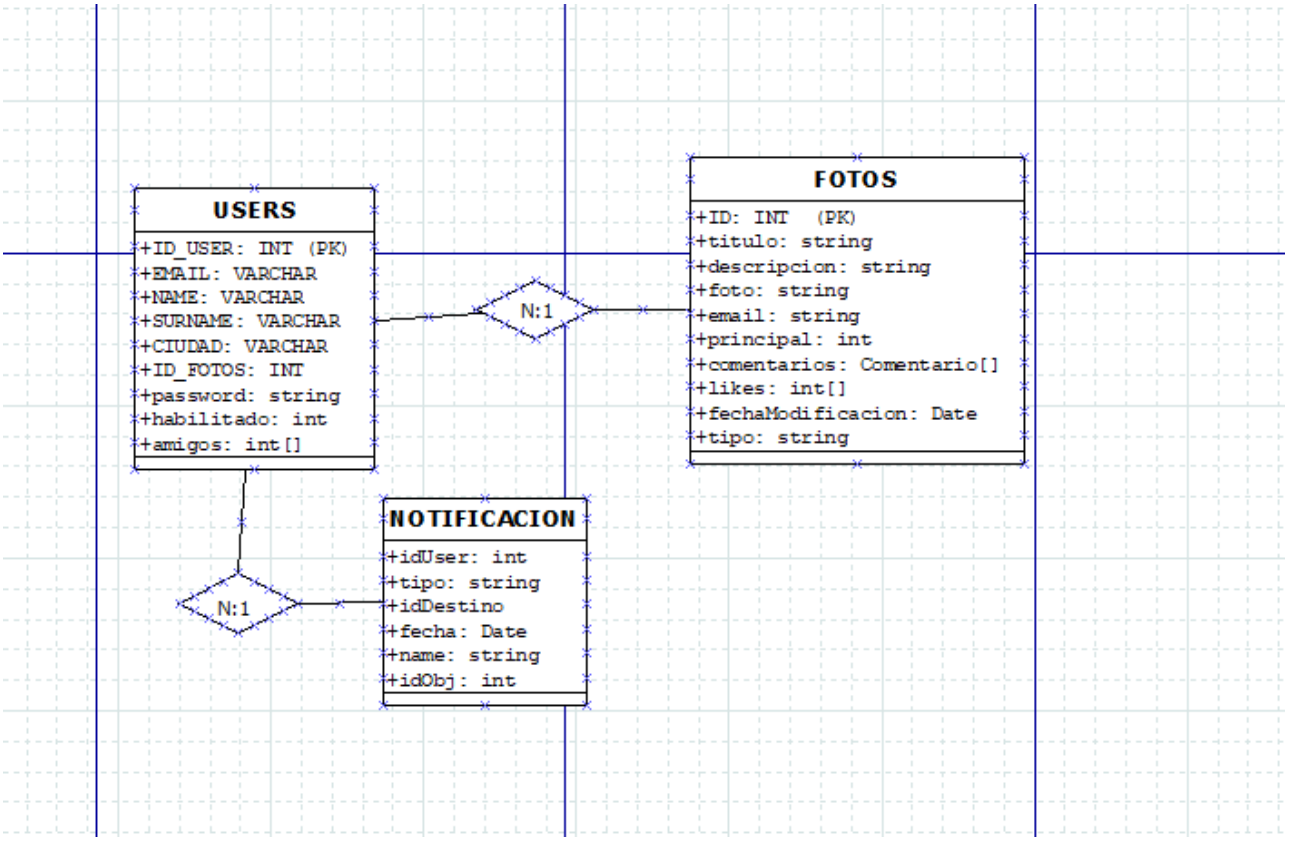
MongoDB es un sistema de base de datos NoSQL orientado a documentos de código abierto y escrito en C++, que en lugar de guardar los datos en tablas lo hace en estructuras de datos BSON (similar a JSON) con un esquema dinámico.

Si tuviéramos que resumir a una la principal característica a destacar de MongoDB, sin duda esta sería la velocidad, que alcanza un balance perfecto entre rendimiento y funcionalidad gracias a su sistema de consulta de contenidos. Pero sus características principales no se limitan solo a esto, MongoDB cuenta, además, con otras que lo posicionan como el preferido de muchos desarrolladores.

