MobiRest: Mobile Applications Extension to Support REST Web Services

Daniel Tiago de Castro

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. João Nuno De Oliveira e Silva

Examination Committee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo

Supervisor: Prof. João Nuno De Oliveira e Silva

Member of Committee: Prof. João Coelho Garcia

20th June 2018
To my parents, brother and girlfriend, for all the love and support.
Acknowledgements

It’s a great feeling to look back and see all I have learned and passed through my studies. It was a long road to get here, but at the end, I’m proud of the capabilities I earned. When I began this thesis I didn’t imagine how ambitious it would be.

First, I would like to thank my supervisor Prof. João Nuno Silva, for his flexibility in finding a subject for a thesis that goes along with my learning wishes and ambitions. In the last months, my way of thinking increased positively thanks to him, so I offer him my sincere gratitude for continuously pushing the limits and making me learn more each day.

To my dear family: my mother, Ana Tiago, my father, Arnaldo Castro, and my brother, Pedro Castro, that always supported me in everything I needed and always motivated me to continue the effort and to never give up on my goals, my eternal thanks.

A huge thank you to my friend Francisco Cunha, for the motivation and support on the hardest times of my thesis. Without him, it would have been much harder to accomplish my goals.

Also, I would like to extend my thanks to Felipe Teixeira, for the help he gave me at the beginning of this thesis, when everything was a new challenge.

Finally, to my girlfriend, Nidia, for all the love and motivation she gave me every day of my work, my honest gratitude.
Declaration

I hereby declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
Resumo

Nos últimos anos, o uso de smartphones e aplicações móveis tornou-se bastante popular. Atualmente, existe já uma ampla gama de tecnologias de desenvolvimento para aplicações móveis. Além disso, a gama de tecnologias está a evoluir a cada ano, num ritmo acelerado, com novos recursos e capacidades. No entanto, há falta de trabalho realizado nas recentes tecnologias híbridas, tais como Ionic e Cordova, no suporte para serviços web. Adicionalmente, na arquitetura típica de aplicações móveis, existe comunicação com um servidor externo, mas os dispositivos próximos podem precisar de comunicar entre si, sem o intermediário de um servidor externo. Assim, combinando essas lacunas com a pressão que os desenvolvedores têm em relação ao desenvolvimento de aplicações móveis para múltiplas plataformas e sistemas operativos, surge a oportunidade de se desenvolver um sistema que fornece serviços móveis do tipo REST, em frameworks de desenvolvimento móvel híbrido. Desta forma, este trabalho propõe um sistema móvel, chamado “MobiRest”, que permite aos programadores usá-lo e integrá-lo nos seus projetos, tornando-os capazes de fornecer serviços web do tipo REST nas suas aplicações móveis.

Palavras-Chave:

Serviços Web, Desenvolvimento Móvel, Framework Híbrida, REST
Abstract

In the last few years, the use of smartphones and mobile applications has become very popular. Nowadays, there are already a wide range of mobile application development technologies. Moreover, the stack of technologies is evolving each year at a fast rhythm, with new features and abilities. However, there’s a lack of work on the recent hybrid technologies such as Ionic and Cordova for support on web services. Additionally, in the typical mobile application architecture, there is communication with an external server, but nearby devices may need to communicate between themselves, without the intermediary of an external server. Thus, combining these gaps with the pressure that developers have regarding the development of multiple-platform mobile applications, it rouses 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 allowing them to provide RESTful web services on their mobile apps.

Keywords:

Web Services, Mobile Development, Hybrid Framework, REST
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>IV</td>
</tr>
<tr>
<td>Resumo</td>
<td>VII</td>
</tr>
<tr>
<td>Abstract</td>
<td>IX</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>X</td>
</tr>
<tr>
<td>List of Figures</td>
<td>XV</td>
</tr>
<tr>
<td>List of Tables</td>
<td>XVII</td>
</tr>
<tr>
<td>List of Acronyms</td>
<td>XVIII</td>
</tr>
<tr>
<td><strong>Chapter 1</strong> Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Results</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Structure of the Document</td>
<td>3</td>
</tr>
<tr>
<td><strong>Chapter 2</strong> State of the Art</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Mobile and Collaborative Applications</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Collaborative Applications</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Mobile Social Networks</td>
<td>6</td>
</tr>
<tr>
<td>2.1.3 Example Applications</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Edge and Fog Computing</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Mobile Apps Development</td>
<td>13</td>
</tr>
</tbody>
</table>
2.3.1 Development Methods ................................................................. 13

2.3.1.1 Comparison ................................................................................. 13

2.3.1.2 Examples ...................................................................................... 14

  Fully Native ............................................................................................ 14

  Hybrid cross-platform: ........................................................................... 15

  Native cross-platform: ............................................................................ 15

2.3.2 REST ................................................................................................. 16

2.3.3 Web Services support on mobile development .................................. 17

Chapter 3  MobiRest .................................................................................. 20

3.1 Proposed solution .................................................................................. 21

3.2 Applications architecture ....................................................................... 21

  3.2.1 Server .............................................................................................. 22

  3.2.2 Client .............................................................................................. 22

  3.2.3 Global vision of the system ............................................................... 23

3.3 Requirements ......................................................................................... 23

3.4 Architecture of MobiRest ...................................................................... 25

3.5 Execution Flow ....................................................................................... 25

3.6 Implementation ....................................................................................... 27

  3.6.1 Technologies .................................................................................... 27

  3.6.2 JSON ............................................................................................... 29

  3.6.3 MobiRest bridge Java - TypeScript .................................................. 29
3.7 Use of MobiRest .................................................................................. 32

3.7.1 Development Environment Configuration ........................................ 32

3.7.2 Project Configuration ........................................................................ 32

3.7.3 Endpoints Registration ..................................................................... 33

3.7.4 Start the Server .................................................................................. 34

3.7.5 Program the endpoints ...................................................................... 34

Chapter 4 Evaluation ................................................................................ 36

4.1 Quantitative Evaluation ........................................................................ 37

4.1.1 Performance Tests ............................................................................. 37

   Settings ................................................................................................... 37

A. Laptop - MobiRest ................................................................................. 37

B. Laptop - External Server ......................................................................... 37

   Scenarios.................................................................................................. 38

   Results .................................................................................................... 38

4.1.2 Energy Consumption Tests .................................................................. 40

   Settings .................................................................................................. 40

   Scenarios.................................................................................................. 40

   Results .................................................................................................... 41

4.2 Qualitative Evaluation .......................................................................... 42

4.2.1 Demonstration app – “Travel Sync” ..................................................... 42

   External libraries of Travel Sync ............................................................ 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interfaces</td>
<td>43</td>
</tr>
<tr>
<td>REST Endpoints</td>
<td>46</td>
</tr>
<tr>
<td>Integration of MobiRest</td>
<td>47</td>
</tr>
<tr>
<td>4.3 Discussion</td>
<td>48</td>
</tr>
<tr>
<td>Chapter 5 Conclusions</td>
<td>49</td>
</tr>
<tr>
<td>References</td>
<td>51</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1 - Areas of Distributed Systems [7].............................................................................................. 5

Figure 2 - Representation of Fog computing and a Fog node [27]................................................................. 9

Figure 3 - Representation of Edge computing [27]........................................................................................... 10

Figure 4 – Representation of the subtypes of Edge Computing as well its potentials, applications and challenges [30]........................................................................................................................................ 11

Figure 5 - Architecture of MEC, including MEC servers and Edge devices [34]. ........................................ 12

Figure 6 - Mobile development approaches [34]. .............................................................................................. 14

Figure 7 – Frameworks for development of fully native applications [37].......................................................... 15

Figure 8 - Survey about the number of hybrid and native apps developers build [41]. ................................. 16

Figure 9 – Example of a RESTful endpoint. YouTube’s API [44]. ................................................................. 17

Figure 10 - Physical architecture of the system and mobile application. ....................................................... 22

Figure 11 – Architecture and execution Flow of MobiRest on a mobile app.................................................. 25

Figure 12 - Cordova application architecture [57]............................................................................................. 27

Figure 13 - Selection mechanism for web content or web services. .............................................................. 30

Figure 14 - Representation of function onRequest() that compares the request with the library of REST endpoints. ........................................................................................................................................... 31

Figure 15 - Set the plug in available. ................................................................................................................ 33

Figure 16 - Example of enabling the server and adding new routes to the server. ........................................ 34

