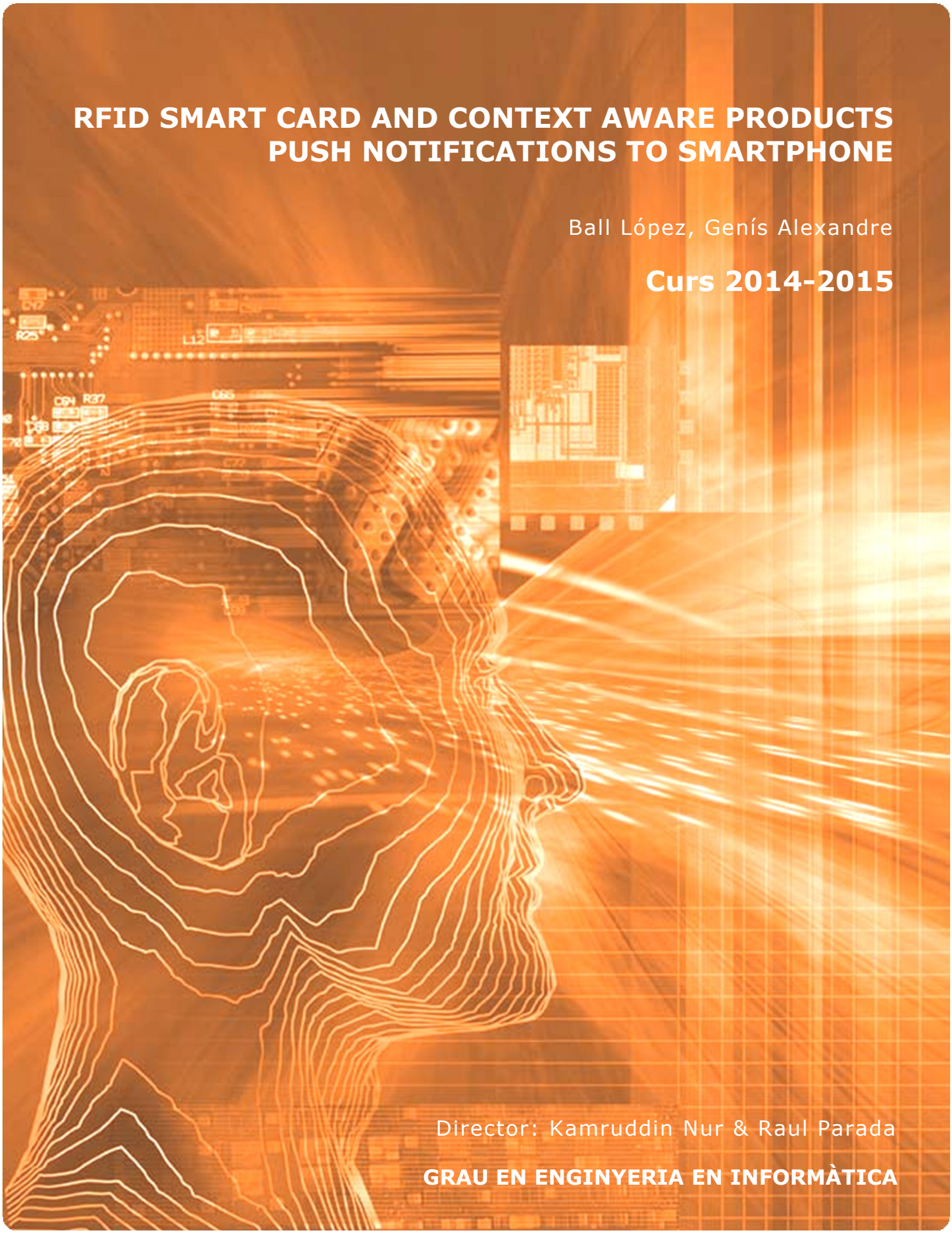
RFID SMART CARD AND CONTEXT AWARE PRODUCTS PUSH NOTIFICATIONS TO SMARTPHONE

Ball López, Genís Alexandre

Curs 2014-2015



Director: Kamruddin Nur & Raul Parada

GRAU EN ENGINYERIA EN INFORMÀTICA

upf. Universitat Pompeu Fabra Barcelona
Escola Superior Politècnica

Treball de Fi de Grau

RFID SMART CARD AND CONTEXT AWARE PRODUCTS PUSH NOTIFICATIONS TO SMARTPHONE

Genís Alexandre Ball López

FINAL GRADE PROJECT

Kamruddin Nur and Raúl Parada

Department Information Technology and Communications

UPF

Abstract

The objective of the project is developing a context aware mobile application that will receive push notifications of promotions once a user is near a store, with the help of RFID inventory, WIFI technology, a database and a push notification server. When a shopper enters in a retail store, if the shopper is carrying an RFID shopping card in his/her pocket or wallet, the RFID system will detect that the shopper is within the store and will send push notifications of products in his/her Android smartphone. Using RFID technology shoppers can receive contextual timely information of the retail products with useful information such as product promotions, photos, discounts, etc. With this project we can improve the overall shopping experience, targeting different promotional notifications to specific users, while being able to differentiate the users that will receive each different notifications with specific filters like context, user preferences, products, etc.

