FAUNA SV

Manual Técnico-APP

Contenido

Acerca de la APP Fauna SV: 3
Requerimientos del sistema móvil: 3
Requerimientos del sistema para Android Studio: 3
EndpointInterface. 3
getAvistamientos(): 4
addAvistamiento(): 4
getAvistamientosByDistance(): 4
getDistanceByPoints(): 5
getToken(): 5
getAnimalesByName(): 5
getAvistamientoByUserorAnimal(): 6
Utilización de Retrofit. 6
Configuración o declaración de Retrofit. 6
Utilización de Call de Retrofit para obtener respuestas de nuestra API. 6

Acerca de la APP Fauna SV:

La APP de Fauna SV es una herramienta que permite a los entusiastas de la naturaleza el poder tener un lugar en donde puedan tener toda la información que ellos necesiten sobre los avistamientos de las distintas especies de animales que habitan en El Salvador, además de poder agregar ellos avistamientos propios.

Esto les permitirá tener una mejor idea de los lugares en que las diferentes especies de animales habitan, para así poder saber a qué lugar tienen que viajar para tener la mayor oportunidad de documentar las especies que ellos deseen.

Este manual de usuario presenta un resumen de cómo se utilizará la APP de Fauna SV.

Requerimientos del sistema móvil:

Para poder asegurar el mejor funcionamiento de la APP se le aconseja que se utilice un celular o Tablet que tenga las siguientes funcionalidades.

Sistema Operativo	Android 5.0
Funcionalidad del Dispositivo	GPS

Requerimientos del sistema para Android Studio:

Para poder modificar o agregar código a la APP se necesitará el uso de Android Studio, la cual necesita como requisitos mínimos del sistema lo siguiente:

Sistema Operativo	Microsoft® Windows® 7/8/10 (32- or 64-bit), Mac® OS X® 10.10 (Yosemite) or higher, up to 10.14 (macOS Mojave), GNOME or KDE desktop Tested on gLinux based on Debian (4.19.67-2rodete2).
RAM	4 GB RAM mínimo, 8 GB RAM recomendado
Disco	2 GB de espacio mínimo, 4 GB recomendado
JDK mínima	Java Development Kit 8
Resolución de pantalla mínima	1280x800

EndpointInterface.

Esta es la interfaz donde tendremos todos nuestros endpoints de Retrofit, que son los que se van a contactar con nuestra REST API para devolvernos toda la información que necesitamos, estos endpoints son los siguientes:

```
@GET("avistamientos")
Call<List<Avistamiento>> getAvistamientos();
```

getAvistamientos():

Es un endpoint que nos va a devolver una lista de objetos de tipo Avistamiento, este método no recibe nada y se utiliza para poder mostrar todos los avistamientos en el mapa y en la lista. Se usa la etiqueta @GET para informar que estaremos haciendo un request por medio de GET, y dentro de la etiqueta @GET tenemos "avistamientos", ya que esta es la locación URL del endpoint en la API.

```
@Headers("Content-Type: multipart/form-data")
@Multipart
@POST("avistamientos/")
Call<Avistamiento> addAvistamiento(@Header("Authorization") String token,
                                   @Part("geom") String geom,
                                   @Part("confirmado") String confirmado,
                                   @Part("fecha_hora") String fecha_hora,
                                   @Part("fotografia") RequestBody fotografia,
                                   @Part("descripcion") String descripcion,
                                   @Part("animal") String animal);
```

addAvistamiento():

Este endpoint se tuvo un problema al implementarlo y es que no se pudo comunicar correctamente con la REST API de Django, se recomienda ver la documentación específicamente de @Multipart en Retrofit, y como esta se puede comunicar con una REST API de Django.

```
@GET("getcercanos/point={location}&m={distance}")
Call<List<Integer>> getAvistamientosByDistance(@Path("location") String point, @Path("distance") String distance);
```

getAvistamientosByDistance():

Es un endpoint que nos va a regresar una lista de enteros, que serán los id's de todos los avistamientos que se encontraron en una determinada distancia, definida por el usuario, medida en metros. Se usa la etiqueta @GET para informar que estaremos haciendo un request por medio de GET, y dentro de la etiqueta @GET tenemos "getcercanos/point={location}&m={distance}", ya que esta es la locación URL del endpoint en la API. Se utiliza la etiqueta @Path("name") para indicar que en la URL de nuestro endpoint en la REST API, queremos que se agregue nuestro @Path("name") dentro de {name} en la URL. En nuestro caso se reemplazará el valor del String point en la parte de {location} de la URL y seria el mismo procedimiento para las otras etiquetas @Path.

```
@GET("getdistpuntos/point1={location1}&point2={location2}")
Call<Double> getDistanceByPoints(@Path("location1") String location1, @Path("location2") String location2);
```

getDistanceByPoints():