MongoDB es una base de datos orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Esto representa una de las diferencias más importantes con respecto a las bases de datos relacionales. Y resulta que no es que no es necesario seguir un esquema. Los documentos de una misma colección - concepto similar a una tabla de una base de datos relacional -, pueden tener esquemas diferentes

2.2 DIAGRAMA



2.3 COLECCIONES

Users: Esta colección recoge los documentos usuarios como propiedad a destacar la lista amigos, que se trata de una lista con los id de los usuarios con los que tiene la relación de amigos.

Fotos: Esta colección recoge los documentos de imágenes, se relacionan con la tabla usuarios mediante el email de estos, como propiedad a destacar sería la de comentarios, que guarda un array con Objetos comentarios.

Notificación: Esta colección recoge los documentos de las notificaciones, esta tabla va a ir guardando todas las notificaciones de cada usuario mientras estas no hayan sido revisadas por el usuario.

2.4 DESPLIEGUE

El despliegue se realiza con los propios servidores de MongoDB con la versión gratuita, esto limita bastante el uso de la base de datos ya que usamos una Ram compartida y un espacio muy limitado.

Enlace referencia:

-Documentacion para desplegar base de datos

3 (BACK) API REST

3.1 TECNOLOGIA

Lenguaje de programación: Python.
ApiRest: FastAPI.
Servidor ASGI : Uvicorn.

FastAPI es un marco web moderno y de alto rendimiento para crear API, basado en Python 3.6 y superior.
La característica más importante: ¡rápido! Rendimiento muy alto, comparable a NodeJS y Go. Basado en Starlette y Pydantic, es una razón importante para el alto rendimiento de FastAPI. También tiene las ventajas de una alta reutilización de código, fácil de usar y gran solidez. Personalmente, también creo que FastAPI tiene una gran ventaja: depuración de API conveniente, generación de documentación de API y capacidad directa para depurar API creadas por usted mismo, lo que destaca el valor de las aplicaciones prácticas.

Enlace de referencia:

Documentación FastApi

Uvicorn es una implementación de servidor web ASGI para python el cual nos va a permitir arrancar nuestra api .

Enlace referencia:

Documentación Uvicorn

3.2 SWAGGER

FastAPI incluye una interfaz de usuario basada en swagger, fácil e intuitivo de usar y con la que poder realizar las pruebas necesarias para poner tu API a punto sin ninguna necesidad de front

3.3 Conexión BD

```
mongo_client_local =
"mongodb+srv://Cristian:root@cluster0.1wun7.mongodb.net/test"
client = pymongo.MongoClient(mongo_client_local)
database = client["FACEDOG"]
conexion = database["USERS"]
```

Necesitamos una conexión con una base de datos MongoDB esta se la pasaremos a "`pymongo.MongoClient`" previa importación. Ahora solo tenemos que indicarle la base de datos con la que queremos trabajar y en caso opcional una tabla.

3.4 RUTAS

Método get de end-point para traer usuarios mediante propiedad email.

-Definimos la ruta @rutas.get al el cual le podemos añadir diferentes parámetros, en este ejemplo se define la ruta de la llamada, el tag de identificación para la interfaz Swagger y el tipo de respuesta.

-Definimos la función que define la lógica que utilizaremos para realizar la llamada a la base de datos y trabajar la información para transformarla en la respuesta esperada.

1º Retiramos todos los posibles espacio al email introducido

2º Dentro de un bloque try y llamamos a schema_get_user, un método que se encuentra en la carpeta eschemas el cual realiza verificaciones en caso de falar alguna verificación lanza la Exception `MultipleInvalid`

3º Definimos el modelo que va a recibir nuestra llamada

4º Hacemos uso del método find_one que nos proporciona la librería de MongoDB para buscar un único resultado, en este caso recibe dos parámetros, ambos un diccionario, con el primero le indicamos el filtro, y el segundo los parámetros que no queremos extraer añadiendo -1.

En caso de no encontrar el email en la base de datos lanza un HttpException con código 404.

