MobiRest: Mobile applications extension to support REST web services

Daniel Tiago de Castro, 72863, MEEC
Instituto Superior Técnico, Universidade de Lisboa

***Abstract*— In the last few years, the use of smartphones and mobile applications have become very popular. Nowadays, there are already a wide range of mobile application development technologies. Moreover, the stack of technologies is evolving each year in a fast rhythm, with new features and capabilities. However, there's a lack of work on the recent hybrid technologies such as Ionic and Cordova on support for web services. Additionally, in the typical mobile application architecture, there is communication with an external server, but nearby devices may need to communicate between each other, without the intermediary of an external server. Thus, combining these gaps with the pressure that developers have regarding development of multiple-platform mobile applications, it arouses the opportunity to develop a system which provides mobile RESTful web services in hybrid development frameworks. This work proposes a mobile system, called *"MobiRest"*, that allows developers to use it and integrate it into their projects, thus making them able to provide RESTful web services on their mobile apps.**

***Index Terms*— Web Services, Mobile Development, Hybrid Framework, REST**

I. Introduction

MOBILE DEVELOPMENT is one of the most growing industries, which gives rise to largest economic sectors in the world. In the last few years, the use of smartphones and mobile applications have become very popular. Several web technology innovations have been allowed software designers and engineers to quickly develop responsive mobile-friendly applications to fulfill people's needs [1]. Furthermore, mobile users have become more and more exigent over the years, they want fast and at the same time easy and interactive mobile experience. Over the last years, statistics point that the number of downloads of mobile applications have been increasing substantially. There was approximately 150 billion in 2016, around 197 billion in 2017, and it is predicted that there will be 353 billion of downloads in 2021 [2].

A. Motivation

In the typical mobile application architecture, there is communication with an external server. In this way, it is necessary to have access to the network so that this communication is possible. However, nearby devices may need to communicate between each other, without the intermediary of an external server. In this case, it would be appropriate to make use of the existing hotspot technologies, thus creating a local network with multiple users / devices connected to each other. Web services are one of the technologies that revolutionized the possibilities and interoperability of mobile applications. Essentially there are two types of web services: SOAP and REST. There has been a rapid growth of web services development in parallel with a large use of web applications and progression of wireless communications. These advances take effect on real life daily applications [3]. Consuming web services on mobile devices is now becoming very common but the combination of hosting web services on these devices can make a wide range of new functionalities and features [4]. In this context, the idea is that instead of using a server in the cloud, the mobile application itself is a server capable of responding to REST requests from multiple users who are or want to connect to it.

II. State of the art

A. Mobile Social Networks

Social networks are the new reality of people. On the digital era, people have the need to communicate each other, share their thoughts, impressions, search for recommendations, and be part of a global community where the distance have become smaller due to the easiness of communication all over the planet.

Modern mobile applications are making use and reinventing the way of using geospatial technologies. At the beginning, mobile phones had only the purpose to provide voice and text communication, but nowadays it is just one slice of the "cake". Other important factors are the web browser and the GPS services [5]. This allows developers to create mobile applications that include LBS services (referred as "Location-Based Services"). Thus, the new network paradigm is to build social networks over mobile applications in the events or locations. This paradigm allows people to communicate and share experiences with minimum required infrastructure and without needing access to Internet [6].

Derived from the LBS, emerged the mobile social network in proximity (MSNP). It represents the decentralized proximal location-based social network that is active in a wireless social network [7]. The essential difference between LBS and MSNP is the physical geographical coverage. The first can refer to any place in the world. As the second emphasizes social

interaction on the proximal area, on the range of wireless network communication technology. Building mobile applications in which there is a location context is very relevant because can add value and make accessible a more modern methodology of interacting.

B. Edge and Fog Computing

Technology has already advanced tremendously that is possible for people to have videoconferences, or to share experiences with the highest resolution such 4k or at least full HD. Regardless of the source, collecting ever-increasing amounts of data pushes up the network bandwidth to its limits [8]. Furthermore, emerges the following question: Is it absolutely necessary to push all the information data to the cloud? In response to this question there were presented two solution models involving the decentralization of the processing layer: fog and edge computing. The term "Fog Computing" was created by Cisco [9] and is defined as "a model to complement the cloud for decentralizing the concentration of computing resources in data centers towards users for improving the quality of service and their experience" [10]. "Edge Computing" corresponds to "a method of optimizing cloud computing systems by performing data processing at the edge of the network, near the source of the data, reducing communications bandwidth. These themes are relevant since this work provides a solution for the decentralized systems.