Es un endpoint que nos va a regresar un valor de tipo Double, que serán la distancia que existe entre la locación actual del usuario y un avistamiento en específico. Se usa la etiqueta @GET para informar que estaremos haciendo un request por medio de GET, y dentro de la etiqueta @GET tenemos "getdistpuntos/pint1={location1}&point2={location2}", ya que esta es la locación URL del endpoint en la API. Se utiliza la etiqueta @Path("name") para indicar que en la URL de nuestro endpoint en la REST API, queremos que se agregue nuestro @Path("name") dentro de {name} en la URL. En nuestro caso se reemplazará el valor del String location1 en la parte de {location1} de la URL y seria el mismo procedimiento para las otras etiquetas @Path.

```
@POST("api/token/")
Call<Token> getToken(@Body User user);
```

getToken():

Es un endpoint que nos va a retornar un objeto de tipo Token, este objeto de tipo Token es el que tendrá la información del Token de login. Se utiliza la etiqueta @POST para indicarle a la REST API que le enviaremos un objeto o dato a la API, y dentro de la etiqueta @POST tenemos "api/token/" que es la locación URL del endpoint en la API. Utilizamos @Body para mandar un objeto completo a nuestra API, en este caso un objeto de tipo User.

```
@GET("animal/")
Call<List<Animal>> getAnimalesByName(@Query("search") String animal);
```

getAnimalesByName():

Es un endpoint que nos retornara una lista de animales, está la utilizaremos como un filtro, para buscar el nombre de un animal en específico. Se usa la etiqueta @GET para informar que estaremos haciendo un request por medio de GET, y dentro de la etiqueta @GET tenemos "animal/", ya que esta es la locación URL del endpoint en la API. Luego usamos la anotación de @Query, que es la que usamos cuando la url nos pide un dato, este @Query("search") en el navegador pasa a ser animal/?search=.

```
@GET("avistamientos/")
Call<List<Avistamiento>> getAvistamientoByUserorAnimal(@Query("search") String data);
```

getAvistamientoByUserorAnimal():

Es un endpoint que nos retornara una lista de avistamientos, está la utilizaremos cuando queramos filtrar los avistamientos ya sea por nombre de usuario o por nombre de animal. Se usa la etiqueta @GET para informar que estaremos haciendo un request por medio de GET, y dentro de la etiqueta @GET tenemos "avistamientos/", ya que esta es la locación URL del endpoint en la API. Luego usamos la anotación de @Query, que es la que usamos cuando la url nos pide un dato, este @Query("search") en el navegador pasa a ser animal/?search=, y esta la utilizaremos para poner nuestra data y filtrar la busqueda.

Utilización de Retrofit.

Configuración o declaración de Retrofit.

Para poder hacer uso de la interfaz y de todos los endpoints que Retrofit nos facilita primero tendremos que hacer una declaración de Retrofit y de la interfaz para poder acceder a todos los métodos.

```
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .build();

//se crea el objeto de nuestra interfaz, que es en donde accederemos a todas las funcionalidades de la API
EndpointInterface apiService = retrofit.create(EndpointInterface.class);
```

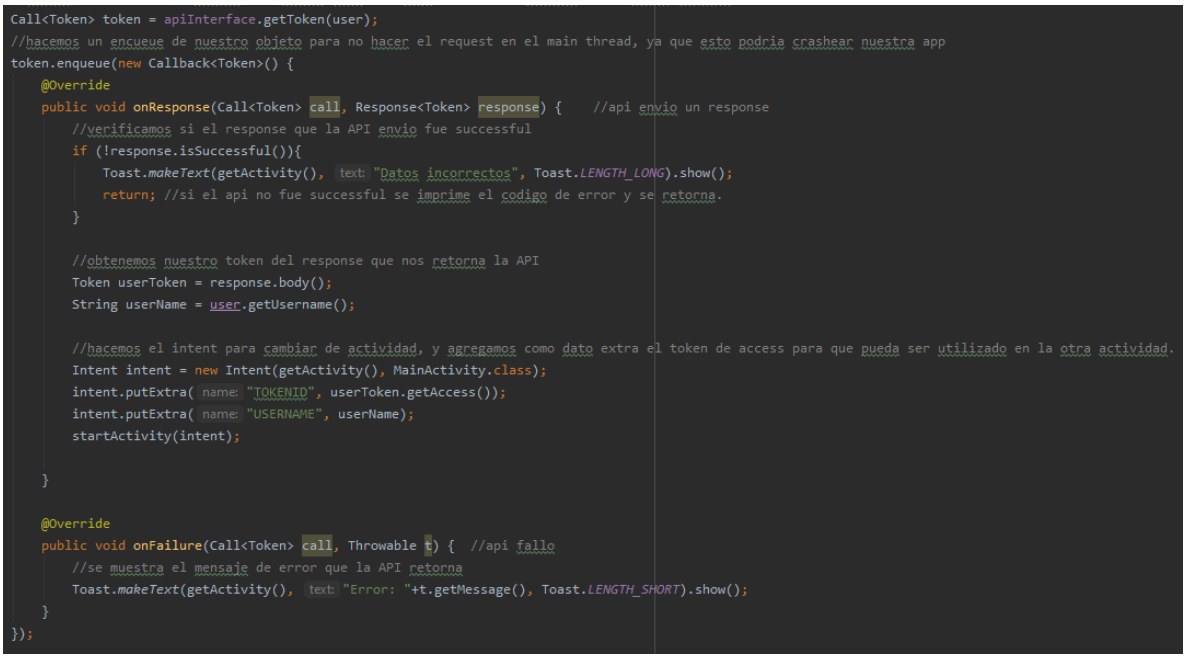
Primero creamos nuestro objeto de Retrofit y a este le tendremos que agregar un Builder, el builder llevara primero cual es la URL base de nuestra API, en este caso la BASE_URL del proyecto es https://faunaelsalvador.herokuapp.com/. Luego le agregaremos un converterFactory que nos ayudara a consumir los Json que responderá la API para poder convertirlos en objetos que JAVA pueda entender, en este caso utilizaremos Gson.