3.5TOKEN

Con la librería jwt podemos generar y descripta tokens de acceso para restringir el uso de nuestra api
Para des encriptar un token solo necesitamos hacer uso del método decode de jwt y pesarle 3 parámetros, el token, el secreto (contraseña), y el algoritmo para desencriptarlo.
Para generar un token hacemos uso el método encode de jwt le pasamos el cuerpo del token, el cual tiene la fecha de caducidad la fecha en la que se creo y el cuerpo en el que podemos definir algunos atributos de uso recurrente.

El metdo auth_wapper lo añadimos a cada llamado que queramos confirmar si tiene acceso autorizado con token

3.6 EMAILS

El email se utiliza para confirmar la cuenta, la lógica seria la siguiente:

-El usuario al registrarse llama a la función enviarEmail.

-Esta función enviar un email al usuario con cogido html incrustado en la cual hay un link “pinche aquí para confirmar tu cuenta”.

-Al pinchar en el link se hace una petición a la Api con el email del usuario para actualizar lar propiedad habilitado.

Para llevar acabo este método se hizo uso de la librería smtplib de python

-Llamamos al método SMTP_SSL le pasamos el tipo de cuenta, el puerto, un contexto.

-Hacemos login con el método loguin.

-Enviamos el email llamando al método sendEmail pasando el email con el que enviaremos el correo, el destinatario y el mensaje que en este caso es un html incrustado.

3.7 DESPLIEGUE

El despliegue se realiza mediante un contenedor Docker y Heroku .

```
FROM python:3.9
COPY requirements.txt /tmp/requirements.txt
RUN pip install --no-cache-dir -r /tmp/requirements.txt
COPY . .

CMD uvicorn conexion:app --reload --host 0.0.0.0 --port $PORT
```

Aquí vemos la imagen necesaria para el despliegue

-**FROM** Le indicamos una imagen de python de la que pueda tirar

-**COPY** le indicamos las librerías necesarias en el archivo requieriments, y la ruta donde se encuentra y donde la va a ubicar en la MV.

-**RUN** le indicamos el comando para instalar todas las librerías necesarias

-**CMD** le indicamos el comando a ejecutar una vez construida la imagen hay que tener en cuenta que heroku establece el puerto de conexión aleatoriamente por lo tanto hay que crear el archivo **.env** del cual hará uso heroku para dar valor a la constante $**PORT**

Enlace de repositorio publico de apiFaceDogç:
[Repositorio de API]

4 (FRONT) ANGULAR

4.1 TECNOLOGIA

Lenguaje de programación: TypeScript.
FrameWork: Angular.

Angular: Angular es una plataforma de desarrollo, construida sobre TypeScript. Es un framework basado en componentes para crear aplicaciones web escalables. Una colección de bibliotecas bien integradas que cubren una amplia variedad de características, que incluyen enrutamiento, administración de formularios, comunicación cliente-servidor y más. Un conjunto de herramientas para desarrolladores que permiten desarrollar, compilar, probar y actualizar el código fuente de la aplicación.

4.2 APP-ROUTING

En este archivo definimos las rutas de nuestra app, gracias a RouteModule podemos definir los “path” y sus rutas hijas “children”. Podemos enrutar componentes y añadirles Guard, y Canctive para controlar el acceso a esas rutas.

```
path: 'home',
  component: HomeComponent,
  children: [
   {path : 'inicio',component:InicioComponent,canActivate: [AppGuard] },
```

4.3 APP-MODULE

En este archivo vamos a importar todos los módulos, componentes y demás que vayamos a utilizar en nuestra app.

Se diferencian dos partes:

-Declaration: donde vamos añadir todos los componentes que utilicemos de manera global

-Imports : donde vamos a importar todos los módulos y librerías que necesitemos en nuestra app

4.4 COMPONENTES

Hay 23 componentes en la app, entre ellos se pueden destacar Inicio, SubirFoto, Administracion, ProgresSnniper,Registro,VistaVeterinario.

