

Retrofit

# How to bring to your project

```
compile 'com.squareup.retrofit2:retrofit:2.3.0'  
compile 'com.squareup.retrofit2:converter-gson:2.3.0'
```

# What you need

- Model class which is used to map the JSON data to
- Interfaces which defines the possible HTTP operations
- Retrofit.Builder class - Instance which uses the interface and the Builder API which allows defining the URL end point for the HTTP operation.

# Model

## Simple class with setters and getters

```
public class TimeZoneApiResponse
{
    @SerializedName("status")
    @Expose
    private String status;
    @SerializedName("message")
    @Expose
    private String message;
    @SerializedName("countryCode")
    @Expose
    private String countryCode;

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getCountryCode() {
        return countryCode;
    }

    public void setCountryCode(String countryCode)
    {
        this.countryCode = countryCode;
    }
}
```

# Interface

```
public interface TimeZoneAPI {  
  
    @GET("get-time-zone")  
    Call<TimeZoneApiResponse> getTimeZone(@Query("key") String apiKey,  
                                          @Query("format") String format,  
                                          @Query("by") String searchBy,  
                                          @Query("lat") String latitude,  
                                          @Query("lng") String longitude);  
}
```

There are five built-in annotations: GET, POST, PUT, DELETE, and HEAD  
Another annotations for data providing: Query, Path, Body

# Retrofit.Builder

```
public TimeZoneAPI getTimeZoneAPI() {
    return new Retrofit.Builder()
        .baseUrl("http://api.timezonedb.com/v2/")
        .client(initClient())
        .addConverterFactory(GsonConverterFactory.create())
        .build().create(TimeZoneAPI.class);
}

@NonNull
private OkHttpClient initClient() {
    HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
    interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
    return new OkHttpClient.Builder()
        .connectTimeout(CLIENT_TIMEOUT_MILLIS, TimeUnit.MILLISECONDS)
        .addNetworkInterceptor(new StethoInterceptor())
        .addInterceptor(interceptor)
        .build();
}
```

# Authorization

```
OkHttpClient okHttpClient = new
OkHttpClient().newBuilder().addInterceptor(new Interceptor() {
    @Override
    public okhttp3.Response intercept(Chain chain) throws
IOException {
        Request originalRequest = chain.request();

        Request.Builder builder =
originalRequest.newBuilder().header("Authorization",
        Credentials.basic("aUsername", "aPassword"));

        Request newRequest = builder.build();
        return chain.proceed(newRequest);
    }
}).build();
```

# What you get when create a request

```
/**
 * An invocation of a Retrofit method that sends a request to a webserver and returns a response.
 * Each call yields its own HTTP request and response pair. Use {@link #clone} to make multiple
 * calls with the same parameters to the same webserver; this may be used to implement polling or
 * to retry a failed call.
 *
 * <p>Calls may be executed synchronously with {@link #execute}, or asynchronously with {@link
 * #enqueue}. In either case the call can be canceled at any time with {@link #cancel}. A call that
 * is busy writing its request or reading its response may receive a {@link IOException}; this is
 * working as designed.
 *
 * @param <T> Successful response body type.
 */
public interface Call<T> extends Cloneable {
    /**
     * Synchronously send the request and return its response.
     *
     * @throws IOException if a problem occurred talking to the server.
     * @throws RuntimeException (and subclasses) if an unexpected error occurs creating the request
     * or decoding the response.
     */
    Response<T> execute() throws IOException;

    /**
     * Asynchronously send the request and notify {@code callback} of its response or if an error
     * occurred talking to the server, creating the request, or processing the response.
     */
    void enqueue(Callback<T> callback);
}
```



# How to deal with a Call

```
public interface Callback<T> {  
    /**  
     * Invoked for a received HTTP response.  
     * <p>  
     * Note: An HTTP response may still indicate an application-level failure such as a 404 or 500.  
     * Call {@link Response#isSuccessful()} to determine if the response indicates success.  
     */  
    void onResponse(Call<T> call, Response<T> response);  
  
    /**  
     * Invoked when a network exception occurred talking to the server or when an unexpected  
     * exception occurred creating the request or processing the response.  
     */  
    void onFailure(Call<T> call, Throwable t);  
}
```

# Sources & useful links

<http://square.github.io/retrofit/>

<http://www.vogella.com/tutorials/Retrofit/article.html>

<http://www.jsonschema2pojo.org/>