C. Mobile Applications Development

There are essentially three types of approaches on building mobile applications: fully native apps, mobile web apps and cross-platform apps. Fully native apps, for example in Java for Android or Objective C for iOS, can access all the platform specific features, including access to the hardware of the device, for example the camera or accelerometer. Web mobile apps in opposition are built using standard web technologies, so they run in every mobile device with a web browser, although they are very superficial because can only access features supported by HTML5. Cross-platform apps are the combination of last two approaches: they are developed using the web standard technologies, which means they support multiple platform. Moreover, with the pre-installation of specific plugins it accesses all the device features that a native app does [11]. Furthermore, cross-platform apps are divided in two types: hybrid cross-platform and native cross-platform. Both approaches offer mostly the same, however the first one renders using HTML and CSS, while the second one renders using native components [34].

In a survey [12] made to developers statistics point that hybrid mobile apps are becoming more popular, thus this work will focus on this type of approach.

D. Web Services Support in Mobile Development

Web services, officially defined as "a software system designed to support interoperable machine-to-machine interaction over a network" [13], are the piece that enables the interaction between the server and the user devices. There are essentially two categories of web services: SOAP and REST. The first one, stands for *Simple Object Access Control* which is based on oriented object approach, defining a standard protocol for XML format exchange on communication. Moreover, REST was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation [14]. It stands for *Representational State Transfer* and, differently of SOAP, is based on a resource-oriented approach, following an architectural style that obeys to several constraints.

Some researchers defend that the idea of *"mobile web services"* is designed for the consume of an API on client-side applications for mobile devices [7], [15], [16]. Although, more recent works such as Gehlen and Pham [17], Pawar et al. [18] suggest the use of mobile web services as "mobile-device-hosted web services".

Nowadays, there are already many open-source tools that easily can make a smartphone turned into a web server [19], [20], [21], [22], necessary for provisioning web services. Berger et al. [23] developed a web application capable of provisioning SOAP web services on mobile devices within a local network using access points. Afterwards, Srirama et al. [24] developed a lightweight mobile hosting SOAP-based system with a handler that points to the appropriate web service.

In other hand, an approach using RESTful web services is more appropriate for the constrained conditions of mobile devices because they provide lightweight features. Besides that, REST architecture is more suitable for the decentralized environment. Paniagua [25] implemented on Android a mobile host provider for REST services. Accordingly, REST is more efficient than SOAP web services.

With the emerging of new technologies, namely, hybrid web-based technologies such as Ionic and Cordova, it is important to update the previous works to the most recent technologies and test the usage of mobile web services on this context.

III. Proposed Solution

Nowadays, there are already a wide range of mobile application development technologies. The stack of technologies is evolving each year in a fast rhythm, with new features and capabilities. Moreover, mobile devices are also suffering a big evolution. They have increased their processing and memory capabilities, becoming each year more powerful in both hardware and software.

As mentioned before, there has been some work done on decentralized systems that host mobile web services in mobile devices. However, there's a lack of work on the recent hybrid technologies such as Ionic and Cordova. Combining this gap with the pressure that developers have regarding development of multiple-platform mobile applications, it arouses the opportunity to develop a system which provides mobile web services in hybrid development frameworks.

This article hereby proposes a mobile system, called *"MobiRest"*, which will allow other developers to use it and integrate it into their projects, to make them able to provide RESTful web services inside their mobile apps.

A. *Applications Architecture*

MobiRest system, is built with the purpose of communication and interaction with other devices over the same network. Thus, several users of this network can access provided endpoints. In Figure 1 it is represented the architecture of applications with *MobiRes*t embedded.

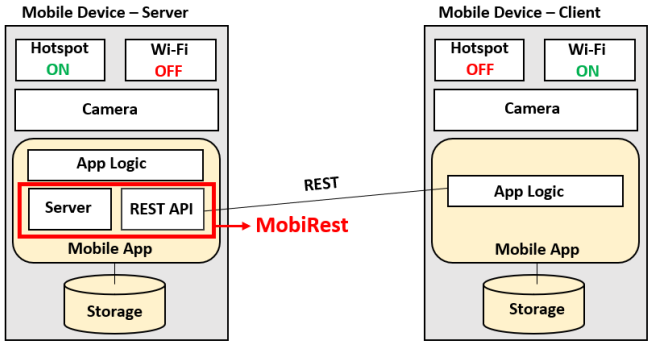

Fig. 1 - Physical architecture of the system and mobile application.

