# Empirical Study on REST APIs Usage in Android Mobile Applications

Mohamed A. Oumaziz[1(✉)], Abdelkarim Belkhir[2], Tristan Vacher[2], Eric Beaudry[2], Xavier Blanc[1], Jean-Rémy Falleri[1], and Naouel Moha[2]

[1] Univ. Bordeaux - LaBRI - UMR CNRS 5800, Talence, France
{moumaziz,xblanc,falleri}@labri.fr
[2] LATECE, Département d'informatique, Université du Québec à Montréal, Montreal, Canada
{belkhir.abdelkarim,vacher.tristan}@courrier.uqam.ca,
{beaudry.eric,moha.naouel}@uqam.ca

**Abstract.** A large set of mobile applications (apps) heavily rely on services accessible through the Web via REST APIs. However, the way mobile apps use services in practice has never been studied. In this paper, we perform an empirical study in the Android ecosystem in which we analyze 500 popular apps and 15 popular services. We also conducted an online survey to identify best practices for Android developers. Our results show that they generally favor invoking services by using official service libraries instead of invoking services with a generic HTTP client. We also present which good practices service libraries should implement.

**Keywords:** Empirical study · Mobile applications · REST API · Rest services · Android

## 1 Introduction

Following the REST principles [6], server side applications are nowadays composed of several stateless independent micro-services [11]. They therefore make client side applications consuming more and more REST services [5]. Such evolution brings new challenges especially for the design of Android applications that now have to handle lots of calls to REST services.

However, little is known on how Android apps use REST services in practice. Such knowledge is of high importance for the service providers since it would help them provide facilities to Android developers and hence improve the usability of their REST services. For instance, do the developers prefer to handle JSON documents or Java objects? Do they want dedicated service libraries or do they want to perform the calls by using a HTTP client library?

In this article, we provide answers to these questions by performing an empirical study in the famous Android ecosystem. Our study focuses on two research questions. Our first research question: ***"As service users, how***


M. Maximilien et al. (Eds.): ICSOC 2017, LNCS 10601, pp. 614–622, 2017.
https://doi.org/10.1007/978-3-319-69035-3_45

***Android developers access popular REST services/APIs in their applications?"***, aims at identifying the developers' habits for accessing REST services. Our study shows that Android developers prefer to use a dedicated service library developed by the service provider if it exists.

Our second question: ***"As service providers, how to design client helper libraries to be popular among mobile applications?"***, aims at identifying which features of service libraries are considered important by developers. For instance, our study shows that the essential features for developers are the existence of a complete documentation, the library's vocabulary consistency with the service's one, the use of raw JSON to exchange data, the handling of authentication, and the ability to fine-tune the HTTP requests issued by the library.

This paper is structured as follows. Sections 2 and 3 respectively describe the study setup and results for the research questions RQ1 and RQ2. Section 4 presents the related work. Section 5 concludes and presents future works.

## 2 RQ1: *As service users, how Android developers access popular REST services/APIs in their applications?*

In this section, we investigate our first research question. We noticed that there are two main ways to access services from Android apps: directly, by using an HTTP client, or by using a library developed by the provider (official) or by its users (third-party). To assess which method is the most popular, we analyze how 15 popular services are used in a corpus of 500 popular apps. Section 2.1 explains how we build the corpus of services and apps. Section 2.2 explains how apps invoke services. Results and observations are then presented in Sect. 2.3.

### 2.1 Corpus

Our corpus consists of two sets: a set of popular apps and a set of popular services that are called by the popular apps. Additionally, we also gather the list of libraries that allow Android apps to interact with the services of our corpus.

We started by gathering a set of popular apps. To that extent, we crawled the top 500 most popular apps provided by the Google Play store[1]. We then downloaded the application packages of each app using the AndroZoo dataset maintained by our colleagues from the University of Luxembourg [1]. During this step, we were only able to download 487 app files.