Table of contents

List of figures

List of tables

1. INTRODUCTION

1.1 Overview

In this project, with the help of the RFID technology, we have designed a system that is able to send push notifications of targeted promotions to shoppers once they enter a store. The system includes an exclusively implemented Android application that will receive and push notifications of targeted promotions to the smartphone of the user. To continue, the system also includes a server that selects and sends the targeted promotions to each user that is using the Android application, a database that stores all the data needed for the system to work properly (user data, accounts, promotions etc.) and finally it includes a specific store gates design that uses the RFID inventory to be able to detect the users near a store with the help of a RFID tag that will be attached to shopping cards of each user. The users will be able to carry their shopping cards in their wallet or pocket.

1.2 Motivation

We started this project because we found that the promotions on many stores are too generic and a lot of times they are not even seen to all the users because of the inability to see them organized in a concrete place. Because of that, most customers end up buying products do not really need and miss promotions that would have been attractive to them and this ends up resulting into bad experience in general. We thought that it would be a great idea to develop a system to be able to get targeted promotions in the time they are needed, to increase awareness and avoid being missed. We have also decided to use the RFID technology for this project because it is a technology that is very easy to implement since the only restriction we have by using it is having to use the hardware needed to implement the technology. Many other technologies instead, have a lot more requirements. For example, the NFC technology has many other dependencies like requiring a compatible device, having to place your device into a scanner etc. With the RFID technology the user should be able to get the promotions with only having to carry a shopping card and a device with them, and that is something many customers usually do when they are going for shopping anyway.

1.3 The objective

The main goal of this project is to send customized push notifications directly to the shoppers smartphones by taking the advantage of Radio Frequency Identification (RFID) technology.
The Radio-Frequency Identification (RFID) is a technology that uses radio waves to read and identify information send by a tag attached to an object. The signal of these tags can reach several meters without being in a direct line-of-sight from the reader.

1.3.1 The goal of the project

One of the goals for the project is to be able to send targeted promotions to a customer using our system with the RFID technology. We also have the goal of developing an Android application compatible as much devices as possible, to communicate with the

server and be able to receive the targeted promotion. We also want the whole system to be as much efficient as possible because of the high concurrency we can expect from this type of implementations. Finally one of the most important goals that we have is being able to send the promotions to all the users just in time (when they are crossing the store gates).

1.3.2 The project Scope

The scope of the project is to be able to design and implement a complete system where a user is able to receive push notifications of promotions from the server using an Android device and the RFID technology before the deadline of six months.

1.3.3 Project sponsors

Our sponsors for the project are:

UbicaLab

1.3.4 Schedule

The schedule followed to implement the project is:

Millestones	Start	End
Read and identify tags from AdvantNet module using a java implementation.	9/1/2015	14/1/2015
Test the implementation at UbiCA lab laboratory.	15/1/2015	19/1/2015
Design and implement a MySQL database that links tags with users.	9/1/2015	14/1/2015
Implement a simple Android interface.	2/2/2015	6/2/2015
Correct errors from the tests in the UbiCA lab laboratory.	20/1/2015	23/1/2015
Test the corrections in the UbiCA lab laboratory.	26/1/2015	28/1/2015
Design the database part for managing preferences and implement an algorithm to retrieve them.	29/1/2015	5/2/2015
Add to the java code an implementation to better manipulate the database and an implementation of the preferences algorithm.	6/2/2015	13/2/2015
Build external server and required connections in java	16/2/2015	2/3/2015
Build notification functionality in the Android application and the server using the GCM API.	3/3/2015	16/3/2015
Test GCM functionality with registered devices	17/3/2015	24/3/2015
Implement own push notification service without the GCM dependency.	24/3/2015	3/4/2015
Implement connection persistence services and improve the server resources.	9/4/2015	15/4/2015
Implement different notifications for different devices also implement account system to register and login accounts	16/4/2015	28/4/2015
Finish main the parts of the project: Showing the promotion pictures at the app and make RFID work locally simulating a generated XML from AdvantNet.	29/4/2015	13/5/2015
System working in a simulated environment with RFID tags and adding extra functionalities for the app (less screens and info below the promotions).	14/5/2015	9/6/2015
Test all the functionalities at the UbiCA lab laboratory simulating a real world scenario and gather data of the tests	10/6/2015	15/6/2015

Table 1. Schedule

1.3.5 Project risks and costs

The projects risks and costs are filled in the table below:

Project implementations	Costs	Risks
RFID functionalities	Low	Low
Android interface	Low	Low
Server connection management with multiple devices	Medium	Medium
Android connection management with server	High	Medium
Push notification functionality	Medium	High
Database functionalities (accounts, purchases, promotions...)	Medium	Low

Table 2. Project risks and costs

As we can observe, the project has high risks with the push notification implementations, we need them to work in real time because the project is time dependent and we need to be able to send the notifications to multiple devices at once. Since the number of requirements in this part is really high we will probably run with problems using implemented services.

The project also has moderate risks with the android application and its connection with the server, we need to implement an App compatible with multiples devices and we do not have many resources to test that.