Figure 17 – Instantiate the provider “ServerApiProvider” that was generated previously......................... 34

Figure 18 - Example of HTTP response using the API function sendResponse(). ........................................ 35
Figure 19-A - Settings for the MobiRest performance tests................................. 37
Figure 20-B - Settings for the performance tests of the external server.................... 37
Figure 21 - Temporal differences between MobiRest and an external server (on sending data). Logarithmic scale.................................................................................................................. 39
Figure 22 - Temporal differences between MobiRest and an external server (receiving data). Logarithmic scale.................................................................................................................. 39
Figure 23 - Settings regarding energy consumption of MobiRest.......................... 40
Figure 24 - Results of the energy consumption tests............................................. 41
Figure 25 - Sign up view of Travel Sync.............................................................. 43
Figure 26 - Welcome view of Travel Sync............................................................ 43
Figure 27 - Button for enabling MobiRest server on Travel Sync app...................... 44
Figure 28 - Generated QR code by Travel Sync.................................................. 44
Figure 29 - Scanning a QR code with Travel Sync............................................. 44
Figure 30 - Travel Sync's view for adding new photographs.................................. 45
Figure 31 - Options for adding friends in Travel Sync........................................ 45
Figure 32 - Feed view of Travel Sync.................................................................. 46
List of Tables

Table 1 - Overview of the LBS applications and level of accuracy required. ........................................... 6

Table 2 - REST endpoints implemented in Travel Sync. ........................................................................... 46
List of Acronyms

API  Application Programming Interface
App  Application
CRUD  Create, Read, Update and Delete
D2D  Device-To-Device
GPS  Global Positioning System
HTML  HyperText Markup Language
HTTP  HyperText Transfer Protocol
IDE  Integrated Development Environment
IoT  Internet of Things
IP  Internet Protocol (address)
JSON  JavaScript Object Notation
LBS  Location-Based Services
MEC  Mobile Edge Computing
MSNP  Mobile Social Network in Proximity
MVC  Model-view-controller
MWS  Mobile Web Services
OTT  Over-The-Top
RAN  Radio Access Network
REST  Representational State Transfer
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In the last few years, the use of smartphones and mobile applications has become very popular. Several web technology innovations have been allowing 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 demanding over the years, they want fast, and at the same time easy, and interactive mobile experience. Over the last years, statistics show that the number of downloads of mobile applications have been increasing substantially. There were 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].

In the typical application architecture, there is communication with an external server that responds to requests that are made. In this way, it is necessary to have access to the network for this communication to be possible. However, nearby devices may need to communicate between themselves, 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. 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 be connected to it.

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 for a wider range of new functionalities and features [4].

On developing mobile applications with the typical architectures, in which they communicate with an external server, there are some aspects to take into account such as: the user experience is dependent
on the speed of the Internet, if this fails the user will not have access to any kind of response from the server. Furthermore, it would be interesting to analyze the performance of this kind of architectures by hosting REST web services on a mobile application. For this reason, it is important to build a system that developers can use in their apps and that can easily provide RESTful web services, hosted inside their own applications. Thus, by integrating this system in their apps, it would be possible to serve users in the vicinity by using the already existing technology of hotspots. This way, and without needing to access the Internet, users of this type of applications would be able to remotely make any kind of REST requests to other users, like, for example, communicating, sharing data between themselves, and saving data from other users on a local database, by using this system.

1.1 Objectives

Having stated the problem, the main objective of this thesis is to build a system, called MobiRest, that allows developers to serve RESTful web services on their own mobile applications.

For such, the main goals are:

- To develop the proposed system, MobiRest, that is a library for mobile apps;
- To provide a web server and web services;
- To build a system that should be easy to program;
- The system should be efficient.

Thus, by integrating this system in developers’ apps, it would be possible to serve users in the vicinity by using the already existing technology of hotspots. This way, and without the need to access the Internet, users of this type of applications would be able to remotely make any kind of REST requests to other users using this system, like, for example, communicating, sharing data between themselves, and saving data from other users on a local database.

Throughout this work, the main steps to fulfil the objectives are as follow:

- To design the architecture of the system;
- To adapt a webserver to be able to provide REST services;
- To describe how other developers can use this system;
- To create a demonstration app with this system;
- To test the performance of the system;
1.2 Results

After a long process of development and testing, the proposed system, named MobiRest, was implemented, accomplishing all the objectives mentioned in 1.1. The results consisted in evaluating the final system in two different ways: quantitative evaluation, where the performance and energy consumption were analyzed on whether they meet the proposed objectives; and qualitative evaluation by demonstrating the system on a real-life application – Travel Sync.

The results of the hard work throughout the implementation of the system translate into a tool capable of provisioning RESTful web services in mobile devices.

1.3 Structure of the Document

Following there’s a brief summary of this thesis structure, chapter by chapter.

Chapter 2 will discuss the concepts of collaborative apps and mobile social networks in proximity and a few examples of application. Also, the theme of edge and fog computing and the literature review regarding frameworks for mobile development and support for mobile web services will be presented.

Chapter 3 will explain in detail the requirements, architecture and implementation of the proposed system, MobiRest. At the end of the chapter, the question of how other developers can use and integrate this system into their apps will be discussed.

Afterwards, in Chapter 4 there’s a demonstration of the system’s features and a discussion of its evaluation.

At the end, Chapter 5 concludes this thesis and makes a short review of the previous chapters, as well as the limitations and possible future work to continue this project.
Chapter 2

State of the Art

This chapter explores the concepts of mobile and collaborative applications. Moreover, the state of the art introduces and resumes mobile social networks in proximity. Other relevant concepts are the Edge and Fog Computing, that emphazise the decentralization of the Cloud processing. Finally, there’s a summary of the several methodologies of mobile development as well as the literature review on web services’ support for smartphones and other mobile devices.
2.1 Mobile and Collaborative Applications

2.1.1 Collaborative Applications

Collaborative applications, also mentioned as “Groupware applications” are defined as a “software application that (a) interacts with multiple users, i.e., it receives input from multiple users and creates output for multiple users, and (b) links these users, i.e., it allows some input of some user to influence some output created by some other user” [5]. For example, a chat application is qualified as being collaborative once a user can send a message to one or more collaborators. Another example is a database application in which a linkage occurs when some user gets data that was deposited by another user. In other words, they are multi-user applications that couple their users, giving the sense of being there even when users are in different locations and by supporting mechanisms of collaboration that cannot be supported in direct face-to-face interaction such as asynchronous collaboration [6]. In Figure 1, it is possible to visualise the areas that collaborative applications overlap.

![Diagram of areas of distributed systems](image-url)

*Figure 1 - Areas of Distributed Systems [7].*

This thesis will enhance the relation between Mobile and Collaborative areas. As shown in Figure 1 collaborative applications may also be real-time applications like chats, may contain multimedia such as file sharing and of course be inserted in a mobile context.
2.1.2 Mobile Social Networks

The modern mobile applications are making use of and reinventing the way of using geospatial technologies. At the beginning, mobile phones had the sole purpose of providing voice and text communication, but nowadays that is just one slice of the “cake”. Other important factors are the web browser and the GPS services [7]. This allows developers to create mobile applications that include LBS services (referred as “Location-Based Services”). LBS is defined as “a platform that provides information services based on a known or the current location, supported by the electronic map platform”. The tourist information systems are perfect examples of this kind of applications, for example Foursquare [8], where tourists can use location to search for restaurants or nightlife places to go to. Other applications such as location tracking services, for example for the safety of children, or context aware games are cases of LBS services. On Table 1 there’s an overview of the LBS applications and level of accuracy required.