To build the set of popular services, we analyzed the 487 APK files to identify which popular services are called. We then used the AndroGuard tool[2] to extract all the strings contained in the DEX bytecode files of each of our apps. From these strings, we extracted the URLs (i.e. starting with `http[s]://`). We then ranked these URLs by their number of occurrences and filtered out the ones

[1] https://play.google.com/store/apps/collection/topselling_free.
[2] https://github.com/androguard/androguard.

that do not correspond to a service (e.g. manual URL browsing). Finally, we manually selected 15 services among the 50 most popular ones.

To identify all the libraries for each service in our corpus, we use Google search with the following query "[service name] android library"; where "[service name]" corresponds to the service's root URL. Then, we manually look at the results to assess whether it describes an official library or a third-party one.

### 2.2 Experimental Setup

To answer our first research question, we check how apps call services, and in particular if they directly use services by making HTTP requests or if they use a dedicated library. To that extent, we first identify the services used by each app in our corpus. Secondly, we analyze if each app uses a library or not to access the services. Using this data, for each service, we classify the apps into three categories: apps using the service without library, apps using the service with an official library, and apps using a third-party library.

To find out which apps are using a given service we followed these steps. First we manually read the documentation of each service to find their API URL. Secondly, we manually browsed the code of all libraries to find out the list of all the Java packages they contain. Finally, we used the AndroGuard tool again to extract all the strings contained in all apps from our corpus. When we were able to find a service's API URL, we considered that the application used the service. In this case, we also looked for the Java package names of this service's libraries in the strings of the app. When we were able to find a package name in the string list, we assumed that the app is using its corresponding library.

Finally, to analyze how developers access services in practice, we perform the following process. For each service provider, we compute the set of all Android apps from our corpus that use it. Then, we partition this set into three subsets: the set of apps that use the official library, the set of apps that use a third-party library or both the official library and a third-party library, and the set of apps that do not use any library. To discuss the favourite way of developers to access the service, we then compare the size of these subsets, normalized by the size of all apps that use the services. Results are discussed in Sect. 2.3.

### 2.3 Results

In our results (accessible on our website[3]), we notice that only 5 out of 15 services are accessed with a HTTP client rather than a library. Moreover, 2 out of the 5 services provide no official library (Instagram and OpenStreetMap). Therefore, libraries are favoured to access services. Additionally, for the 10 services where a library is preferred, it is always the official library that is preferred, even if there are only 3 cases where no third-party library is available. In conclusion, **official libraries are the favourite way of developers to access services.**

[3] http://se.labri.fr/a/ICSOC17-oumaziz/.

However, although there are many HTTP clients, developers still prefer standard ones that are embedded in the Android Framework. The top four being in order: *HttpUrlConnection*, *HttpsUrlConnection*, *DefaultHttpClient* and *AndroidHttpClient*. We also notice that developers tend to use more than just one HTTP client, this can be related to the features that each client offers depending on developer's needs. For instance, *HttpsUrlConnection* is able to handle HTTPS requests while *HttpUrlConnection* only handles HTTP requests.

### 2.4 Threats to Validity

We discuss here the threats to validity of our study. The techniques used to detect client libraries and API URLs are not infallible. For instance, if an app is obfuscated, our techniques probably fail to identify URLs and used libraries. Also, there is the construction of URLs by string concatenations. Since we made a static analysis, we cannot catch all possible strings that could be built at runtime. Finally, we had to manually look at all available libraries for each service in our dataset. We may have missed few of them. Our corpus only contains 15 services and about 500 apps. Therefore, our results might not be generalizable to all Android apps. We attempt to provide all the necessary details to replicate our study and analysis, Scripts and datasets are also available online.

## 3 RQ2: *As service providers, how to design client helper libraries to be popular among mobile applications?*

To answer the second research question, we first studied the steps that apps follow to call a service, and the different kinds of libraries used under the hood. Then, from this process we identified the good and bad practices that should be followed when designing a service library. In the third step, we conducted an online survey to validate these good and bad practices by experts. As a final step, we analyzed official service libraries provided by popular services to verify if the latter are conform to these practices. We now detail each step.