Luego creamos el objeto de nuestra interfaz y la vamos a igualar al .class de nuestra interface, pero primero le pasaremos a Retrofit esta clase para que la pueda adaptar y se pueda consumir con relativa facilidad. Es por eso que nuestro objeto EndpointInterface está siendo igualado a Retrofit.create(EndpointInterface.class).

Utilización de Call de Retrofit para obtener respuestas de nuestra API.

Para poder conectar y recibir los datos de nuestra API utilizamos un objeto llamado Call<Type> donde Type debe de ser el tipo de objeto que la respuesta de la API nos va a brindar.

Se explicará con el siguiente ejemplo:



```
Call<Token> token = apiInterface.getToken(user);
//hacemos un encueue de nuestro objeto para no hacer el request en el main thread, ya que esto podria crashear nuestra app
token.enqueue(new Callback<Token>() {
    @Override
    public void onResponse(Call<Token> call, Response<Token> response) {    //api envio un response
        //verificamos si el response que la API envio fue successful
        if (!response.isSuccessful()){
            Toast.makeText(getActivity(), text: "Datos incorrectos", Toast.LENGTH_LONG).show();
            return; //si el api no fue successful se imprime el codigo de error y se retorna.
        }

        //obtenemos nuestro token del response que nos retorna la API
        Token userToken = response.body();
        String userName = user.getUsername();

        //hacemos el intent para cambiar de actividad, y agregamos como dato extra el token de access para que pueda ser utilizado en la otra actividad.
        Intent intent = new Intent(getActivity(), MainActivity.class);
        intent.putExtra( name: "TOKENID", userToken.getAccess());
        intent.putExtra( name: "USERNAME", userName);
        startActivity(intent);

    }

    @Override
    public void onFailure(Call<Token> call, Throwable t) {  //api fallo
        //se muestra el mensaje de error que la API retorna
        Toast.makeText(getActivity(), text: "Error: "+t.getMessage(), Toast.LENGTH_SHORT).show();
    }
});
```

Utilizamos el Call<Token> ya que la API nos retornara un objeto de tipo Token, a este objeto le igualamos nuestro objeto de interfaz y accedemos al método deseado, en este caso ingresamos a apiInterface.getToken(user) y le brindamos el objeto de tipo user que es el que necesita.

Luego de esto debemos hacer que nuestro objeto de tipo call empiece a hacer la transacción, esto lo haremos con enqueue, es importante usar enqueue ya que, si la transacción se hace en el main thread, puede hacer que nuestra aplicación falle, enqueue evita que se haga en el main thread para que este problema no se dé. A nuestro enqueue le brindamos un nuevo objeto de tipo Callback<Type>, donde type será el tipo de objeto que esperamos en la respuesta, en este caso será Callback<Token>. Este Callback tiene dos métodos predefinidos, los cuales son onResponse y onFailure.

onResponse: este método se ejecutará siempre que la API retorne alguna respuesta, OJO, que retorne una respuesta no confirma que sea una respuesta exitosa.

onFailure: este método se ejecuta cuando la API falla y no retorna nada, en este vamos a mostrar al usuario que hubo un error, y cuál es el mensaje de ese error, para que tenga una mejor idea de que fue lo que fallo.

Para nuestro método de onResponse lo primero que tenemos que verificar es si la response fue exitoso, para esto hacemos un if(!response.isSuccessful()) lo cual retornara True si la response NO fue exitosa, si no fue exitosa nosotros mostramos un mensaje al usuario de cual pudo haber sido el error y retornamos para que el código no se siga ejecutando.

Si if(!response.isSuccessful()) es False, entonces el código se sigue ejecutando y es ya en este punto donde podemos obtener los objetos que vienen de respuesta de nuestra API. Para esto creamos un objeto del tipo necesario y se lo igualamos a response.body(), en este caso hacemos esto en la línea de:

Token userToken = response.body();

Ahora nuestro userToken tiene toda la información que venía en el response de nuestra API.

Esta es los pasos en general para poder hacer cualquier request a nuestra API con cualquiera de los endpoints que tenemos definidos en nuestra interfaz.