It is distinguished for the same device two different states in which applications will operate: as a server, or as client. In the first case the system will have activated the embedded server together with the hotspot functionality. Moreover, all the REST requests made within the same local network will be addressed to this server. In the second case the server is disabled, and the device connected by Wi-Fi to the created hotspot.

The architecture of the system corresponds to a Client-Server structure among the period of time that users pretend to share media content. This is, although the system respects a client-server architecture, the point is that the server is not centralized and fixed in the same device. In different times, the same user can be in on the different states of *MobiRest.* Hence, the system altogether, operates as a decentralized system since it doesn't forward computation processes to the Cloud, and in different temporal slices, it can act as a client or as a server.

B. *Requirements*

To be able to achieve the proposed solution, some requirements are needed to be met.

1. *MobiRest* will be used in hybrid mobile applications. It will be easily integrable with existent applications developed with Cordova (or Apache Cordova family technologies) and/or Ionic.
2. *MobiRest* will be targeted at Android Operating System;
3. *MobiRest* will support the provisioning of RESTful web services on hybrid mobile applications.
4. *MobiRest* will use Wi-Fi and hotspot technologies;
5. *MobiRest* will work in offline mode. In other words, without "3G connection".
6. *MobiRest* will be easily programmable.
7. *MobiRest* will have a dedicated file in JavaScript/TypeScript where developers will registry an API for their systems' endpoints by using the offered API functions of *MobiRest*. Thus, the system will be adaptable and easy changeable
8. The server, which will be based on the *"Cordova-Httpd"* plugin [26] will require several features:
   - Will have support CRUD operations;
   - Will allow the distinction between requests for serving a web page or a REST request. It will successfully serve both types of requests;
9. It should be efficient in terms of performance and battery usage.
10. It should be a lightweight system.

C. *MobiRest Architecture*

In Figure 2, there are represented the architectural blocks of *MobiRest.*

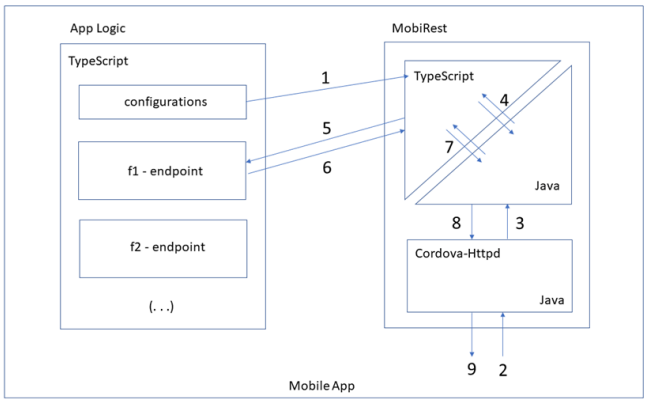

Fig. 2- Architecture and execution Flow of MobiRest on a mobile app.

The "App Logic" block contains all the configurations necessary to integrate MobiRest system on a mobile app. Besides that, it contains the library of endpoint functions that are pretended to execute according to each web service.

The main structure of *MobiRest* contains a Java webserver - *Cordova-Httpd plugin*. Then, it divides two structures: the Java block that intersects and parses the HTTP messages; and the TypeScript block that communicates with the API endpoints and serves the web services. All these blocks together make part of a mobile application.

D. *Execution Flow*

The execution flow is made according to the Figure 2. Each step is numerated and described following:

1. *MobiRest* receives the information needed for the integration of the web services developed. Now, MobiRest knows the RESTful web services to serve.
2. After informing the system which are the endpoints to serve, *MobiRest* is able to respond to requests.
3. The received HTTP requests are intercepted and parsed. If the requests are web requests, they are served by the web server on the Java side. If they correspond to REST requests, the next step is responsible for handling them.
4. The received HTTP requests in case of being RESTful, are transposed to the TypeScript layer through complex

mechanisms of asynchronous programming, waiting then for the REST response.
5. Within the TypeScript layer of *MobiRest*, the requests are analysed and, according to their URIs and HTTP methods, the correspondent endpoint function received on step 1 is called and operates the necessary logic of the application to serve the web service.
6. After processing possible received data from the request (POST method) and executing the web service, *MobiRest* system builds a response with the results achieved on step 5.
7. The HTTP response built, is forwarded to the Java layer. Once again, this bridge is made by asynchronous complex mechanisms that will be discussed with more detail further.
8. The response message is transferred to the web server, responsible for the communication with the clients.
9. Finally, the HTTP message of the response is sent to the respective clients.