<table>
<thead>
<tr>
<th>Application</th>
<th>Accuracy</th>
<th>Application</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>News</td>
<td>Low</td>
<td>Gaming</td>
<td>Medium</td>
</tr>
<tr>
<td>Directions</td>
<td>High</td>
<td>M-Commerce</td>
<td>Medium to High</td>
</tr>
<tr>
<td>Traffic Information</td>
<td>Low</td>
<td>Emergency</td>
<td>High</td>
</tr>
<tr>
<td>Point of Interest</td>
<td>Medium to High</td>
<td>Sensitive Goods Transportation</td>
<td>High</td>
</tr>
<tr>
<td>Yellow Pages</td>
<td>Medium to Low</td>
<td>Child Tracking</td>
<td>Medium to High</td>
</tr>
<tr>
<td>Car Navigation</td>
<td>Medium to High</td>
<td>Pet Tracking</td>
<td>Medium to High</td>
</tr>
<tr>
<td>Personal Navigation</td>
<td>High</td>
<td>Electronic Toll Collection</td>
<td>Medium to High</td>
</tr>
<tr>
<td>Directory Assistance</td>
<td>Medium to High</td>
<td>Public Management System</td>
<td>Medium to High</td>
</tr>
<tr>
<td>Fleet Management</td>
<td>Low</td>
<td>Remote Workforce Management</td>
<td>Low</td>
</tr>
<tr>
<td>Car Tracking</td>
<td>Medium to High</td>
<td>Local Advertisement</td>
<td>Medium to High</td>
</tr>
<tr>
<td>Asset Tracking</td>
<td>High</td>
<td>Location-Sensitive Billing</td>
<td>Medium to Low</td>
</tr>
</tbody>
</table>

In the examples of Table 1, traffic information, points of interest, personal navigation and directory assistance are good examples of applications that can use integrated resources of proximity.

Social networks are the new reality of people. On the digital era, people have the need to communicate with each other, share their thoughts, impressions, search for recommendations, and be part of a global community where the distance has become smaller due to the easiness of communication all over the planet. Social networking made a great impact on the communication all over the world. Mobile applications like Facebook or Instagram made this possible.

On the other hand, the new network paradigm is to build social networks over mobile applications in events or locations. This paradigm allows people to communicate and share experiences with minimum required infrastructure and without needing access to the Internet [9]. The literature review in this area will be discussed in deeper detail in further sections.
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 [10]. The essential difference between LBSN (location-based social network) and MSNP is the physical geographical coverage. The first can refer to any place in the world and is shared globally over the community. Whereas the second emphasizes social interaction on the proximal area, thus the social content is shared with the nearby users on the range of wireless network communication technology.

This type of applications and social networks can be seen as a solution for the limitations in physical meetings due to the facilitation of face-to-face interaction between users that are close to each other. There are many case studies where it can be implemented which will be discussed further.

In this theme, there are essentially two approaches to enable proximity awareness: Over-The-Top (OTT) and Device-To-Device (D2D) [11]. In the former there is a centralized server which takes care of the interactions between proximal users. This server collects the locations and profiles of users, using it to discover geographical proximity and social similarity between different users. The latter, as the name states, corresponds to the communication point-to-point of proximal users on the range of wireless signal.

Some researchers discuss the decentralized mobile social networks. These are represented by a group of participants on a network using their mobile applications dynamically without a centralized server. This type of architecture isn’t a commercial product on the market of mobile applications and social networks yet. Still, there has been some investigation in this area. Researchers focus on ways to support decentralized systems to overcome the limitations of centralized systems. The Proximiter project [12] allows content sharing on portable devices by the implementation of an HTTPD web server (Apache Hypertext Transfer Protocol Server) on proximal devices.

Over the last few years, there has been some investigation to find solutions for MSNP. Pratistha [2002] [13] Srirama [2006] [14], Doulkeridis [2007] [15] and Pawar [2007] [16] proposed mobile web service (MWS) solutions hosting web services on mobile devices. Furthermore, within the scientific community, there has been attention given to the MobiClique project [17], a middleware for mobile social networking in proximity by the use of Bluetooth/Wi-Fi technologies and opportunistic networks.

2.1.3 Example Applications

Mobile Learning, or M-Learning is defined as “the process of learning mediated by handheld devices such as smart phones and tablet computers” [18], [19]. This is a field with growing importance nowadays. Students and teachers can take advantage of the technologies’ evolution and use them as a tool on the learning process. As mobile devices become increasingly prominent worldwide, m-learning is already taking action and it is a promising area of development and investigation. From national education ministries to local schools, there are supportive policies being experimented with to promote new ways
of mobile learning. This provides an extension of the traditional methods of teaching and brings dynamism, quick access to information, easy sharing of files, and many other advantages.

Furthermore, it facilitates the use of methodologies such as gamification, and new ways of learning with quizzes and games. In general, the old classrooms have space to become more interactive and interesting.

Thus, building mobile applications in which there is a location context and, more specifically, a proximal network of participants (in this case, for students and teachers) is very relevant. This way, the use of mobile technologies can add value and make accessible a more modern methodology of interacting. The future of learning is in developing tools that take advantage of the newer technologies, for example mobile applications that can be used in various contexts: field trips [20], tours of heritage sites and museum visits [18]. Mobile Field Trip, is an example of a project in which students with access to mobile application resources can answer quiz questions about geology on field trips. In this case it would be useful to have a system that allows these applications to work without needing to access the internet.

In collaborative applications it is very important to have tools to provide communication in real time. The term “chat” in the computer world means talking to other people using a computer [21].

There are many examples of chat applications but in this context, the discussion is about those who are connected to the theme of Mobile Social Networks in Proximity (MSNP). Yufeng Wang [11], presented an hybrid chat system, HYChat, for MSNP. According to this system, when users are on the range of wireless signal, the application enables interaction using D2D methodology; when two devices aren’t in a Wi-Fi’s direct range, automatically it is switched to the online mode, in which the interaction is intermediated by a central server via Internet.

Tourism is another sector that could draw benefit from an MSNP provider system. Consider a guided tour, for instance. It would be interesting and useful to have a mobile application that could connect the tourists and the tourist guide to add value to the tour. The guide could share information or images in real time, offline, without access to Internet.

The case studies discussed are real life examples of application where this type of systems may be useful. In fact, there are many examples where this type of system can add value and enable new features and ways of communication.

2.2 Edge and Fog Computing

Technology has already advanced tremendously. It made possible for people to have videoconferences, or to share experiences in the highest resolution of 4k or at least full HD. This evolution provides new features and originates new concepts. For example, video streaming has evolved into augmented reality.
and virtual reality. This factor causes an increase in the amount of data generated each day. But these wonders come with a price: the network bandwidth limit. Regardless of the source, collecting ever-increasing amounts of data pushes up the network bandwidth to its limits [22]. Furthermore, the following questions emerge: Will the cloud network alone be able to support the huge amount of data that is being generated? Is it absolutely necessary to push all the information data to the cloud? To find an answer to these questions and with the development of the IoT (Internet of Things), two solution models were presented, involving the decentralization of the processing layer: fog and edge computing.

The term “Fog Computing” was created by Cisco [23] and is defined as “a model to complement the cloud for decentralizing the concentration of computing resources (for example, servers, storage, applications and services) in data centers towards users for improving the quality of service and their experience” [24].

“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. This reduces the communications bandwidth needed between sensors and the central data center by performing analytics and knowledge generation at or near the source of the data” [22]. Edge devices may not stay continuously connected to a network, for example laptops, sensors, and smartphones. By eliminating the distance and time it takes to send data to centralized sources, it enables the improvement of the speed and performance of data transport, as well as devices and applications on the edge [23]. The key difference between Fog and Edge computing is that Fog is associated with cloud services, processing and storing the data generated at a Fog node close to the source [25] whereas Edge computing is defined by the exclusion of the cloud [26]. On Figure 2 and Figure 3 there’s a representation of Fog and Edge computing.

These themes are relevant since this work provides a solution for the decentralized systems.

![Figure 2 - Representation of Fog computing and a Fog node](image)

On Fog computing the intelligence is pushed down to a local area network in opposition to the centralized cloud. This local network is called Fog node [27].
There’s a relation between Fog and Edge computing, but in the latter the intelligence is pushed directly to devices, as, for example, automation controllers (PCA’s) [27]. These Edge devices are able to make decisions and store data.

On the literature, there’s been, over the last years, some interchangeability between both terms. According to some authors, both terms are equivalent. On the other side, other authors like Soumya and Koustabh [28] establish the comparison of both and present Edge computing as intermediate layer between the end devices (called as “Edge devices”) and the cloud. This layer can be classified into three types: Fog Computing, Mobile Edge Computing and Cloudlet Computing. On Figure 4 is presented a diagram with the various types of Edge Computing and the advantages of these technologies.

The different variations of Edge Computing will be explained according to this definition.