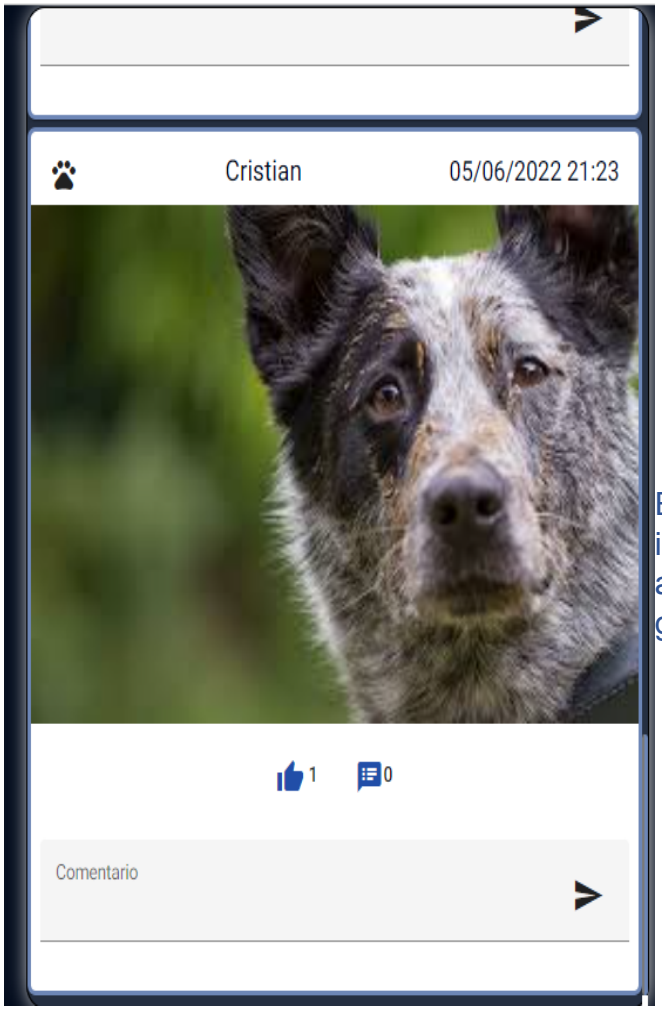
4.4.1 INICIO

Inicio: Este es el componente principal, al que el usuario accede al entrar en la app. Este componente recoge información del usuario mediante su Token y haciendo uso de la librea JWT.

Hay tres partes principales:

Aquí encontramos un menú con diferentes opciones para que el usuario pueda interaccionar con la app.

