## Handling failed network request

**SimpleResponse**

We created the data class SimpleResponse, that will be responsible for encapsulate the success or failure of the network requests we will have to make.

This data class holds 3 parameters: status, data, exception.

Status is a sealed classed created that simulates an Enum. It has a Success and a Failure state. As it’s a sealed class, this class can be only used within the SimpleResponse file.

A companion object is then created, to treat both cases, success or failure.  
In short, companion objects are singleton objects whose properties and functions are tied to a class but not to the instance of that class — basically like the “static” keyword in Java but with a twist.

When passing a successful Response of type T, the function returns a SimpleResponse where the status is Success, the data(which is the response) and an exception as null.

However, if the passing response was failed, then it returns a SimpleResponse where the status is Failure, data is null and the exception receives the response’s exception.

**ApiClient**

In this class, an inline function safeApiCall was created which accepts as parameter a function that points to another one that will return a Response of type T, and this function will return a SimpleResponse of that same type.

This function will return the result of a Try-Catch block.

In the Try, the api request is invoked and in case it doesn’t fail at any point, the successful response is passed as a parameter to the SimpleResponse.success function, and it is used as return for this safeApiCall.

However, if the api request invoked fails at any point, the exception is handled and passed as parameter to the SimpleResponse.failure function. And consequently, returned for the safeApiCall.

private inline fun <T> safeApiCall(apiCall: () -> Response<T>) : SimpleResponse<T> {  
 return try {  
 SimpleResponse.success(apiCall.invoke())  
 }catch (e: Exception){  
 SimpleResponse.failure(e)  
 }  
}

Now we updated the getMovieById class to call the api request inside the safeApiCall. Therefore, it now returns a SimpleResponse rather than a Retrofit Response.

suspend fun getMovieById(movie\_ID: Int): SimpleResponse<GetMovieByIdResponse>{  
 return safeApiCall **{** movieDbService.getMovieById(movie\_ID) **}**}

**SharedRepository**

When the repository receives the apiCliente response, it can now check if it was failed, if it was not successful or if it was successful. There’s a difference between failed and not successful:

Failed - means that the api request failed completely. For example, there’s was no internet connection available at the moment of the api.

Not successful – the api request was completed, but something was wrong in the server side. Even though the request was completed, it didn’t return what we expected.

For both situations described above, we are just returning null. If it was successful, then we return the request body, which will be used by the viewmodel.