Fog computing, as discussed before, illustrates a platform that transfers Cloud Computing to the proximity of end users. It presents a computing layer that leverages device-to-device gateways and wireless routers, which are called “Fog Computing Nodes” [28]. These nodes are responsible for computing and storing data locally, at the end devices, before sending it to the Cloud. According to some researchers [28], [29], it should be noted that opposite to other types of Edge Computing, these Fog Nodes, are placed at any point of the hierarchy between the Cloud and the end devices. Another relevant characteristic of Fog Computing is the heterogenic nature it offers in respect to its nodes, including access points, routers, switches, IoT gateways, set-top boxes, not being limited only to these elements.
Figure 4 – Representation of the subtypes of Edge Computing as well as its potential, applications and challenges [30].

Also, Fog Computing is specifically designed for applications that require real-time response with the minimum latency possible. Obviously, IoT applications are good current examples for this point.

Like Fog Computing, Cloudlet is a new architectural paradigm which represents the middle tier of a 3-tier structure: end device, Cloudlet, and, at the top of the hierarchy, the Cloud. In the current language, Cloudlet is seen as a “data center in a box” with the purpose, once again, to carry Cloud capabilities closer to the end user [30]. Koustabh et al. [28] states that on the literature review there are several authors on the various Edge Computing technologies but, on the other hand, there’s a lack thereof on establishing the differences between their implementation. Moreover, Koustabh et al. and other authors make precisely this comparison, emphasizing the major characteristics relevant for each one. Thus, in other words, Cloudlet can be defined as “a trusted cluster of computers, well connected to the Internet, with resources available to use for nearby mobile devices” [28], [29].

This concept appeared with the convergence of mobile computing, IoT and Cloud Computing, prototyped by the Carnegie Mellon University [30]. Satyanarayanan et al. [29] presented the concept of virtual machine-based Cloudlets, which was one of the pioneer projects on this subject. According to this prototype, running a virtual machine (VM) with capabilities for provisioning resources to the end
users, over a WLAN network with a high bandwidth, results in a low latency and real-time response for the applications.

Mobile Edge Computing (MEC) is a new paradigm initially presented by the European Telecommunications Standard Institute (ETSI) as “a new platform that provides IT and cloud-computing capabilities within the Radio Access Network (RAN) in close proximity to mobile subscribers” [26], [31]. Like other kinds of Edge Computing, there are MEC nodes co-located with a Radio Network Controller that run multiple instances of their host, which is capable of computing and storing data on a virtualized interface [28], [32]. These intermediate nodes have MEC servers in their location, as represented in Figure 5, that, by running and processing tasks closer to the customers, make it possible to reduce the network congestion and consequently the response time to the requests made.

Hence, MEC can be described as Cloud servers running at the edge of mobile networks and that have the purpose to serve as a complement to the centralized Cloud services. More precisely, it consists of a middle layer that processes requests made from the end devices, deciding whether it must be forwarded to remote Cloud or if they are able to generate a quick response. Thus, MEC offers location and context awareness, as well as higher bandwidth with lower latency for the customers.

Figure 5 - Architecture of MEC, including MEC servers and Edge devices [34].
2.3 Mobile Apps Development

This subsection will start by exploring the mobile development methods that already exist for the several types of applications and will describe their main examples. Also, it will discuss the literature review on support for web services in mobile devices.

2.3.1 Development Methods

There are essentially three types of approaches on building mobile applications: fully native apps, mobile web apps and cross-platform apps. Below, a comparison of those methods is presented.

2.3.1.1 Comparison

Fully native apps can access all the platform specific features, including access to the hardware of the device, for example the camera or accelerometer. This type of app is developed using a specific platform technology (e.g.: Java for Android, Swift for iOS).

Web mobile apps, in contrast, are built using standard web technologies, for example HTML5, or JavaScript, so they run in every mobile device with a web browser, although they are very superficial because they can only access features supported by HTML5.

Cross-platform apps are the combination of the last two approaches: they are developed using the web standard technologies, which means they support multiple platforms. Moreover, with the pre-installation of specific plugins they can access all the device features that a native app does [33].

Furthermore, cross-platform apps are divided into 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. This means that native cross-platform will be closer to native apps.

Figure 6 illustrates the major differences between the three types of mobile development.
As seen in Figure 6, native and hybrid mobile apps can gain access to the hardware functionalities of the smartphone through device APIs, while web apps are limited and cannot have access to these functionalities.

### 2.3.1.2 Examples

**Fully Native**

When developers are building a fully native app, it is specific for a certain operating system. This presents the highest potential cost on the mobile applications market, since, in general, companies want to reach the most users possible, and so, this requires having developers for each operating system.

For this type of applications, programmers use IDEs (Integrated Development Environment). The already existing platforms for development in the native mobile development are: for Android, developers program in Java using the Android Studio IDE; for iOS devices, Objective-C has been the most dominant programming language. Also, Swift emerged in the year of 2014, and took the place of Objective-C for iOS devices. Both languages use the XCode IDE; for the Windows Phone devices, the used languages are C# and Visual Basic along with Microsoft tools, which use Visual Studio IDE. In Figure 7 there’s an abstract of the main languages and IDEs for each type of devices.
Hybrid cross-platform:

In respect to hybrid cross-platform apps, several frameworks for mobile development have been developed. However, the ones that come up time and time again are:

Apache Cordova, an open-source mobile development framework that allows developers to build cross-platform apps using the standard web technologies such as JavaScript, HTML5 and CSS [34]. In an overview, this type of applications run within wrappers targeted for each platform (it might be for example Android, or iOS, etc), and accordingly, rely on standards-compliant API bindings that enable the access to the hardware functions of the device, such as data, network status, camera, etc;

PhoneGap [35] is a framework built on top of Cordova, more precisely, it is a distribution on Cordova. The major benefit is that it offers tools that can run in the Cloud;

Ionic [36] is another framework that is built on top of Cordova. It improves the design of the applications by integrating the well-known Single Page Application framework called Angular;

Trigger.io Forge is a cross-platform framework that enables web developers to create native mobile apps using HTML5 [37]. It contains JavaScript API that provides functionalities and UI components such as camera, contacts, top bar, etc.

Native cross-platform:

This type of application is gaining relevance over the last years. The most famous example is React Native, as an extension of React.js. It is an open-source Single Page Application framework that allows developers to build native applications using JavaScript and React. Accordingly, with React Native “you don't build a ‘mobile web app’, an ‘HTML5 app’, or a ‘hybrid app’. You build a real mobile app that's indistinguishable from an app built using Objective-C or Java” [38].
Another example of framework is NativeScript. Like the previous ones, it is also an open-source framework for building truly native mobile apps with Angular, TypeScript or JavaScript [39].

Also popular is Xamarin, a mobile cross-platform using C# language, which comes with its own IDE [40].

In the past two years, what percentage of your apps were hybrid (mix of web and native code)? What percentage do you expect in the next two years?

![Survey about the number of hybrid and native apps developers build](image)

In Figure 8 we can see the results of a survey made to developers about the percentage of native and hybrid mobile apps they developed recently. The results show that hybrid and cross-platform apps are becoming more popular.

### 2.3.2 REST

REpresentational State Transfer, is the acronym for REST, published for the first time by Fielding [42]. It is an architectural style based on HTTP protocols for web services that encourages applications to be simple, by defining specific constraints. This type of web services is recognized for having good performance, being lightweight, scalable and modifiable.

On this architectural style, all the exchanging data and functionalities are considered a resource, and each resource is accessed by a URI. Then, according to the specified HTTP method applied on the request, certain operations are executed. The basic principles regarding REST architecture are characterized by the following constraints: client – server architecture, uniform interface, statelessness, tiered and cacheable.

The HTTP methods correspond to a major part of the “Uniform Interface” constraint. REST architecture takes advantage of the HTTP different operands that are associated with CRUD operations and are described as follows:
Create: associated with POST verb of HTTP. This operation creates or adds new entries in database systems;
Read: this operation is related to the GET verb, which searches and retrieves information from the user;
Update (modify): is associated with the PUT verb and modifies an entry or a raw of a table in a database system.
Delete: obviously it corresponds to the DELETE verb of HTTP, which deletes information stored in a database.

As an example, shown in Figure 9 is an endpoint of YouTube’s API, which consists on a URI associated with the GET HTTP verb.

```
GET {base_URL}/subscriptions?part=snippet &mine=true
```

Figure 9 – Example of a RESTful endpoint. YouTube’s API [44].

Figure 9 represents the endpoint of YouTube’s API, that, when applied, forces the server to send back a list of the channels that a user subscribes to.