To continue, the project also has risks with the server connection management because we need to do a really efficient implementation since the server will need to be able to manage multiple device connections at the same time and as mentioned before the push notifications need to be performed in real time and the server performance cannot be affected by having to deal with multiple connections.

Finally, The implementation of the Android application, the connection management of both sides and the push notification service will be the most demanding steps of the project. The Android application has to have numerous functionalities and has to be able to communicate with the server and the push source. This will demand a lot of effort to implement it and we will have to focus on planning really well the design of the architecture before doing anything. The push notification system also requires adapting the whole system to it because the server and the Android application will need to use them and that can imply a lot of coding, modifications and dependencies.

2. THE STATE-OF-THE-ART

2.1 The RFID technology

The RFID (Radio Frequency Identification) is a technology that is being used more and more the last few years. The technology opens a way to identify and locate objects indoor using a RFID passive tag attached to it that is able to passively (without any source of energy) emit a signal that is easily recognized by an antenna (there are also active tags but we will not describe them since they are outside our Topic). Nowadays, a system with RFID technology is able to detect multiple Tags at the same time without collision conflicts since every tag has its own ID and the system can read sequentially different tags using an anti-collision algorithm.

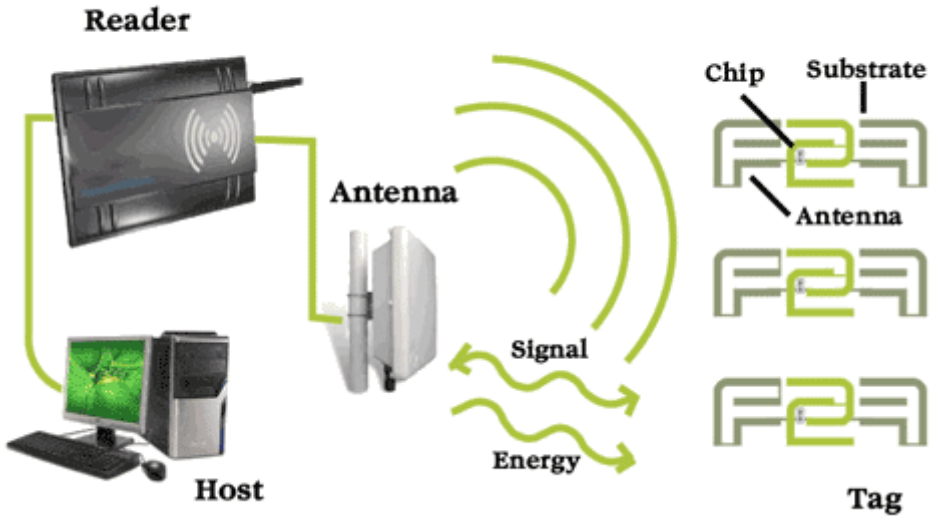


Figure 1. General RFID system

2.2 Areas of application of the RFID technology and its improvements

The RFID technology is being applied in a lot of areas since is a good tool to simplify and improve communications and processes. One of those areas is shopping. In the shopping context the last few years there has been a relatively increase of the amount of technologies that use RFID to improve the overall experience of both shoppers and vendors, but since it's a relatively new technology, there still are a lot of processes that could be improve using the RFID technology on that context. One of them is to improve the process of finding products as a shopper that are both in our needs and as less expensive as possible.

With the use of RFID technology the vendors are now able to improve the experience of customers. Now shoppers are for example able to use shopping cards as an interactive tool to obtain information of the items carried by the cart[5] and the vendors are for

example able to better control their stock[6] or control the processing and supply of products[1].

2.3 The RFID technology in the project

With this project, we want to go further using a common device that almost any shopper has, the Smartphones. Some studies, reveal that the integration of Smartphones and RFID technologies have a lot of market potential and this integration is being applied on some sectors already[3]. There are even some technologies being studied and applied on the shopping context that use smartphones and RFID[2,4] but those only contemplate the possibility of showing details of products the shopper is viewing.

On this project, again, we are aiming to go further and take more profit of the capabilities of the integration of RFID with Smartphones by developing an application that will be able to display not only information about different products but also recommendations based on the historical data of the shopper.

With that way, the vendors will be able to provide the clients with a system that will automatically increase the knowledge of their stock products and will also be capable to easily promote and relate products with shoppers using their data and gather statistics that could easily help them predict promotion results and create better promotions. With this system also, the clients will be able to see the promotions that are interesting for them skipping the ones that aren't related to their needs.

With our system we can improve the overall shopping experience to the both parts of the shopping sector (vendors and clients), using a relatively cheap technology and taking profit of the big impact of Smartphones in our society.

3. METHODOLOGY

To implement the project, we have prioritized the implementation of the Android application, his connection with the server and the push notification functionality. Because of that, we have spent almost half of the total time only with it.
To begin with, we have taken the first month for implementing the basic functionalities to communicate with the RFID module and the MySQL database with our initial java server.
At the start of the second month we began with the implementation of the Android application with an initial simple interface implementation. During this second month we also started building the java server itself to be able to start working with preferences. During the second half of that month we also started building the communications between the server and the Android application.

To continue, during the third month we have implemented the GCM connections from the server and the Android application to be able to send and receive notifications. At the end of this same month after some testing we had to discard the GCM implementation for our system.