1

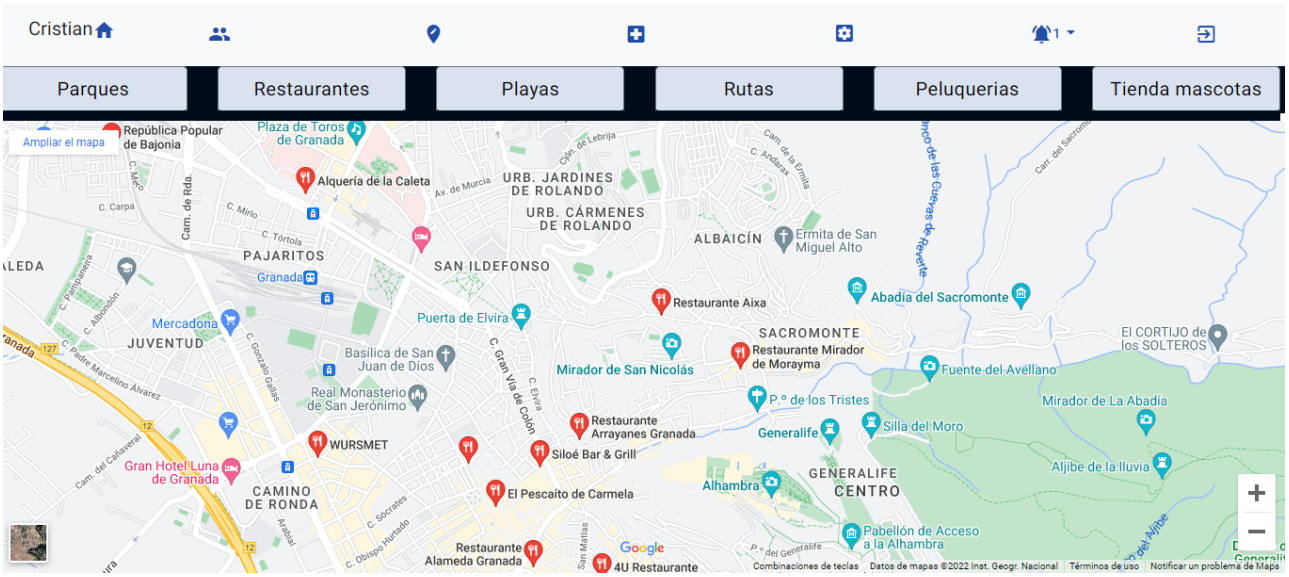


El componente noticias, que muestra imágenes de usuarios vinculados como amigos y en la cual puedes indicar que te gusta la publicación o comentarla.

El componente chat, con el que el usuario puede chatear en vivo con los usuarios registrados.

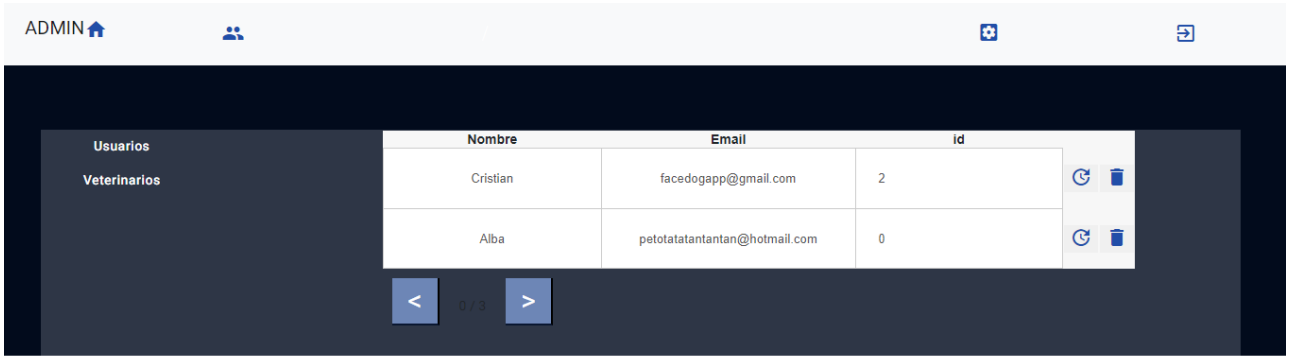
4.4.2 ZONAS

Aquí el usuario puede encontrar rápidamente ubicaciones relacionadas con las mascotas.