### 2.3.3 Web Services support on mobile development

On this chapter, the concept of collaborative applications was already introduced and this theme has then been further explored and deepened by discussing the mobile applications based on location and proximity. Additionally, and related to this subject, the literature review on Fog and Edge computing has been discussed. Hence, it is time to discuss the state of the art on the development and support for web services on mobile devices.

On the area of collaborative applications, developing a service-oriented application, namely MSNP’s, requires the need to obey the standard protocols that support the interoperability between several users, applications, devices and operative systems.

Web services, officially defined as “a software system designed to support interoperable machine-to-machine interaction over a network” [43], are the piece that enables the interaction between the server and the interface of 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 a technology based on oriented object approach that define a standard protocol for XML format exchange on communication. Moreover, REST was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation [42]. It stands for Representational State Transfer and
differs from SOAP in the way that it is based on a resource-oriented approach. It follows an architectural style that obeys several constraints.

Yoshikawa et al. and other researchers defend that the idea of “mobile web services” is designed for the consumption of an API on client-side applications for mobile devices [10], [44], [45]. On the other hand, more recent works from researchers such as Gehlen and Pham [46], Pawar et al. [16] and other researchers suggest the use of mobile web services as “mobile-device-hosted web services”. On this new vision, mobile devices like smartphones, PDAs and other media players are the providers of those web services.

According to Chii Chang [10] a mobile web service represents a mobile service that ensue standards of web services, this is, operated by a web service standard protocol such as SOAP or REST. Essentially, there are two components required to provide web services on a mobile host device: WSDL (Web Service Description Language) and SOAP/REST. Furthermore, these technologies are supported by components built on top of network socket servers.

Nowadays, there are already many open-source tools that can easily turn a smartphone into a web server. Some examples of these tools are: Nokia Mobile Web Server [47], which provides web services on Symbian operating system; CocoaHTTPServer [48] and Mongoose Web Server [49], which allow iOS devices to run a HTTP web server; iJetty [50] and kWS [51], which let Android devices provide HTTP web services.

Over the past years, mobile web services have been used in several application domains.

Berger et al. [52] developed a web application capable of provisioning SOAP web services on mobile devices within a local network using access points such as Wi-Fi or Bluetooth and a custom webserver, despite their approach lacking efficiency since SOAP and XML formatting aren't scalable, due to the fact that SOAP messages are difficult to cache.

Afterwards, Srirama et al. [14] developed a lightweight mobile hosting SOAP-based system with a handler that points to the appropriate web service. The handler reads the HTTP messages and deserializes the requests. Comparing to the previous approach, this one is more efficient and lightweight since a compressed version of SOAP is used (kSOAP), although it still uses the XML formatting for service descriptions and interchange of messages.

Moreover, Kim and Lee [53] suggested the replication and migration of web services and, thus, sharing tasks between devices. Accordingly, this migration can occur when there's a change of context on a device, for example shortage of battery level, changes in location, etc. Therefore, it stops providing the service on its own. Even though this approach reduces the workload on the device, SOAP was still used as the protocol of communication, which is heavyweight [54].
On the 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 [55] implemented a mobile host provider for REST services on Android. Accordingly, REST is more efficient than SOAP web services. Besides that, on this work JSON formatting was used as the intermediary for the interchange of messages, which is more lightweight than the XML formatting implied in SOAP services.

Also AlShahwan and Moessner [3] compared SOAP and REST web services in mobile hosts. They developed two frameworks in Java for Mobile Edition (JME) for providing web services of each type, respectively. Their results lead to the same conclusion: that RESTful-based frameworks have better performance than SOAP-based frameworks. This has mainly to do with the fact that REST architecture supports caching and demands lighter processes power.

Most of the previous works about the provisioning of web services into mobile devices use aged technologies like SOAP and XML. 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 in this context.
Chapter 3

MobiRest

This chapter will describe the proposed system, MobiRest, the requirements to consider while developing the system, as well as the architecture, implementation and how other developers can use and integrate it in their applications.
3.1 Proposed solution

Nowadays, there are already a wide range of mobile application development technologies. The stack of technologies is evolving each year at a fast pace, with new features and capabilities. Moreover, mobile devices are also undergoing a big evolution. Brands are competing against each other to have products with the newest and most exclusive features, before their competitors. Moore’s Law says that the number of transistors doubles approximately every two years. In analogy to this theory, mobile devices have also increased their processing and memory capabilities, becoming more powerful in both hardware and software each year.

As discussed before, there has been some work done on decentralized systems which 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 feel regarding the development of multiple-platform mobile applications, it rouses the opportunity to develop a system which provides mobile web services in hybrid development frameworks.

This thesis 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. Through the integration with the proposed solution, development projects will be able to make and respond to REST requests via HTTP standard protocols such as GET, POST and DELETE.

3.2 Applications architecture

This section will focus on the architecture of the system (integrated on a mobile app), within a local network.

The MobiRest system’s purpose is to provide communication and services among collaborative mobile social networks in proximity. Therefore, it can be used in multiple mobile devices that contain the necessary hardware requirements above mentioned.

When using the MobiRest system, since it is built with the purpose of communication and interaction with other devices over the same network, several users of this network can access the provided endpoints.

Figure 10 shows the structure of applications with MobiRest system embedded on mobile devices.
Thus, on its architecture it distinguishes, for the same device, two different states in which applications will operate:

a) As a server;

b) As a regular client.

### 3.2.1 Server

The system will have the embedded server activated, as well as the hotspot functionality enabled. Moreover, all the REST requests made within the same local network will be addressed to this server’s corresponding IP and Port, which will process them and respond according to the REST API implemented.

### 3.2.2 Client

The system will be in client mode with the server disabled and connected by Wi-Fi to the hotspot created by the user mentioned in point a). Furthermore, users in client mode will make REST requests to the user, sharing information and media content among members of the local network.

Please note that this is the case when multiple users are using mobile applications that have integrated MobiRest system. However, any other client applications can connect to mobile devices that have MobiRest embedded on their mobile apps. This means that this corresponds to a system that allows...
interoperability between multiple devices, applications, systems and enables machine-to-machine interaction over the HTTP standard protocols.

### 3.2.3 Global vision of the system

The architecture of the system corresponds to a typical Client-Server structure during the period of time that users wish to share media content. Meaning, although the system respects a client-server architecture, the point is that the server is not centralized. Each user can be using MobiRest playing the role of a server, but also, some other times, they can act just like a single client and make web and REST requests to other applications using 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.

### 3.3 Requirements

As mentioned in section 3.1, a system capable of hosting and consuming RESTful Web Services on mobile devices will be developed. These services, explained more in-depth further on, imply endpoints according to each resource that is manipulated. Furthermore, these endpoints vary from application to application, depending on the specific features they each application might have.

1. **MobiRest** will be used in hybrid mobile applications. It will be easy to integrate with existing applications developed with Cordova (or Apache Cordova family technologies) and/or Ionic. Thus, this system will facilitate RESTful web services communication as simply as adding the plugin of this system to existing applications developed on the technologies above mentioned;

2. **MobiRest** will be targeted at the Android Operating System;

3. **MobiRest** will support the provisioning of RESTful web services on hybrid mobile applications. As mentioned before, the proposed system will be able to provide REST mobile web services, hosting them inside the mobile device. On the next chapter, a demonstration will be presented, an example of an application with defined endpoints for different web services implemented;

4. **MobiRest** will use Wi-Fi and hotspot technologies;

5. **MobiRest** will work in offline mode. In other words, without “3G connection”. Users won’t have the need to access the Internet to share content and communicate with other users. This communication will be possible by using Wi-Fi and hotspot technologies;
6. *MobiRest* will be easily programmable, after adding the corresponding system plugin. Also related to the previous requirement, the system will provide API functions to facilitate this integration and it will be easy to program according to developers’ needs.

7. *MobiRest* will have a dedicated file in JavaScript/TypeScript where developers will register an API for their systems’ endpoints by using the offered API functions of *MobiRest*. Thus, the system will be adaptable and easily changeable without the concern of having to study the full system structure and avoiding the need to edit other files of the system. Thus, after adding the plugin to their applications, developers will only need to adapt the file that will consist on a library of endpoints. At the end, the system will be distributed as a product, easily configurable to fulfill developer’s needs, a tool capable of providing and hosting REST Web Services;

8. The server, which will be based on the “Cordova-Httpd” plugin [56] will require several features:
   • It will have support for CRUD operations by supporting at least the following HTTP methods: GET, POST and DELETE;
   • It will have a mechanism that allows the distinction between requests for serving a web page or a REST request. It will successfully serve both types of requests;
   • It will redirect to a main page when it doesn’t find the file/page requested;

9. It should be efficient in terms of performance and battery usage.

10. It should be a lightweight system.
3.4 Architecture of MobiRest

This subsection describes the architecture of the MobiRest system and its execution flow.