IV. Implementation

Fundamentally, this work was implemented in Cordova and Ionic frameworks, using Java for the backend, and Angular framework together with TypeScript for the mobile app logic, including the integration for the REST endpoints, as it seemed to be the most suitable choice on this project. Moreover, it was used the Cordova plugin already mentioned, cordova-httpd [26] because it contains an embedded web server for hybrid mobile applications. With this server, it is possible to serve files and web pages. However, it doesn't offer the option for hosting RESTful web services.

Thus, along this work, the major part of the implementation was made by programming over this server in order to provide the objectives proposed.

A. MobiRest Bridge Java - TypeScript

This subsection covers the deepened details of steps 4 and 7 of the Execution Flow and explains the mentioned concepts of asynchronous programming done along this work. This was the most challenging part on the implementation of *MobiRest.*

The technologies used, namely Cordova, imply the use of asynchronous programming, defined as "a form of parallel programming that allows a unit of work to run separately from the primary application thread" [27]. The moment when this code is executed is undefined, it may be immediate, or might happen later on, thus it is asynchronous. This paradigm although being complex and hard for debugging, is common in servers, because the response is not automatic, and depends on external communication factors such as latency and network traffic.

The bridge between the Java and the TypeScript layers of *MobiRest* are explained on the following description. After intercepting and parsing the HTTP requests and store its information in separate variables, a selection mechanism analyses if the request is a regular HTTP web request or if it corresponds to a web service. In the first case, the request is served on the file server component of the Java layer. In the second case, it is called a function that encapsulates the request's variables on a JSON and through a "callback" function, it is forwarded to the TypeScript layer and waits for a response. On the TypeScript side, the parameters of the request are compared with the endpoints received in step 1 of the Execution Flow, and the correspondent method of the web service is invoked, sending back the "callback" response (HTTP message).

B. Use of MobiRest

The deepened steps on how to use *MobiRest* system are described succinctly in the associated master thesis of this article [28].

V. Evaluation

A. Performance Tests

It was evaluated the time to process requests on *MobiRest* versus an external server.

1) Settings

All the tests were made using the tool *Postman* [29] on a laptop. This tool allows making collections of requests, and therefore the same request can be iterated the number of times users whish.

In Figure 3 a) and b) it is outlined the settings for the tests made in *MobiRest* and in the external server.

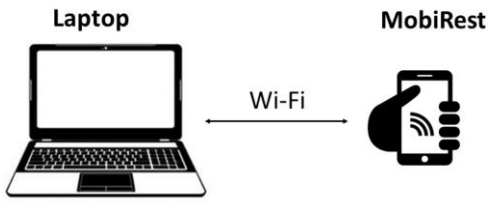

Fig. 3-a) - Settings for the MobiRest performance tests.

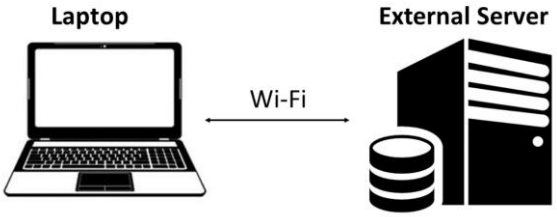

Fig. 3-b) Settings for the performance tests of the external server.

The same web services tested in *MobiRest* were hosted and tested in the external server Sigma03 server of Instituto Superior Técnico [30].

Furthermore, all the performance tests of *MobiRest* were made on a Samsung J6 (2016) smartphone, with Android 7.1.1 version and a Quad-Core 1.2 GHz processor.

2) Scenarios

Multiple scenarios were built and *MobiRest* was tested on different REST endpoints, as following:
external server on the tests made:

- 1 request of 10 MB;

- 10 requests of 1 MB;
- 100 requests of 100 KB;
- 1000 requests of 10 KB;
- 10 000 requests of 1 KB;
- 100 000 requests of 100 B;

In each scenario, servers sent or received 10 MB in total.

In the case of the external server, the tests were made on a different Wi-Fi network than the one where the server was running, while on the tests made on *MobiRest* the requests were made on the same local network of the server.

3) Results

The obtained results are presented in Figures 4 and 5.