***Step 1. Process to consume a service.*** We study the general process followed by any app to call a service. The process is divided in two sub-processes: Authentication which is optional (where the client asks for access right), and service consumption (where the client interacts with the service). During this process the app uses different libraries for: parsing, OAuth (to ask for permission), and HTTP Clients (to deal with the HTTP protocol).

***Step 2. List of good/bad practices when developing a service library.*** We identify here the good/bad practices that must be followed when designing a service library.

***Step 3. Online survey to validate the good and bad practices.*** The goal of the survey is to confirm the best practices that must be followed by service providers in their libraries to ease consumption by developers. The survey is

available online[4]. Based on the good and bad practices identified in *Step 2*, we build a survey on Google Forms and emailed it to 2000 Android developers randomly selected from the top 500 Android apps developers for each Google Play's category. We also submitted the survey as a Reddit Thread on the very active subreddit *Androiddev*, and advertised the survey through social networks. 51 Android developers responded to our survey and 83% of them are familiar with Android development. The survey and its results are available on our website.

***Step 4. Analysis of the official REST libraries.*** Finally, we manually analyzed 11 libraries and 14 services from our corpus. We did not analyze the OpenStreetMap and Instagram services because no libraries are available for these services, and the Google API Client library groups GoogleMaps, GoogleSignIn and YouTube services. All have been analyzed by three experienced Android developers to verify their conformance with the practices identified and validated in the two previous steps. We performed this analysis using their documentation, source code and provided examples.

### 3.1 Results of Research Question 2

For each identified good/bad practice, we first give a description as follow:

① *JSON vs. XML.* Always choose JSON over XML when both are proposed by the API provider.

② *Typed Response vs. Non-typed Response.* The response returned from the library for a given query should be a Java Object. In contrast, a *Non-typed Response* is a response returned as a JSON or XML format.

③ *Encapsulated HTTP Queries vs. Non-encapsulated HTTP Queries.* The HTTP query should be encapsulated in a method proposed by the interface of your library. A *Non-encapsulated HTTP Query* has to be manually built by the developer with all the needed parameters.

④ *Full vs. Non-exhaustive API support.* The Service Library should cover all the services proposed by the REST API.

⑤ *Consistent vs. Inconsistent vocabulary with documentation.* The vocabulary used in the code when naming classes, methods and attributes should correspond to the one used in the documentation of the REST API.

⑥ *Documented vs. Non-documented Library.* The library should be well documented, the user should be able to understand how to access the REST API endpoints preferably with code samples.

⑦ *Allowing Authentication vs. Third-party Authentication.* When an authentication is required to consume the offered services by the REST API. It is preferable that your Service Library allows authentication.

[4] http://bit.ly/clientpractices.

⑧ *Android Specific Functionalities vs. Only General Functionalities.* A good practice is to provide some Android specific functionalities such as widgets, views and fragments instead of providing only general functionalities.

We then chose to further discuss only 5 out of our 8 identified good/bad practices. The survey results are highlighted in bold.

① From our corpus, it seems that APIs favour the JSON format over others. Every library allows to return at least a response in the JSON format and provides sometimes other formats (XML, CSV, etc.). Although JSON is the most popular, it is not by default for all libraries. **In our survey, 92.2% of the developers stated that JSON was preferred.** This could be due to the fact that the JSON format is easier to handle, while also faster to load and to parse compared to XML files [2].