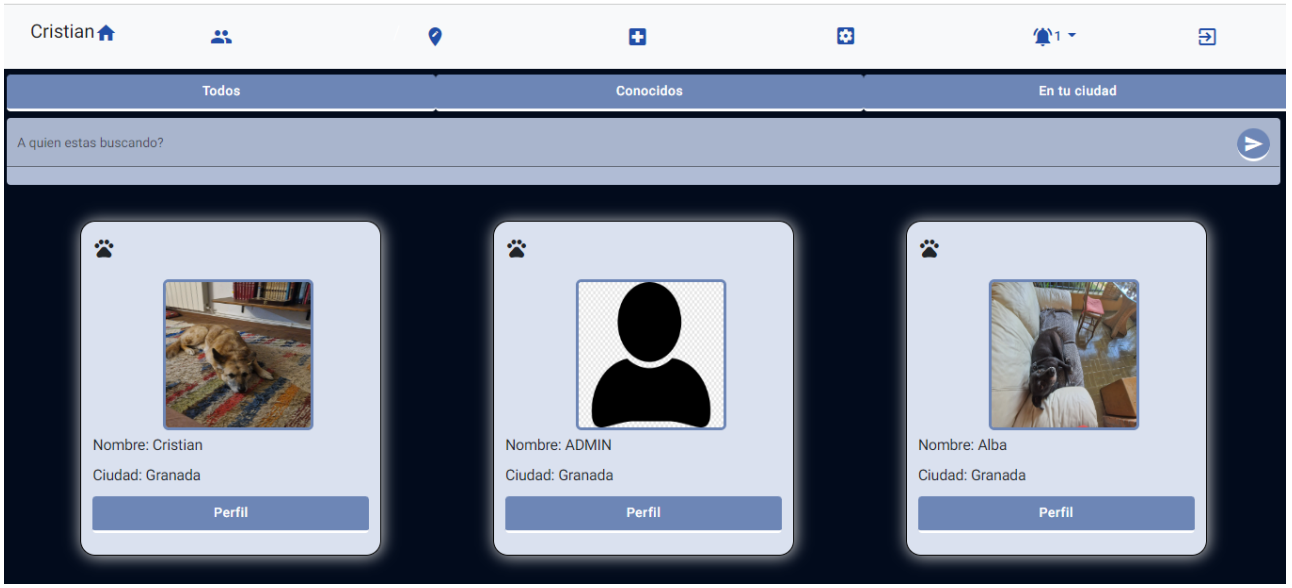
4.4.3 ADMINISTRACION

En este componente gestionamos los usuarios registrados en facedog, podemos borrarlos o actualizar sus propiedades.



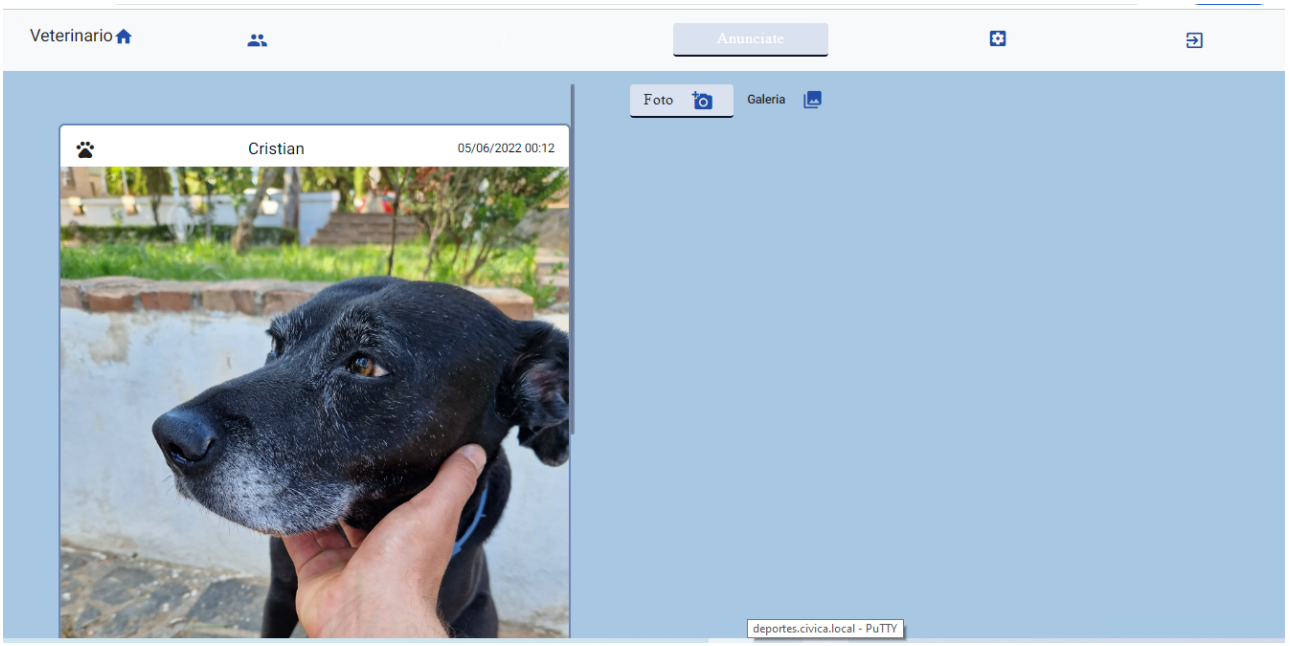
4.4.4 AMIGOS

Aquí el usuario puede encontrar mas perfiles,por ciudad, amigos, todos o un buscador que busca por nombre.