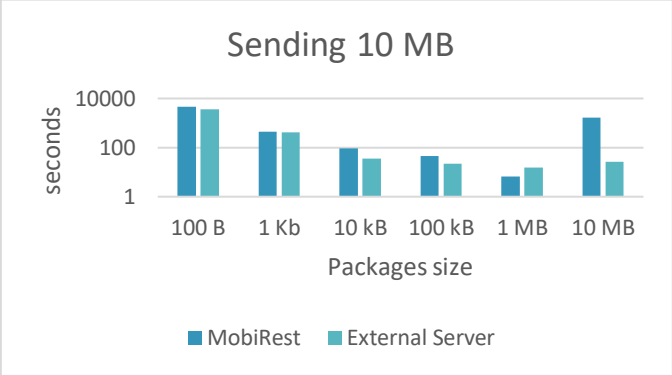

Fig. 4 - Temporal differences between MobiRest and an external server (on sending data). Logarithmic scale.

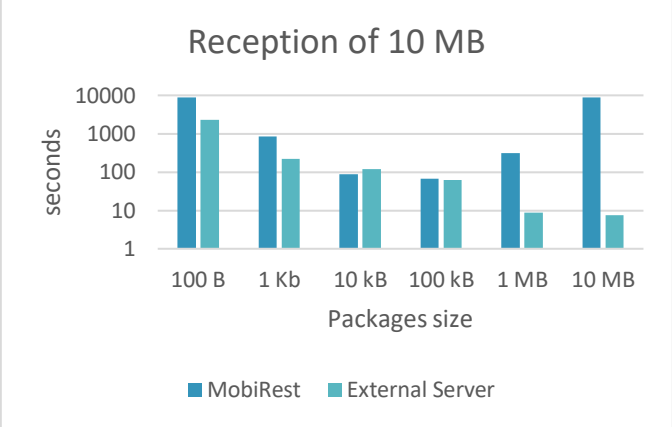

Fig. 5 - Temporal differences between MobiRest and an external server (receiving data). Logarithmic scale.

Analyzing the obtained results, it is possible to observe that on a range of medium size packages *MobiRest* had a good performance, almost equal comparing to the communications made with an external server. When were made 100 thousand of requests with small size packages, the performance is slightly lower, but still comparing with an external server, is reasonable. The system only had not reasonable performance when the packages are big (10 MB).

Besides that, when focusing on middle range of packages, *MobiRest*, surpassed the performance of the external server for packages of 1 MB (on GET requests), and on packages of 10 KB (on POST requests).

Thus, in general terms, it is possible to affirm that users will have good performance of *MobiRest* system, since on the medium range the system works fine.

B. Energy Consumption Tests

It was also evaluated the energy consumption of *MobiRest*.

1) Settings

In Figure 6 it is outlined the settings for the energy consumption of MobiRest.

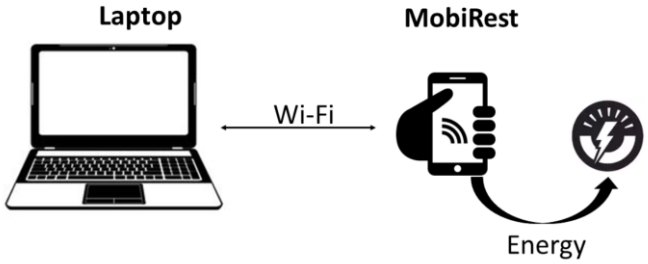

Fig. 6 - Settings regarding energy consumption of MobiRest.

Once again, it was used the *Postman* tool to generate requests to the MobiRest server. To measure the energy consumption, it was used a device that, by connecting it to the smartphone with MobiRest, could measure the consumed energy during the tests.

2) Scenarios

To evaluate quantitatively this aspect there were made tests by running a certain number of requests in different scenarios:

- On maximum light with *MobiRest* running and responding to 1500 requests of 100 KB each;
- Without light and with *MobiRest* running and responding to 1500 requests of 100 KB each;
- With the smartphone awake, with maximum light;
- With the smartphone in sleep state.

In all the different scenarios the procedures were the same: there was made a certain number of requests to *MobiRest* server, while measuring the time to respond to all the requests. Then, the smartphone running MobiRest was shut down, and charged until the level of battery it was at the beginning of the tests. During the charge period, the smartphone was connected to a device that measures the energy crossed (in mAh). This was how the amount of energy spent during the tests was measured. The first two tests took approximately 20 minutes each, therefore the last two tests occurred with the same amount of time, but without using *MobiRest*. The point was to observe the differences of energy consumption with and without running *MobiRest*.

3) Results

In Figure 7 there are presented the results of the tests.