Figure 11 represents the architectural blocks of MobiRest. The “App Logic” block contains all the configurations necessary to integrate MobiRest system on a mobile app. These configurations will be explained in further detail later in section 3.7 – Usage of MobiRest. Besides that, it contains the library of endpoint functions that are necessary to execute according to each web service.

The next block is the main structure of MobiRest. First, it contains a Java webserver inside the plugin “Cordova-httpd”. 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 up the parts of a mobile application.

3.5 Execution Flow

The execution flow is made according to Figure 11. Each step is numbered and described as follows:
1. On the first step, MobiRest receives the information needed for the integration of the web services developed. First, the developer programs the intended endpoint functions on a TypeScript file, named library.ts. Afterwards, they push the related information to a vector that includes a pointer for the endpoint functions and the description of each endpoint regarding the corresponding HTTP method and URI. Now, MobiRest knows which of the RESTful web services to serve, simply by adding a new library package to the system.

2. After informing the system of which endpoints to serve, MobiRest is able to respond to requests. The Cordova native plugin, named Cordova-Httpd, is a webserver built in Java that receives all the requests, in the format of HTTP messages.

3. On this step, the received HTTP requests are intercepted and parsed in their body and headers. If the requests are only singular 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. This step is the most interesting part of the execution flow, and also the most challenging on the development of the MobiRest system. Here, the received HTTP requests, if they are RESTful, are transposed to the TypeScript layer through complex mechanisms of asynchronous programming, waiting then for the REST response. More deepened details regarding this step will be explained later, on section 3.6.3.

5. Within the TypeScript layer of MobiRest, the requests are analysed and, according to their URIs and HTTP methods, the corresponding 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, the MobiRest system builds a response with the results achieved on step 5. This response consists on another HTTP message which varies according to the type of request that was received.

7. Similarly to step 4, the HTTP response built, is forwarded to the Java layer. Once again, this bridge is made by asynchronous complex mechanisms that will be discussed in more detail further on.

8. At this step, 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.
3.6 Implementation

This subsection presents the technologies, development frameworks, programming languages, external libraries, files structure of MobiRest and the deepened details of the steps 4 and 7 of section 3.5, adopted when implementing the MobiRest system.

3.6.1 Technologies

Cordova is one of the frameworks used along the implementation of MobiRest. The process until the final Cordova application is represented in Figure 12. It works through the transformation of the web languages into each platform (Android, iOS, Windows Phone, etc) with API bindings that allows for the access to the native functionalities of devices. Thus, this type of application is executed in a WebView within the native application wrapper, which renders the HTML components. This means that when developing native plugins in Cordova, they must be developed on each platform’s programming language that is desired for the plugin to support. Hence, taking in account the short period of time for this thesis and the difficulty associated with its implementation, MobiRest focused on Android platform.

In the world of mobile applications, most of the Android apps are written in Java language since it runs on a Java language environment. Aside from that, it is easier than other development languages for Android such as C++, which has the pointer arithmetic, and there’s a large number of developers already proficient in Java. These reasons make Java a famous and common programming language while developing mobile apps in the Android operating system. Furthermore, the Cordova-Httpd [56] plugin
that was used as a basis for this thesis didn’t escape this rule, it uses Java on its backend — the web server. Thus, a great piece of MobiRest’s implementation was made in Java.

Still on the theme of technologies, Ionic stands out since it is a Hybrid Cross-Platform framework that offers the development power for building mobile apps with “native-feeling”, using web standard technologies like HTML, CSS and JavaScript/TypeScript. Complementing Cordova, the main focus of this framework is on the look and the UI of the mobile apps, leveraging the interaction with Cordova native plugins. As such, this framework was chosen to be used for the application’s logic and frontend. More precisely Ionic 3 version was chosen, as it is recent and compatible with Angular 4.0 version, which introduces several new features, smaller and faster apps.

Since Ionic 3 was chosen for this work, for the reasons mentioned above, it is relevant to talk about Angular 4.0, which comes together with this version of Ionic. Angular 4.0, in opposition to AngularJS that uses the concept of Model-View-Controller (MVC), takes the most modern frameworks’ paradigm, in which the structure should be oriented by components. A big advantage of Angular 4.0 is that it allows the loading of content only when required, which further improves the performance and responsiveness of a mobile application, important for the performance of MobiRest.

In this paradigm the structuring of the code between providers, pages, data mocks and models may be a little confusing at the beginning, but, in fact, it is very useful and provides non-repeatable code, since the same component can be reused. Moreover, since the purpose is for other developers to program their own endpoints for web services, it is much easier with this type of structure to integrate and add new library files to the system without having to study the entire files of the system. The fact that this technology is made by blocks (components) makes it easy to add new pieces of code that interact with the full application, thus simplifying the task for other developers to configure their endpoints on the MobiRest system.

For these reasons, the pack Ionic 3 – Angular 4.0 – TypeScript was chosen as it seemed the most suitable choice for this project.

At the beginning, the implementation of MobiRest started with the backend on the Cordova framework. The reason why Cordova was chosen is due to the fact that it is an open-source framework for mobile development that allows developers to build mobile applications solely with their knowledge on the web standard technologies such as HTML, CSS and JavaScript. Thus, web developers, besides building websites, can also build mobile apps for cross-platform. Moreover, this technology allows access to the hardware features of mobile devices by installing Cordova native plugins (for example for accessing the device’s camera, databases, QR Codes, etc). It is within this context that MobiRest fits. The overall idea for this system is to provide a Cordova plugin easy to install on hybrid applications.

The Cordova plugin mentioned previously was used on this thesis, cordova-httpd [56], which was based on the tiny web server NanoHttpd (Java) [58] because it contains an embedded web server for hybrid
mobile applications in Cordova and/or Ionic. 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, throughout the course of this thesis, the major part of the implementation was made by programming over this server in order to provide the objectives proposed.

To summarize, fundamentally, this thesis 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.

3.6.2 JSON

Along the implementation of MobiRest, the JSON format was utilized. JSON stands for JavaScript Object Notation [59]. It is a lightweight data-interchange format that is easy for humans to read and write. A JSON corresponds to a text format that follows some conventions familiar to programmers of the C-family of languages (C, C++, C#, JavaScript, Java, Python, Perl, among others), which makes it excellent for data-interchange.

To answer the question “What is the reason for using JSON?” let’s contextualize its emergence first. Initially, the language for data-interchange was XML, but this had a few limitations, for example being hard to manage in JavaScript. Hence, web developers started to use JSON instead of XML for the data exchange between the web and mobile clients and backend services [60].

Regarding web services, SOAP relies exclusively on XML formatting, although REST architectural style doesn’t imply the need to use XML. So, on this thesis the data exchanging language used was JSON as it’s more lightweight and easier to manage on the universal programming language in browsers and on the web – JavaScript.

3.6.3 MobiRest bridge Java - TypeScript

This subsection covers the details of steps 4 and 7 of section 3.5 and explains the mentioned concepts of asynchronous programming done along this thesis. As it was mentioned before, the most challenging part on the implementation of MobiRest is on the Java – TypeScript bridge. This process is explained as follows.

The technologies used along the implementation of MobiRest, namely Cordova, imply the use of asynchronous programming. It is defined as “a form of parallel programming that allows a unit of work to run separately from the primary application thread” [61]. When this unit of work is concluded, the main thread receives a notification with the results and whether it was successful or if it failed. This paradigm, although being complex, is common in servers, because the response is not automatic, and depends on external communication factors, for example, latency and network traffic. Furthermore, the concept
of “callback” [62] emerges, which corresponds to an executable code passing through arguments on a function. The moment when this code is executed is undefined, it may be immediate, or it might happen later on, which is why it is asynchronous.

The API functions used for enabling the server, or when a certain request arrives, are asynchronous. Moreover, the server is the basis for all the communication and implementation throughout this thesis, so it’s an important subject to discuss. Furthermore, asynchronous programming represents a complex concept which is hard for debugging, at least on the web technologies and frameworks used, as well as on a mobile application. The necessary changes to the original system of the server, on this subject, will be discussed below.