② Over the 11 libraries we studied, 6 return a domain-specific object representing an entity of the API (e.g. a File in the DropBox Library). So users don't need to parse the response. However, some libraries return an object containing data. For example, Facebook returns a GraphResponse object that contains the response, which is either a JSONObject, a JSONArray or a Java String. **In contrast, in our survey more than 70.6% of developers prefer to have responses as Java Strings.**

③ Almost all (10 out of 11) libraries except LinkedIn encapsulate HTTP queries. Users do not have to build their own requests, they can use predefined methods. However, libraries such as Facebook allow to build custom requests while providing encapsulated queries. **In the survey, 37.3% of developers think that modifying encapuslated queries is mandatory, and 47.1% of them think that it is appreciated. Therefore, although the majority of libraries encapsulate queries, developers still prefer to have access and control the queries.**

⑦ All analyzed services require authentication to be used. Authentication is a means to secure which data are reachable to someone, but also to control the request flow for avoiding overloading servers. All libraries implemented the entire service authentication protocol, namely OAuth2. **In the survey, developers confirm the necessity to implement the whole service authentication protocol with 58.8% who appreciate it and 29.4% who request it to be mandatory.**

⑧ Almost half of the libraries (5 out of 11) provide at least one Android specific functionality such as Widgets, Activities, or Views. Providing such functionalities can help developers focus on their own apps instead of trying to integrate logic from a third-party environment. **However, 86.3% of developers consider that providing such functionalities is not important.**

### 3.2 Threats to Validity

We discuss here the threats to validity of this section. The terminology used in the survey might have been misunderstood by the responders. However we wrote

definitions and examples to mitigate these threats. Our survey was answered by only 51 Android developers. Therefore, our findings might not be generalizable.

## 4 Related Work

In the following, we discuss some relevant research done on assessing bad and good practices in REST APIs as well as research on libraries identification.

***Bad and good REST practices.*** In [12–14], we evaluated the design of several REST APIs based on good and bad REST practices, also called REST patterns and antipatterns. We proposed automatic approaches to detect them. However, we evaluated APIs without considering any interaction with clients, and in particular mobile clients, as we do here. Other works proposed similar (anti-)patterns detection approaches in service applications, but implementing other techniques such as bi-level optimisation problems [17] or ontologies [4].

In [15], Rodriguez et al. evaluated the conformance of design best practices in REST APIs from the perspective of mobile apps. They analyzed these practices on a large dataset of HTTP requests collected from a Mobile Internet traffic. This work is the first that has studied the traffic of HTTP requests from the mobile perspective. However, the best practices analyzed are rather common to any kinds of REST APIs, and they focused on HTTP requests.

In contrast, in this paper, we consider practices that may apply on mobile apps. We take also into account the interaction between clients and REST APIs by analyzing all the process from the authentication to the service consumption, and thus while considering all kinds of message exchanges (requests, responses). We study also how REST APIs are implemented and documented.

***Libraries identification.*** There are several works that have been done for identifying advertisement libraries in Android apps. Book et al. [3] and Grace et al. [7] used a whitelists based method for identification. There are also tools such as AdDetect [10] and PEDAL [9] that applied machine learning techniques (SVM classification) to identify advertising libraries even if apps are obfuscated.

Teyton et al. [16] applied static analysis on the source code on a group of libraries to automatically extract Java package names. They identified 1185 different libraries which they then used to automatically identify Java libraries dependencies. Wang et al. [18] proposed a novel clustering-based technique to automatically identify Android third-party libraries. Their technique identified more than 600 different Android libraries in a corpus of 100,000 apps. Li et al. [8] proposed a novel approach for identifying third-party libraries from Android apps. Rather than using code similarity, they used code dependencies.

In this paper, we used the Java package names as a way to identify libraries. However, we had to identify service libraries, to do so, we used API URLs to determine if an app was using a service and then we applied this library identification technique to look if it was through a service library.

## 5 Conclusion and Future Work

While nowadays Android apps rely more than ever on REST services, no study has been performed on how Android apps invoke services. We alleviated this situation by performing an empirical study of 15 popular web services on a dataset of almost 500 popular Android apps. We show that developers prefer to use official libraries. We also show that developers prefer to use HTTP clients rather than libraries and prefer default clients provided in the Android Framework.