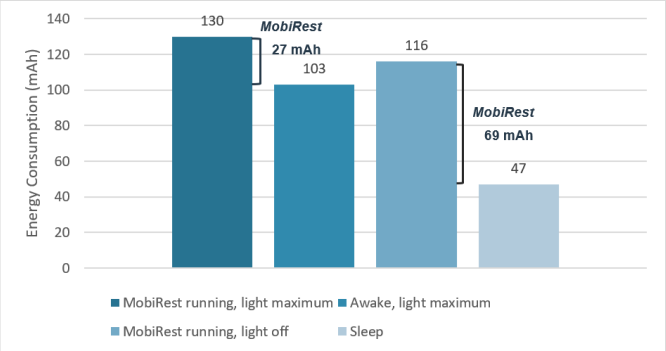

Fig. 7 - Results of the energy consumption tests.

By analyzing Figure 7 it is possible to observe that with the smartphone with light in maximum level and running *MobiRest* for 1500 requests, the consumption of energy was only 26% more comparing to the 103 mAh awake with the light in maximum level. When the light was off and *MobiRest* was responding to 1500 requests, the difference to the sleep state was a little bigger, but any other application generally will have the double of energy consumption comparing to the sleep state. Besides that, it is to note that on a local network, generally, on a regular use, an application doesn't have to respond to so many requests (150 MB in total) on a temporal space of 20 minutes. This means that, for example if the battery of a smartphone has the duration of a day on idle state, with *MobiRest* running, the smartphone can stand for 9 hours and half.

Furthermore, it was calculated the amount of energy that each request of 100 KB takes, in each case:

- With light on maximum it took 18 µAh;
- Without light it took 46 µAh.

Thus, in general terms, *MobiRest* met the requirements proposed regarding energy efficiency.

C. Demonstration App – "Travel Sync"

In addition, this work proposes to be made a demonstration example of application in Ionic and Cordova, that will be called *"Travel Sync"*, for sharing photographs in groups of people. It makes use of *MobiRest* system to allow sharing media content within a local network without the need of accessing the Internet. With this app, users can have access to the camera of their devices and be able to take photographs and share them with their friends. People stablish new "friendship" on this app by scanning (with the camera) a QR code generated by his friend. All the data is stored on databases provided by the application on the device that is running the server.
Thus, people can be, for example, be in a concert, where sometimes the 3G connection is weak, then connect each other and share photographs.

1) External Libraries

To access all the features proposed, there was the need to install the external libraries:

- Cordova Plugin Camera;
- Cordova Plugin SQLite (local internal databases);
- Cordova Plugin Barcode Scanner (QR codes).

2) REST endpoints

To fulfil the communication features above mentioned there were defined the REST endpoints of *MobiRest*, described on Table 1.

Table 1- REST endpoints implemented in Travel Sync.

HTTP Method	Endpoint	Description
GET	/api/photos	It reads from the database the data regarding the photos of the user's friends and sends them to the client.
POST	/api/photos	It stores on the database a photography that the client took.
POST	/api/friends/add	It stores all the information about a friend of the client.
GET	/api/friends	It returns all the users that connected to the same "master" user (server). This is all the users of the same social network

Thus, by using *MobiRest* system, after integrating the library with the endpoints referred above, and implementing the app logic, the application *Travel Sync* is capable of executing RESTful web services such as:

- Send a photography taken to the server of MobiRest, then store it on the device that has the server enabled;
- Receive photographs of friends, that are stored on the server's database and display them on a view;
- Adding new friends, by sending the client's information to the friend;
- Getting the name of all the users connected to his/her "master", and consequently, that are in the same network.

Besides web services, *Travel Sync* was implemented with other related functionalities, included on the app logic module referred on section III-C. These functionalities include:

- Register users and store the correspondent information such as username and password on the database of the application. This way this information is stored locally. Note that it is needed this information and a token generated on the process of adding friends;
- Add automatically to the photographs taken a marker with the name and description.

To keep up with this section, there it is presented an example of the user interfaces of *Travel Sync* app, in Figure 8.

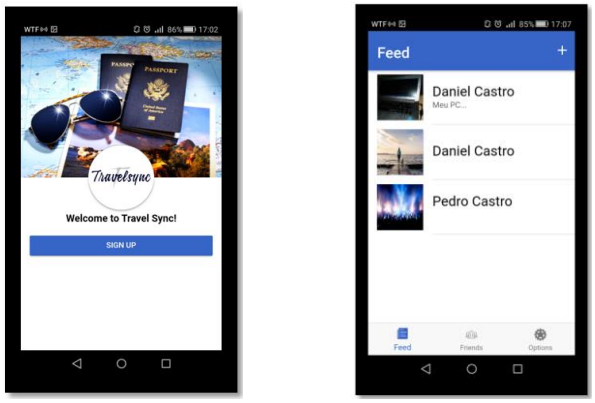

Fig. 8 - Examples of user interfaces of Travel Sync app.