Coming up next, during the fourth month we have built our own push notification service after some researches on how other famous applications deal with this and designing our own way to push notifications. During this month we have also designed a way to handle the connection persistence requirement and implemented an account system for the Android application.

Finally, the last two month we finished the main parts of the system making all the functionalities of the server work completely with the RFID hardware, improved the code, corrected errors, improved the compatibility of the Android application, added some extra functionalities and we finally did during the last few weeks, some tests in the lab with one of those simulating a real world environment.

4. THE PROPOSED SYSTEM

4.1 System overview

To start with, since in this project we aimed to implement a system that has to work in a very specific time frame because we want that our users receive push notifications of promotions **while entering** to a store, the whole system design is based on that critical restriction.

The system is composed by five major modules:

- A push notification server
- A android app
- An AdvaNet software
- A MySQL database
- A RFID hardware

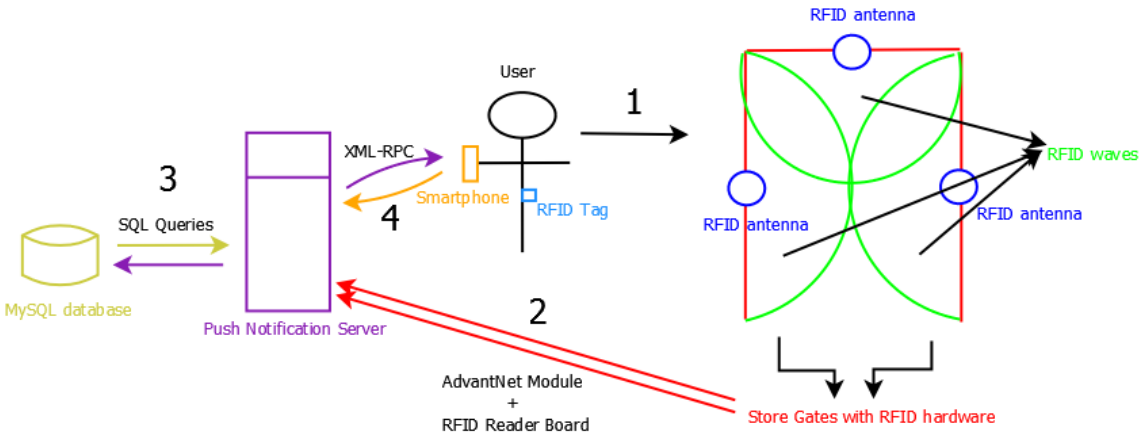


Figure 2. General Architecture

The push notification server is the module that sends all the data required to the users. That is, the push notifications and all the data that the app needs to load every promotion like the pictures, the information about the promos, etc.
Also, the push notification server is the module in charge of receiving and sending all the requests and all the data required for the main functionalities of the client app to work, that means that this module is the one that makes the login and register petitions to work in conjunction with the MySQL database and this means that is the one also in charge of registering most of the data required to the database.

The android app is the module that will be given to the clients of a store. With this module the users will be able to interact with all the functionalities of the system. The application is in charge of working in conjunction with the push notification server module by sending and receiving all the request and data required for the main functionalities to work. Also is in charge of showing all the required feedback to the

users of the system and must be therefore clear, simple and easy to use and interact with.

The AdvantNet software proprietary of the Keonn Technologies is the module that is in charge of all the RFID dependencies of the system. By communicating with this module the push notification server is able to retrieve all the users that are entering the store. This works because the users will be given a shopping card with a specific RFID tag attached to it. With this RFID tag the system will be able to identify each different user, for later targeting specific notifications to them, those, as we will see later on, are based on their resent purchases. This module to summarize is the one that we will use in conjunction with all the RFID hardware, to scan and read all the RFID tags that are near.

The MySQL database is the module that stores all the data and its inter-dependencies required for the whole system to work properly, that is, the login information, the user data, the products, the promotions, etc. Everything is stored in the database and through specific queries the push notification server is able to retrieve all this information each time is needed.

Finally, the RFID hardware is the module composed by all the hardware needed to make the whole RFID communications needed to be able to scan and detect the tags that are near the entrance of the store. The hardware needed is:

- RFID antenna (we used the model 'ADAN-p12EU-FL-200' in our experiments)
-

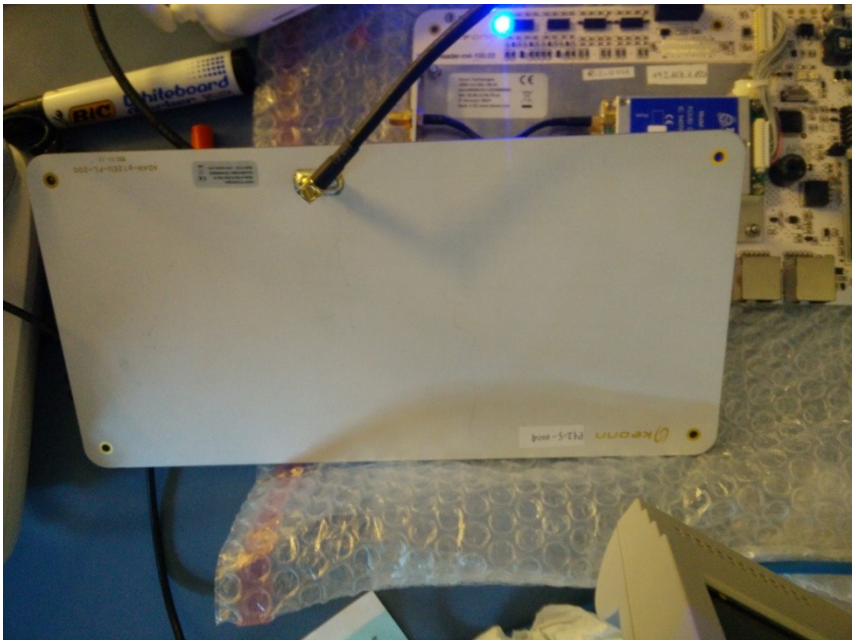


Figure 3. RFID Antenna

- RFID reader (we used a model M6e-a in our experiments)

Figure 4. RFID Reader

- RFID tags (we used different formats in our testing)

Figure 5. RFID Tags

- RFID cables
- Ethernet cable

4.2 Hardware

The minimum hardware needed to make the system working, consist on the RFID hardware, a server machine, a database server, a router and an android device. In the next diagram we can see a how they link together:

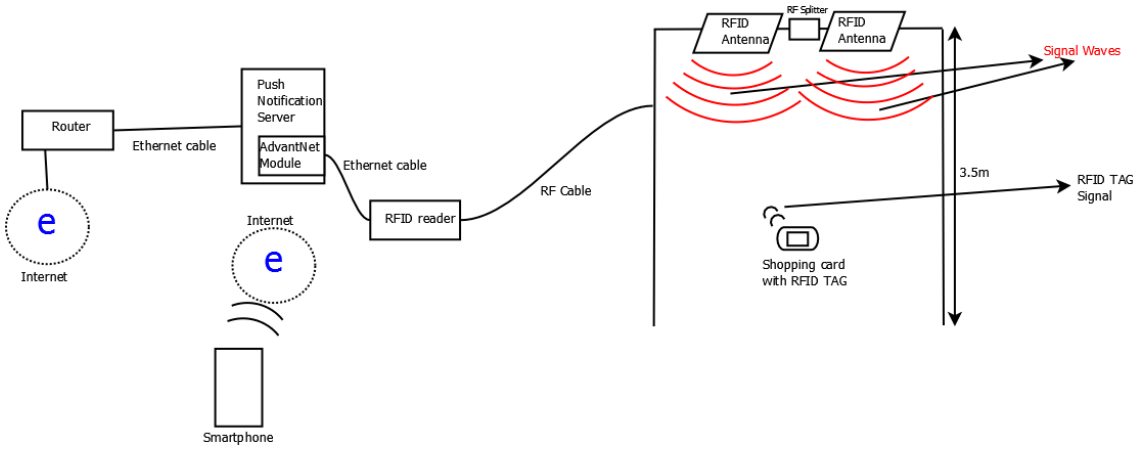


Figure 6. Hardware

As seen in the diagram, the store gates have to have a specific design in order to make sure that the RFID system is able to detect the users properly. In order to detect the tags that are crossing the gates properly in this specific system, the antennas have to be installed at the top of the gates at a height of no more than 3.5m. With that design as we will see in the "Experimental results" section, the antennas are able to detect the tags attached to the shopping cards that are inside a wallet and the wallet inside a pocket of the user, when they are near, with a fair margin even in the power output set for the antennas.

4.3 Software architecture

4.3.1 Architecture design

General Architecture:

The general workflow and communications of the modules that compose the whole system is as follows:

1. A user carrying a shopping card with a RFID tag attached enters a store.
2. The epc[1] of his RFID tag (that will be attached to his shopping card) is detected by the RFID hardware.
3. The AdvanNet module that is running in the server and continuously scanning for new tags detected by the antennas, retrieve the tag for the server.
4. The server queries the database to identify the user of the tag and all the data[2] needed to generate the targeted promotions.
5. The server communicates with the Android application installed in the Smartphone of the user, sending all the data required for the application to work properly and be able to do its functionalities.
6. The Android application parses the data received, loads and displays the promotions to the screen with a specific picture of each promotion and generates a push notification for each promotion.

On the step 4, we take only the purchases of the recent months because the preferences of a user are mutable and that means that those can be changing from time to time.

For this system to work properly it is **crucial** that the types of products that are identifiable for the server (and stored in the database) are well designed, that means that the categories of products have to be not too much specific while also being not too much generic. Also, we store the date of the last time we have sent promotions to the user because we have introduced a restriction (that can be modified) to the server for only sending promotions if the user doesn't have received new promotions in the last **24 hours**, we will talk later on about this restriction and why we made it.

[1] Epc is a unique number that is send through the RFID TAG all the time passively.

[2] Data includes the products that the user has bought the past six months, the promotions that are running and are compatible with the preferences of the user and the last date that the user has received new promotions.