Second, we propose a list of good/bad practices, identified through an analysis of the practices of popular services and an online survey involving 51 developers. We show that the important features for libraries are: the use of raw JSON, authentication handling and the possibility to fine-tune HTTP requests.

As a future work, we plan to extend our practices' list and to extend the size of our dataset of services and apps in order to have more generalizable results.

**Acknowledgement.** The authors thank the Android developers for answering the survey. This study is supported by NSERC and FRQNT, Canada and Quebec research grants.

## References

1. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: AndroZoo: collecting millions of android apps for the research community. In: 13th MSR, pp. 468–471 (2016)
2. Betts, T.: Mobile performance testing - JSON vs XML. Blog. https://www.infragistics.com/community/blogs/torrey-betts/archive/2016/04/19/mobile-performance-testing-json-vs-xml.aspx. Accessed 20 June 2017
3. Book, T., Pridgen, A., Wallach, D.S.: Longitudinal analysis of android ad library permissions. arXiv preprint arXiv:1303.0857 (2013)
4. Brabra, H., Mtibaa, A., Sliman, L., Gaaloul, W., Benatallah, B., Gargouri, F.: Detecting cloud (Anti)Patterns: OCCI perspective. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 202–218. Springer, Cham (2016). doi:10.1007/978-3-319-46295-0_13
5. Danielsen, P.J., Jeffrey, A.: Validation and interactivity of web API documentation. In: 20th ICWS, pp. 523–530 (2013)
6. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
7. Grace, M.C., Zhou, W., Jiang, X., Sadeghi, A.R.: Unsafe exposure analysis of mobile in-app. advertisements. In: 5th ACM WiSec, pp. 101–112. ACM (2012)
8. Li, M., Wang, W., Wang, P., Wang, S., Wu, D., Liu, J., Xue, R., Huo, W.: LibD: scalable and precise third-party library detection in android markets. In: 39th ICSE, pp. 335–346. IEEE Press (2017)
9. Liu, B., Liu, B., Jin, H., Govindan, R.: Efficient privilege de-escalation for AD libraries in mobile apps. In: 13th MobiSys, pp. 89–103. ACM (2015)
10. Narayanan, A., Chen, L., Chan, C.K.: Addetect: automated detection of android ad libraries using semantic analysis. In: IEEE ISSNIP 2014, pp. 1–6. IEEE (2014)
11. Newman, S.: Building Microservices - Designing Fine-grained Systems, 1st edn. O'Reilly, New York (2015)

12. Palma, F., Dubois, J., Moha, N., Guéhéneuc, Y.-G.: Detection of REST patterns and antipatterns: a heuristics-based approach. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 230–244. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45391-9_16
13. Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y.-G., Tremblay, G.: Are RESTful APIs well-designed? detection of their linguistic (Anti)Patterns. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 171–187. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48616-0_11
14. Petrillo, F., Merle, P., Moha, N., Guéhéneuc, Y.-G.: Are REST APIs for cloud computing well-designed? an exploratory study. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 157–170. Springer, Cham (2016). doi:10.1007/978-3-319-46295-0_10
15. Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J.C., Canali, L., Percannella, G.: REST APIs: a large-scale analysis of compliance with principles and best practices. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 21–39. Springer, Cham (2016). doi:10.1007/978-3-319-38791-8_2
16. Teyton, C., Falleri, J.R., Palyart, M., Blanc, X.: A study of library migrations in java. J. Softw. Evol. Process **26**(11), 1030–1052 (2014)
17. Wang, H., Kessentini, M., Ouni, A.: Bi-level identification of web service defects. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 352–368. Springer, Cham (2016). doi:10.1007/978-3-319-46295-0_22
18. Wang, H., Guo, Y., Ma, Z., Chen, X.: Wukong: a scalable and accurate two-phase approach to android app. clone detection. In: ISSTA 2015, pp. 71–82. ACM (2015)