D. Discussion

The developed mobile application, *Travel Sync*, makes use of the *MobiRest* system and is a real-life example that can benefit of such system. It also demonstrates that is possible to host and provide RESTful web services through smartphones with hybrid development. With *MobiRest*, *Travel Sync* application doesn't need to communicate with a central server in the Cloud, facing then vulnerabilities such as privacy and security issues. Moreover, decentralized systems, mentioned before by the term "computing at the Edge", are more secure than computing in the Cloud, by the fact that aren't so vulnerable. Moreover, in centralized systems, it is required a "3G connection" to make it possible the communication to the Cloud, so that users can access to the stored data, or to communicate with other users. Thus, using *MobiRest* system becomes advantageous since users only need to connect via Wi-Fi to the local server, which can be without Internet access – offline mode.

If *Travel Sync* hadn't used *MobiRest* system, location awareness should use GPS systems of mobile devices, which consume a big amount of energy. Besides that, the central server should store the location coordinates of all the users at different times and process it, discovering nearby users. Then, the users would have to send the photographs taken to the server in the Cloud, through 3G connection, which may be limited in certain occasions.

Furthermore, the results of the evaluation tests show that *MobiRest* system responds well in terms of performance and power consumption in local networks. Additionally, it can be also a powerful tool for mobile applications, namely Social Networks in Proximity that, in more hostile environments such as concerts and other crowded events, may origin 3G connection losses, and consequently failure of communication on the typical client-server applications through the Cloud. Thus, *MobiRest* system demonstrated being efficient and simplifies all this process of mobile social networking in *Travel Sync* app.

VI. Conclusion

This work started by introducing the theme and discussing its motivation. The main goal was to develop *MobiRest* system, that allows developers to provide RESTful web services on their own mobile applications, leveraging thus the hotspot and Wi-Fi technologies.

In order to fulfil this mission, in section II it was made a research regarding the existent mobile development frameworks and support for mobile web services. Also, it was explored the state of the art regarding mobile social networks, namely "in proximity". This research was important to know how other works solutioned this problem, and to detect the gaps missing. Other relevant subjects within the context of this work were explored such as Edge computing, explaining its relevance over this work, corresponding to a solution for decentralized systems.

In section III, it was presented the proposed solution and described the architecture of mobile applications with this system embedded, including an explanation of each module of the system. This section also specified all the requirements that *MobiRest* should met, as well its internal architecture and an execution flow explanation.

After, in section IV an evaluation and demonstration of the *MobiRest* system was made and the results were discussed. On the evaluation tests, *MobiRest* demonstrated having a general good performance, especially in middle sized packages of data. Furthermore, in terms of battery life it was observed that the consume of energy doesn't represent a huge constraint comparing with other types of usage on smartphones.

For future work, *MobiRest* can be extended to support other operating systems such as iOS, or Windows Phone, leveraging the benefits of hybrid applications in terms of multiple platforms' support. This extension should be similar to the one implemented along this work for the Android operating system. Moreover, this extension can take advantage of the existent server on the mentioned plugin, Cordova-Httpd [26], used over this work, and follow the steps deepened discussed on the section "*MobiRest* bridge Java - TypeScript", described in Chapter 3. Thus, developers can use *MobiRest* on their apps, for multiple platforms and target a bigger number of users.

Acknowledgment

The author of this article would like to thank Prof. João Silva for is constant supervision and support.

References

[1] F. Shahzad, "Modern and Responsive Mobile-enabled Web Applications," *Procedia Comput. Sci.*, vol. 110, pp. 410–415, 2017.

[2] "• Annual number of mobile app downloads worldwide 2021 | Statistic." [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed: 07-Mar-2018].

[3] F. AlShahwan and K. Moessner, "Providing SOAP web services and RESTful web services from mobile hosts," *5th Int. Conf. Internet Web Appl. Serv. ICIW 2010*, no. June, pp. 174–179, 2010.

[4] K. E. Mohamed and D. Wijesekera, "Performance analysis of web services on mobile devices," *Procedia Comput. Sci.*, vol. 10, pp. 744–751, 2012.

[5] S. Kumar, M. A. Qadeer, and A. Gupta, "Location

based services using android (LBSOID)," *2009 IEEE Int. Conf. Internet Multimed. Serv. Archit. Appl.*, no. February 2015, pp. 1–5, 2009.