4.4.5 ROL-VETERINARIO

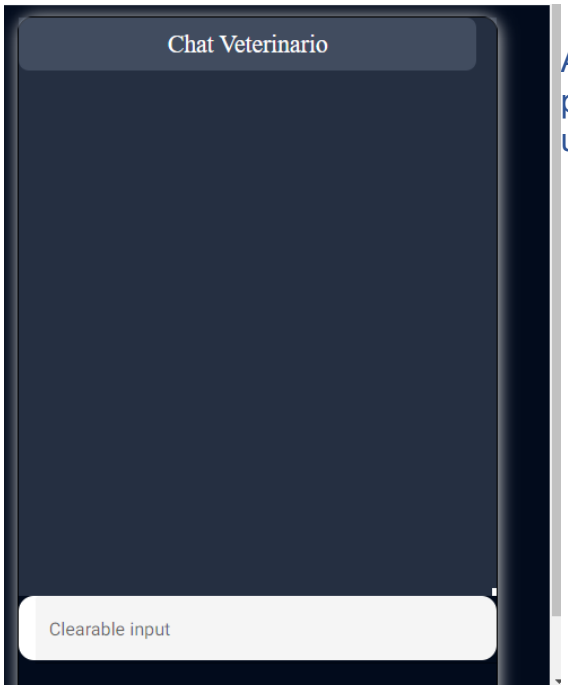
Aqui acceden los roles de veterinario, se cargan las fotos que los usuarios han subido en el componente consultas, y aquí pueden comentar las fotos los veterinarios, para dar citas, aconsejar o cualquier cosa que puedan solucionar.



4.4.5 CONSULTAS

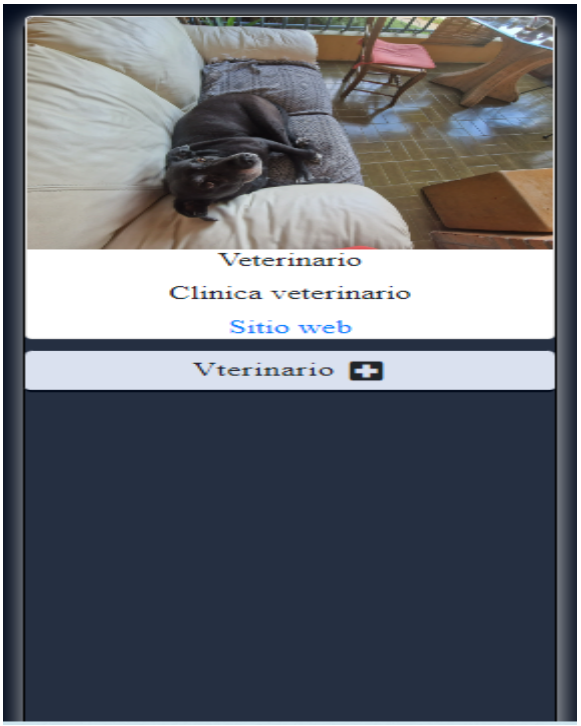
Su distribuciones muy similar a la del componente inicio.

Aquí el usuario ve los comentarios del veterinario y en caso de que le haya resultado de ayuda puede pinchar en el botón, este una vez clicado desaparece y pone el comentario con un fondo verde, para indicar que el comentario resulto de ayuda.



Al igual que el componente inicio tenemos un chat para hablar en directo con veterinarios o mas usuarios interesados.

Aquí hay un botón que permite añadir una imagen y compartirla con veterinarios y una tarjeta que va pasando anuncios de veterinarios cada 6 segundos



4.5 SERVICIOS

UsuariosService: Aquí vamos a realizar todas las llamadas y suscripciones a solicitudes HTTP y HTTPS que hagamos a nuestra api. Utilizamos la librera HttpClient para poder realizar la conexión.

User-register: Aquí recogemos información de el usuario de uso recurrente como su rol, nombre, email y un método de jwt que verifica la caducidad del token, en caso de estar caducado redirecciona al usuario al componente **Login.**

Message: Este servicio nos proporciona un componente de Material el cual nos proporciona un método al que le podemos pasar diferente propiedades, se hace uso del texto que lleva el mensaje y de el elemento cerrar, para poder utilizar este servicio hay que declararlo en el provaider de el archivo appModule, le podemos asignar propiedades como la duración que mantendrá el mensaje activo.