First, on the Java layer, the requests that go to the webserver are intercepted and the information that comes with the HTTP request is parsed and stored in several variables. These variables contain important information regarding the URI, HTTP method and other parameters that are necessary further on to process the received request. If there’s a POST request, for example, it keeps the content of the body (raw input) on a variable.

After, a selection mechanism, shown in Figure 13, was created, which analyses if the request is a regular HTTP web request or if it corresponds to a web service. This process occurs by reading the URI and detecting if the path starts with “/files”, which corresponds to the folder where the web pages and other files should be stored to serve the clients. If the request is asking for web content, it is forwarded to a function named “serveFile()” on the Java webserver of Cordova-Httpd plugin. On the other hand, if it corresponds to a web service, it calls a function in Java, named “serveJavascript()”, that is responsible for making the interaction between the Java and the TypeScript layers. Thus, this function is invoked, carrying as parameters the variables referring to the request’s headers. Inside this function a JSON is created with the mentioned variables. This is the mechanism used for the interoperability between different programming languages.

```
public Response serve( String uri, String method, Properties header, Properties params, String rawInput, Properties files )
{
  if(uri.startsWith("/files")){
    return serveFile( uri, header, myRootDir, true );
  }else{
    return serveJavascript( uri, header, myRootDir, true );
  }
}
```

*Figure 13 - Selection mechanism for web content or web services.*

The used mechanisms of asynchronous programming are commonly named as “callbacks”, because the function is invoked by the executed method at some semantically important time, with an asynchronously obtained result, that is literally called back. In this case, the result of the REST request is returned asynchronously.
To implement this type of functions, in Cordova it’s necessary to extend the “CordovaPlugin” class on the Java side, defining the API methods to be invoked on the TypeScript side. Besides that, it is also necessary to register the same API functions on a JavaScript file of the plugin and export it as a module. This way, the names of the API functions are recognized and can be used on both sides.

The webserver used already comes with defined methods for starting and stopping the server. However, to make it possible to send the request to the TypeScript layer, two more API methods were defined: “onRequest()”, showed in Figure 14, and “sendResponse()”. The first one, invoked on the TypeScript side, brings as a parameter the callback function that constantly traverses all the positions of the vector mentioned in step 1, and test if the parameters on the received JSON (created on the Java side with the request information) correspond to the tested endpoint. More precisely, it tests if the URI and the HTTP method correspond to some endpoint function defined on the library of endpoints. If so, the corresponding function is invoked;

```javascript
cordova.plugins.CoreHttpd.onRequest(
  function(request) {
    // apiProvider.array_endpoint_receiver is a vector that contains the
    // list of endpoints that the developer programmed
    for(let entry of apiProvider.array_endpoint_receiver) {
      // path is the URI, method is the HTTP method
      if(entry.path == request.path) & (entry.method == request.method)) {
        entry.function(apiProvider, request);
      }
    }
  }
);
```

*Figure 14 - Representation of function onRequest() that compares the request with the library of REST endpoints.*

The second API method implemented, represented in Figure 18 and also invoked on the TypeScript side, is responsible for sending an HTTP message with the result towards the JAVA side. It is worth noting that this HTTP message can be whatever the developer wishes to send back, according to his app logic, developed by him.

On the Java side, the mentioned function “serveJavascript()”, receives the result, and the thread that was sleeping wakes up. Therefore, it iterates the JSON message received from the TypeScript layer, and converts it to an output stream, sending it through the webserver to the final client.
3.7 Use of MobiRest

In order to use the MobiRest system the developer must follow a list of steps to integrate the system with their mobile apps in a hybrid environment using the technologies of Cordova, Ionic, Angular and TypeScript/JavaScript. The list is accompanied by several images that are excerpts of code demonstrating how to make some of the steps.

3.7.1 Development Environment Configuration

1. It is assumed that developers have already properly installed Cordova and Ionic (as well as its requirements);

2. Download the code of MobiRest.
   Type ionic cordova plugin add https://github.com/daniel-ist/MobiRest.git

3.7.2 Project Configuration

To integrate the MobiRest system, developers need to follow the instructions below:

1. Import MobiRest in your Ionic project (Ionic 2 or 3 or higher).
   In file app.components.ts declare the cordova variable by typing, after the imports section:

   ```
   declare var cordova: any;
   ```

2. Then, inside the method platform.ready().then() type the line:

   ```
   CorHttpd = (cordova & cordova.plugins & cordova.plugins.CorHttpd) ?
   cordova.plugins.CorHttpd : null;
   ```

   As shown in Figure 15.
3. Generate a new provider that will handle the requests and communicate with the server on the backend. Type on the command-line tool:

```
ionic generate provider ServerApiProvider
```

4. Replace the file generated on providers folder (the name of the folder should be `server-api`) by the file with the same name that is on the plugin folder that was added to the project.

### 3.7.3 Endpoints Registration

1. `MobiRest` is now able to receive new REST endpoints that will be programmed on the "library.ts" file, like the example available.

2. To add the endpoints to the server, call the API function:

```
addRoute(param1, param2, param3);
```

Where the parameters are the following:

- Param1 corresponds to the URI that developers intend to associate to the endpoint, on string format.
- Param2 is the HTTP verb that developers intend to assist (GET, POST, DELETE, etc). It must be on string format.
- Param3 corresponds to the method that developers want to call when the server receives the corresponding URI and HTTP method. Therefore, it is called from the `library.ts` file the corresponding method.

Figure 16 shows an example of implementation for point number 10.
3.7.4 Start the Server

1. The server must be enabled by calling the API function:

```java
this.serverApiProvider.enableServer();
```

Where `serverApiProvider` is an instance of the provider generated, instantiated on the constructor. It is demonstrated in Figure 17.

3.7.5 Program the endpoints

1. To generate a response, on the methods that developers programmed for the corresponding endpoints, the API function must be called:

```java
sendResponse(request.requestId, res);
```

Where the parameter `res` corresponds to the HTTP response that is desired and the parameter `request.requestId` is a necessary parameter containing the id of the request for the callback. An example is given in Figure 18.
Figure 18 - Example of HTTP response using the API function sendResponse().

Please note that for lower Android versions this system might not work, especially below versions 4.(*). In some other versions the WebView might not load properly, staying on a black screen while opening the application. In those cases installing a new WebView can make it work properly by adding the plugin CrossWalk [63].

```javascript
var res = {
  status: 200,
  headers: {
    'Content-Type': 'text/html',
    'Content-Length': '22',
  },
  body: "<b>Make a request!</b>"
}
cordova.plugins.CorHttpd.sendResponse(request.requestId, res);
```
Chapter 4

Evaluation

This chapter presents a discussion of the results of this project, along with a demonstration of its features.
4.1 Quantitative Evaluation

In order to analyze the efficiency in terms of performance and energy consumption described on the requirements, tests to evaluate these aspects were conducted.

4.1.1 Performance Tests

This section evaluates the time to process requests on MobiRest versus an external server.

Settings

All the tests were made using the tool Postman [64] on a laptop. This tool was chosen because it allows the making of collections of requests, and therefore the same request can be iterated as many times as users wish.

In Figure 19 A and B the settings for the tests made in MobiRest and in the external server are outlined.

A. Laptop - MobiRest

![Figure 19-A - Settings for the MobiRest performance tests.](image1)

B. Laptop - External Server

![Figure 20-B - Settings for the performance tests of the external server.](image2)
The same web services tested in *MobiRest* were hosted and tested in the external server Sigma03 of Instituto Superior Técnico [65].

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.

**Scenarios**

Multiple scenarios were built and *MobiRest* was tested on different REST endpoints. Following, are described the number of requests made per size of data sent or received by *MobiRest* and the 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 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.

In all of the tests of Figure 21 HTTP requests of type GET were made, in which both *MobiRest* and the external server retrieved 10 MB in total of the iterations. This means that on the first pack of requests, for example, 100 thousand packages each with 100 Bytes were retrieved, which corresponds to 10 MB in total. On the second pack of requests, 10 thousand requests for packages with 1 KB were made, which again amounts to the 10 MB sent by the external server and *MobiRest*, and so on. In Figure 22, instead of sending data, *MobiRest* and the external server received and read data (POST requests). The tests were done for the same number of requests and sizes as those of Figure 21.

**Results**

The obtained results are presented in Figure 21 for the case where both servers are sending 10 MB and in Figure 22 in the case of receiving 10 MB.
Figure 21 - Temporal differences between MobiRest and an external server (on sending data). Logarithmic scale.