[6] G. A. . Prasad, B. . Muthu Kumar, and S. . Dhayanandh, "A novel and efficient android based mobile ad-hoc social networks model," *Int. J. Appl. Eng. Res.*, vol. 10, no. 55, pp. 1149–1154, 2015.

[7] C. Chang, "Service-Oriented Mobile Social Network in Proximity Caulfield School of Information Technology Monash University," no. 190, 2013.

[8] "Edge Computing vs Fog Computing – Ridwan Shariffdeen – Medium." [Online]. Available: https://medium.com/@rshariffdeen/edge-computing-vs-fog-computing-5b23d6bb049d. [Accessed: 27-Mar-2018].

[9] "Edge computing vs. fog computing: Definitions and enterprise uses - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html. [Accessed: 28-Mar-2018].

[10] B. Varghese, N. Wang, D. S. Nikolopoulos, and R. Buyya, "Feasibility of Fog Computing," 2017.

[11] J. Caballero, E. Bodden, and E. Athanasopoulos, "Engineering secure software and systems: 8th International Symposium, ESSoS 2016 London, UK, April 6–8, 2016 proceedings," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9639, pp. 72–88, 2016.

[12] "Ionic Framework - 2017 Developer Survey." [Online]. Available: https://ionicframework.com/survey/2017#results. [Accessed: 11-May-2018].

[13] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley, 1998.

[14] "Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)." [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Accessed: 10-Apr-2018].

[15] W. Zahreddine and Q. H. Mahmoud, "An agent-based approach to composite mobile Web services," *Adv. Inf. Netw. Appl. 2005. AINA 2005. 19th Int. Conf.*, vol. 2, pp. 189–192 vol.2, 2005.

[16] T. Yoshikawa, K. Ohta, T. Nakagawa, and S. Kurakake, "Mobile Web service platform for robust, responsive distributed application," *Proc. - Int. Work. Database Expert Syst. Appl. DEXA*, vol. 2003–Janua, pp. 144–148, 2003.

[17] L. P. Guido Gehlen, "Mobile Web Services for Peer-to-Peer Applications," *Second IEEE Consum. Commun. Netw. Conf. 2005. CCNC. 2005*, pp. 427–433, 2004.

[18] P. Pawar, S. Srirama, B. J. Van Beijnum, and A. Van Halteren, "A comparative study of nomadic mobile service provisioning approaches," *NGMAST 2007 - 2007 Int. Conf. Next Gener. Mob. Appl. Serv. Technol. Proc.*, no. Ngmast, pp. 277–283, 2007.

[19] "CocoaHTTPServer. Available at https://github.com/robbiehanson/CocoaHTTPServer."

[20] "Mongoose." [Online]. Available: https://code.google.com/archive/p/mongoose/. [Accessed: 07-Apr-2018].

[21] "iJetty. Available at https://github.com/jetty-project/i-jetty."

[22] "kWS - Android Web Server – Apps no Google Play." [Online]. Available: https://play.google.com/store/apps/details?id=org.xeustechnologies.android.kws. [Accessed: 07-Apr-2018].

[23] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath, "Web services on mobile devices-implementation and experience," *Proc. DARPA Inf. Surviv. Conf. Expo. MCSA-03*, no. Wmcsa, pp. 100–109, 2003.

[24] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile Web Service Provisioning," *Adv. Int'l Conf. Telecommun. Int'l Conf. Internet Web Appl. Serv.*, pp. 19–25, 2006.

[25] C. Paniagua, "Discovery and push notification mechanisms for mobile cloud services," no. May, 2012.

[26] R. Xie, "cordova-httpd plugin." [Online]. Available: https://github.com/floatinghotpot/cordova-httpd.

[27] "Asynchronous Programming." [Online]. Available: https://stackify.com/when-to-use-asynchronous-programming/. [Accessed: 18-Apr-2018].

[28] D. Castro, "MobiRest: Mobile Applications Extension to Support REST Web Services," Instituto Superior Técnico, 2018.

[29] "Postman | API Development Environment." [Online]. Available: https://www.getpostman.com/. [Accessed: 07-May-2018].

[30] "Acesso ao cluster sigma | Suporte ao Utilizador • Direção de Serviços de Informática." [Online]. Available: https://suporte.dsi.tecnico.ulisboa.pt/manual-do-utilizador/acesso-ao-cluster-sigma. [Accessed: 07-May-2018].