LoaderService: Este servicio se utiliza para llamar al componente ProgresSnipper , tiene una propiedad a la cual le cambiamos el valor de true a false según estemos realizando una petición o no, para mostrar la componente ProgresSn

LoaderInterceptor: Aqui interceptamos todas las solicitudes que realizamos http o https, al detectar una petición llama a el método .show() de LoaderService y poner a true y asi cargar el componente ProgresSnipper al terminar la solicitud llamamos al método .hide().

Http-interceptor: Aquí interceptamos todas las solicitudes Htpp y Htpps para añadir el token al header de esta, y así conseguir el acceso necesario de la api, que requiere de token valido.

DocumenteService: Este servicio comunica el front con el servidor del chat, se documenta el el punto **5 CHAT**

4.6 Guards

AdminGuard: Utilizando al clase de AngularRouter CanActive podemos verificar el rol del usuario y devolver un boleano según encontremos. Este metodo se puede añadir en la solicitud de registro y así redireccionar al usuario a la parte que nos interese.

Clase AdminGuard:

```
@Injectable({
  providedIn: 'root'
})
export class AdminGuard implements CanActivate {
  userRole:string=""
  constructor(private  jwt :JwtHelperService, private router : Router, private userService : UsuriosService){
  }
  canActivate (

    ): Observable<boolean> | Promise <boolean  | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree{
          You, hace 3 semanas • fron …

    this.userRole =this.jwt.decodeToken(localStorage.getItem("token")!).sub.roles
      if(this.userRole === "Admin"){
          return true
      }else{
          this.router.navigate(['/home/administrador'])
          return false;
      }
    }

}
```

4.7 Despliegue

El despliegue se realiza con **FireBase,** para lograrlo hay que crear un cuenta de firebase registrar tu aplicación, una vez registrada ha que vincular el cmd ubicado en la raíz del proyecto y hacer un firebase login, una vez logueado el funcionamiento es muy sencillo, basta con hacer un build de la app y hacer un firebase deploy se encarga de recoger la carpeta dist con el proyecto compilado y lanzarlo en una ruta.

App desplegada

5 CHAT

Hay un chat global en el componente inicio, con el que puedes chatear con cualquier usuario registrado y que se encuentre activo en ese momento, también hay otro chat en el componente consultas, este es un chat en el que también participan los perfiles con el rol de veterinario.

5.1 Tecnología

Se utiliza Socket.io para la comunicación entre cliente servidor, es una tecnología que permite comunicación bidireccional sobre un único Socket TCP.

5.2 Conexión

Se ha creado un archivo java script (servidor) en la raíz del proyecto, ahí declaramos la lógica que llevara tipo de comunicación.

En el servidor se declara el scket con el protocolo de comunicación y llamando al método .on() con el que permanecemos a la escucha, a continuacion se delcara un método que recibe dos parámetros, el nombre de la función y el objeto que recibe del cliente, en nuestro caso uno de los métodos que se utiliza es el “editDoc” que recibe un objeto documento y comprueba por id , el cual se lo asignamos desde el cliente, su identificación para sobre escribirlo, seguidamente llamamos al método .emit, este recibe

dos parámetro, el nombre de la función y el objeto a emitir, y con e emitimos el objeto a todos los sockets conectados en ese momento. Con el método .listen() abrimos el puerto 3000.

En el cliente e el archivo app.module declaramos una variable de tipo SocketIoConfig que guardara la url del servidor.

Por otro lado creamos un servicio que sera el encargado de comunicar con el servidor, para ello declaramos los objetos que recibiremos mediante eventos del servidor y los métodos que emitirán al servidor el objeto, creamos la función editDcument() que recibe un objeto documento y con el método .emit() de Socket.io emitimos al servidor.

5.3 Despliegue

EL despliegue se realiza creando un DockerFile con el que instalaremos las libreiras necesarias en la ruta indicada y ejecutaremos el comando para arrancar el servidor, como Heroku asigna los puertos de manera aleatoria, se crear en la raíz del proyecto “chat” el archivo .env para que heroku pueda sobrescribir el puerto, le asignamos ese puerto como variable en el comando.