Figure 22 - 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 100 thousand requests with small size packages were made, the performance is slightly lower, but still comparing with an external server, it is reasonable. The system only had an unreasonable performance when the packages are big (10 MB).
Besides that, when focusing on middle range 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 state that users will have good performance on the *MobiRest* system, since on the medium range the system works reasonably well.

### 4.1.2 Energy Consumption Tests

This section evaluates the energy consumption of the *MobiRest* system.

**Settings**

In Figure 23 the settings for the energy consumption of *MobiRest* are outlined.

![Figure 23 - Settings regarding energy consumption of MobiRest.](image)

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

**Scenarios**

To evaluate quantitatively this aspect, tests were made 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: a certain number of requests to the MobiRest server were made, 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 was the same as at the beginning of the tests. During the charge period, the smartphone was connected to a device that measures the energy flow (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.

Results

Figure 24 presents the results of the tests.

By analyzing Figure 24 it is possible to observe that with the smartphone with light at the 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 at the 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. Furthermore, it is worth noting that on a local network, generally, on regular use, an application doesn’t have to respond to so many requests (150 MB in total) on a time frame 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, the amount of energy that each request of 100 KB takes was calculated, 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.

## 4.2 Qualitative Evaluation

### 4.2.1 Demonstration app – “Travel Sync”

Additionally, this thesis proposes that a demonstration example of the application be made in Ionic and Cordova, which will be called “Travel Sync”, for sharing photographs in groups of people. It makes use of the MobiRest system to allow sharing media content within a local network without the need to access the Internet. With this app, users will have access to the camera of their devices and will be able to take photographs and share them with their friends. People establish new “friendship” on this app by scanning (with the camera) a QR code generated by their friend. All the data is stored on databases provided by the application on the device that is running the server.

Thus, people can, for example, be at a concert, where sometimes the 3G connection is weak, then connect to each other and share photographs. Another example would be a group of friends that is travelling and, in another country, share their photographs, without needing to spend mobile Internet data.

**External libraries of Travel Sync**

To implement all the features of the app logic module some external libraries (Cordova plugins) were required. These are described as follows:

**Cordova Plugin Camera:**

This plugin [66] is responsible for allowing the access to the device’s camera.

**Cordova Plugin SQLite:**

Using this plugin [67] the application can create tables and add entries on a local database.
Cordova Plugin Barcode Scanner:

This is the plugin [68] responsible for generating and scanning QR codes, required for the process of making new friendships over Travel Sync.

User Interfaces

When Travel Sync is loaded, it displays a welcome view, which has the option to sign up as shown in Figure 26.

![Figure 26 - Welcome view of Travel Sync.](image)

After selecting the “Sign Up” option, the view changes to the one presented in Figure 25. In here, users must register their information regarding name and username. By clicking on the “Sign up” button, the application stores the corresponding information of the user on a local database of the device.

The users of the group decide which member of the group will have the server enabled. Figure 27 presents the button responsible for calling the API function of the server that enables it. After clicking on the “Enable Server” button, the server will be enabled with the IP according to the network where the device is connected, and this user will now be the “master”, to whom the other users of the group make the requests when using the app.

Moreover, since Travel Sync is a mobile social network, users can add friends and connect to the server through scanning QR codes generated by the “master” user. Those features are represented in Figure 28, Figure 29 and in Figure 31.
In Figure 29 there's an example of Travel Sync scanning a QR code, and in Figure 28 an example of a QR code generated by Travel Sync app.
When *Travel Sync* scans a QR code (user A), it contains information about the user generator of the code (user B) and his IP address, which corresponds to the address of the server ("master" user - B). Then a POST request is sent to the server through the received IP address, with the information of the user A, more specifically his username and also a token received that was generated by the user B at the moment of the sign up. Then, the user B, that displayed the QR code, receives the request and, if the token coincides, it stores the information and both users become friends over *Travel Sync* social network.

Then, users can take photographs by clicking on the plus button and add descriptions to them. An example is shown in Figure 30. After publishing a new photograph and description, the application sends a POST request with the corresponding data to the server, which can be enabled on the current device, or in another device using the *Travel Sync* app. Then, the server will store information regarding said photograph, including the username of the person who took it.

Furthermore, the view changes and the users have, at this point, access to a feed that contains all the photographs taken by their friends on *Travel Sync*. This feed view is presented in Figure 32.
REST Endpoints

To fulfil the communication features above mentioned, the REST endpoints at the “library.ts” file of MobiRest, described on Table 2 were defined.

Table 2 - REST endpoints implemented in Travel Sync.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/photos</td>
<td>It reads the data regarding the photos of the user’s friends from the database and sends them to the client.</td>
</tr>
<tr>
<td>POST</td>
<td>/api/photos</td>
<td>It stores a photograph taken by the client on the database.</td>
</tr>
</tbody>
</table>
**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 the *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 photograph 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 3.4. These functionalities include:

- Register users and store the corresponding information such as username and password on the database of the application. This way this information is stored locally. Note that this information is necessary along with a token generated on the process of adding friends;
- Automatically add a marker with the name and description to the photographs.

**Integration of *MobiRest***

According to section 3.7, some steps were taken regarding the integration of *MobiRest* system.

First, the corresponding plugin of the system was added to the project and all the configuration settings were made: the dependencies were imported to the project, the required variables were declared and the server provider offered by the system was added to the project.

At this point, the project was ready for the endpoints’ registration: the logic functions corresponding to each web service (the ones mentioned on the previous section) were defined on the “library.ts” file. Finally, these functions were associated to the project through the method “addRoute()".
4.3 Discussion

The developed mobile application, *Travel Sync*, makes use of the *MobiRest* system and is a real-life example that can benefit of such a system. It also demonstrates that it’s possible to host and provide RESTful web services through smartphones with hybrid development. With *MobiRest*, the *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, as they aren’t as vulnerable. Moreover, in centralized systems, a “3G connection” is required to make the communication to the Cloud possible, so that users can access the stored data, or communicate with other users. Thus, using the *MobiRest* system becomes advantageous since users only need to connect via Wi-Fi to the local server, which can be done 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 large amount of energy. Aside from 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 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 also be a powerful tool for mobile applications, namely Social Networks in Proximity that, in more hostile environments such as concerts and other crowded events, may originate 3G connection losses, and consequently failure of communication on the typical client-server applications through the Cloud. Thus, the *MobiRest* system demonstrated being efficient and it simplifies all this process of mobile social networking in the *Travel Sync* app.
Chapter 5

Conclusions

This chapter makes a brief summary of all the previous chapters and concludes the final considerations about *MobiRest* system. At the end it mentions future work that can continue this work.
This thesis started by introducing in the first chapter the theme and by describing the objectives as well as the structure of the document. The main goal was to develop the *MobiRest* system, which 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 Chapter 2 some research was made regarding the existing mobile development frameworks and support for mobile web services. Also, the state of the art regarding mobile social networks was explored, namely “in proximity”. This research was important to know how other works solved this problem, and to detect the gaps missing. Other relevant subjects within the context of this thesis were explored such as Edge computing, explaining its relevance over this thesis, corresponding to a solution for decentralized systems.

In Chapter 3, the *MobiRest* solution was presented and the architecture of mobile applications with this system embedded was described, including an explanation of each module of the system. This chapter also specified all the requirements that *MobiRest* should meet, as well as its internal architecture and an execution flow explanation. At the end of the chapter a simple list of steps that developers must follow in order to use *MobiRest* and integrate it in their apps was presented.

Afterwards, in Chapter 4 an evaluation and demonstration of the *MobiRest* system was made and the results were discussed. On the evaluation tests, *MobiRest* demonstrated having a generally good performance, especially in middle sized packages of data. Furthermore, in terms of battery life it was observed that the consumption of energy doesn’t represent a huge constraint comparatively to other types of usage on smartphones.

*MobiRest* is a powerful and innovative tool that takes a novel approach in solving the necessity of decentralized systems, easy to integrate, and easy to program and configure, thus offering developers new features for their apps by provisioning efficiently RESTful web services. Furthermore, it presents new possibilities for the clients of applications with *MobiRest*. 
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 thesis for the Android Operating System. Moreover, this extension can take advantage of the existing server on the mentioned plugin, Cordova-Httpd [56], used throughout this work, and follow the steps deeper 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.

References