Push notification server architecture:

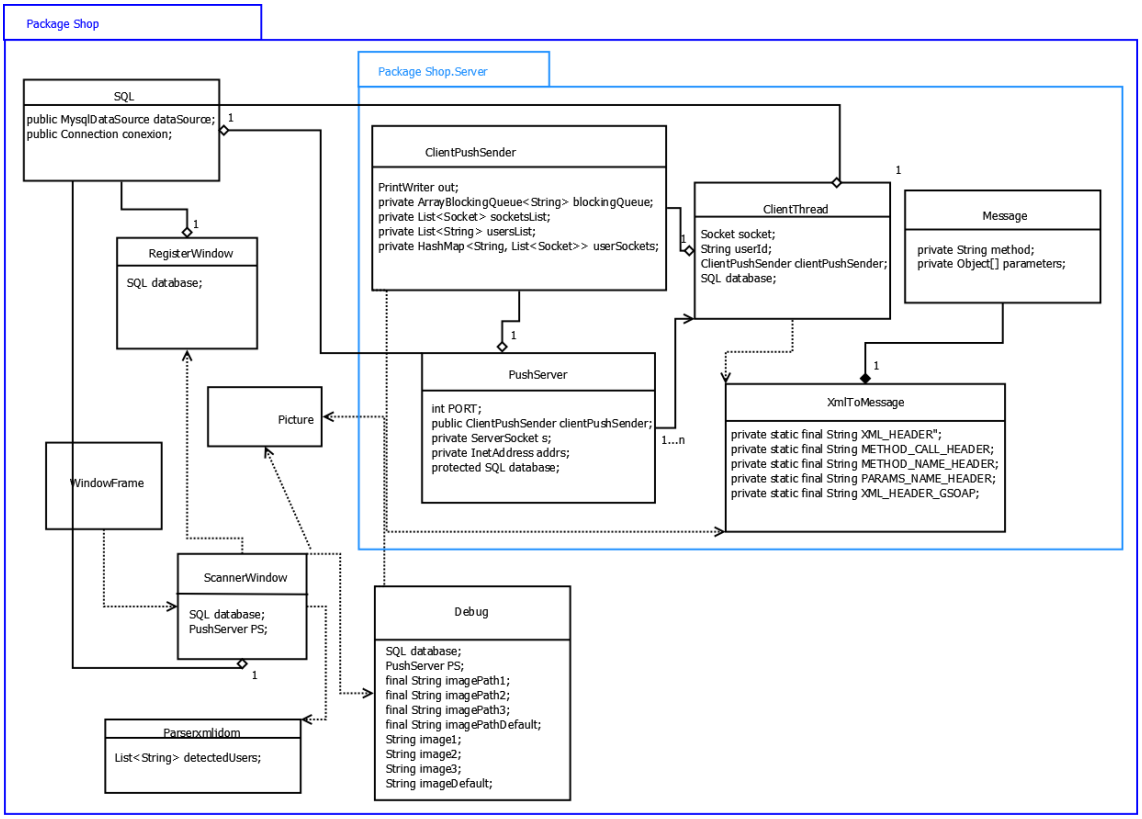


Figure 7. Server Architecture

The push notification server is the central and most important module of the whole system because of that, it's the only module that communicates with all the others and acts as the central service. Because of that, the push notification server is composed by all the different functionalities needed for being able to interpret all the data in all the different formats that are used by the other modules.

The push notification server is able to communicate with the MySQL database using the SQL format for its communication with that module, is able to communicate with the Android application module using TCP sockets as the data transfer protocol and XML-RPC as the format of the data and is finally able to retrieve the data detected by the RFID module using HTTP as the data transfer protocol and XML as the format of the data. Coming up next, we will be analyzing all these formats and protocols while describing the main parts of the server.

To start with, the server modules are divided in two packages, the main package named "Shop" and a package inside that first, named "Server". The package "server" contains all the modules related with the communications with the android application, that is, all the needed functionalities to be able to send notifications, promotions, pictures etc. and to be able to receive login petitions, account registering petitions etc, from the Android application. However, the main package "Shop" contains the required functionalities, to be able to manipulate the server, like different graphical screens and also contains the required functionalities to be able to communicate with the MySQL database and to be

able to retrieve the data gathered by the AdvantNet module. Finally, the package "Shop" also contains the main algorithm that decides when to use each of the whole functionalities inside the "scannerWindow" class that refers to the main screen of the graphical functionalities.

As for the code as a whole, the push notification server has a total of 4 JFrame java classes that represent the 4 different graphical screens that an administrator will be able to manage in the server. The classes mentioned are:

- WindowFrame
- RegisterWindow
- ScannerWindow
- Debug

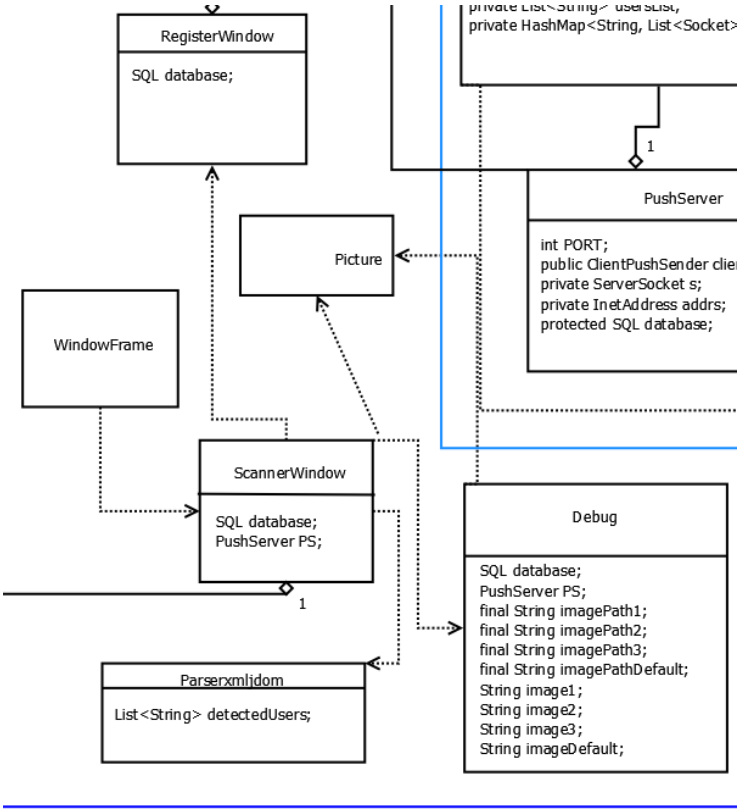


Figure 8. Class diagram for screen displaying for the push notification server (cut fragment from original diagram)

The "WindowFrame" class is the first graphical screen that is displayed when launching the push notification server and because of that contains the main method.

To follow with the components in the code of the push notification server, to be able to communicate with the other modules the code contains some specific java classes that provides the needed functionalities to adapt, parse or transform the data to different formats to establish the different communications, these classes are:

- Parserxmljdom
- Picture

- SQL
- XmlToMessage
- Message

To start with, the "Parserxmljdom" provides the needed functionalities for the pushnotification server to parse the data received from the AdvantNet module. Specifically, transform the XML data generated in a file by the AdvantNet module that contains the detected RFID tags. This XML file is updated in real-time providing only the RFID tags that are detected at one specific time frame and removing them once they stop being detected.

The "Picture" class however, contains the needed functionalities to be able to retrieve a local image file and parse it into a String with a base64 format. With this the push notification server is able to fill the picture of a promotion into a formatted XML file that will later send to the Android Application.

Furthermore, the class "Message" is the class that contains the needed functionalities to be able to send any type of data to the Android application. With this class the push notification server is able to store any type of data into the "Message" object instantiated using this class as a type and after it is able to transform all that data stored into a String type that follows the XML-RPC format, using the "generateXMLMessage()" method included in the Message class itself.
Although if the Message object, should be able to store any type of data on it, we decided to transform previously all the data that we wish to store to the message into a String, to be able to better control the String codification of that data, this is why we for example, transform the image files into a base64 String, even when we should be able to store that data into the Message object without having necessarily to codify it at all into a String previously.
On the contrary, the "XmlToMessage" is the class that provides the functionalities needed to transform a String that follows the XML-RPC format into an instance of a "Message" object. With this class, alongside with the functionalities of the "Message" class, the server is able to work with the data received from the Android application.

To finalize with the adapter classes, the "SQL" class is the one that provides the needed functionalities to communicate with the server of the MySQL database. This class uses the Java API named "MySQL JAVA Connector" and provided by MySQL, adapting the basic methods to do queries to the MySQL database and to insert and update the tables in it. With this class, the code needed to operate with the database gets a lot shorter and a lot more readable, making the internal code of the push notifications server a lot more compact.

To finalize with the push notification server's code, we left to the end some of the classes that are inside the "Shop.Server" package. As we mentioned before, the classes inside this package are the ones that provides all the functionalities for the push notifications server to be able to **send** data to the Android application.

Specifically, the classes that we left to the end and that are inside the mentioned package are:

- ClientPushSender
- ClientThread
- PushServer

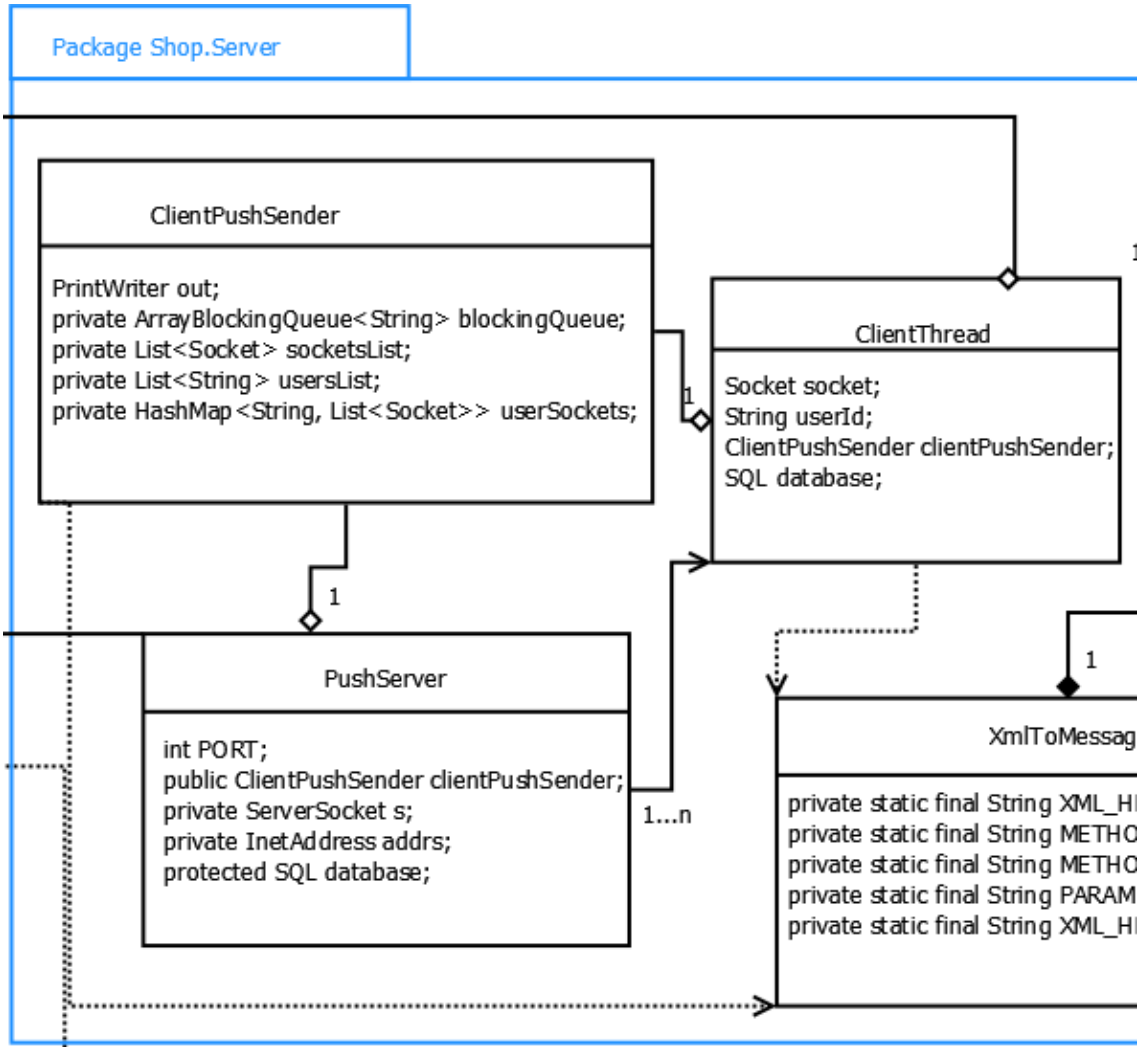


Figure 9. Class diagram of connection management module in the push notification server (cut fragment from original diagram)

We left the description of these classes for the end of the section, because they are the ones that we find that are the most important. That is because they are the ones that provide the functionalities to be able to link, start and manage the communications between the push notification server and the android application that is installed in the device of each user. With this, the push notification server is able to transmit and receive the data that we have previously formatted using the classes described before ("Picture", "Message" and "XmlToMessage").

Particularly, the "PushServer" class provides the functionalities to be able to listen for connect petitions from the Android application to the push notification server. This class creates a "ServerSocket" connection that will be listening to a determinate address and

port. Those address and port will be the ones that the Android application will be looking for when trying to establish a connection to the server.

Additionally, the "ClientThread" is the class that provides all the functionalities needed to handle and manage all of each possible different petitions of a determinate user (login petition or register petition). An instance of the "ClientThread" is created each time a new user connects to the push notification server, this is due to the need of having to deal with the concurrency of multiple users contacting with the server in the same time frame. Each time this class is instantiated and started, it creates a new thread and that will be looking only for petitions to the connection (socket) of the specific user that has started the communication. Each time the "PushServer" class receives a new petition, creates a new socket to handle the petition and start a communication link between the server and the user (with the Android application) and starts a new instance of the "ClientThread" class with the created socket as a parameter. Thanks to this, **only one** "ClientThread" instance will be dealing with **only one** of the total connections.

To end with the section, the last class that we have to describe is the "ClientPushSender". This class is the one that contains the functionalities to be able to send any data to any of the sockets that are running in the push notification server. This class also contains a list of all the open sockets that are running in the server. Is the responsibility of the instances of "ClientThread" to manage the list of sockets contained in this class. Each time a "ClientThread" instance receives a petition from the socket to start a communication it updates the list of sockets contained to the "ClientPushServer" adding the socket to it and each time a "ClientThread" instance receives a petition to end a communication from the socket it updates the list removing that socket. Additionally, only one instance of the "ClientPushServer" class is running in the client push server and is created in the "PushServer" class the moment that the "ServerSocket" is started.

The "ClientPushSender" class also has a "hashmap" that maps all the sockets with the ID in the database of each user that has started the connection that is being maintained with each socket. This "hashmap" is also managed by each "ClientThread" instance. Because of that, for the communications with the push notification server, it is mandatory that the Android application sends the ID of the user that is using that application. In the next section we will describe how we manage this in the Android Application side.

Finally, with the "ClientPushSender" class the push notification server is able to send data to each client by only passing as arguments the data that is needed to send and the user ID in the database of the user we wish to send the data in his device.

Android App Architecture:

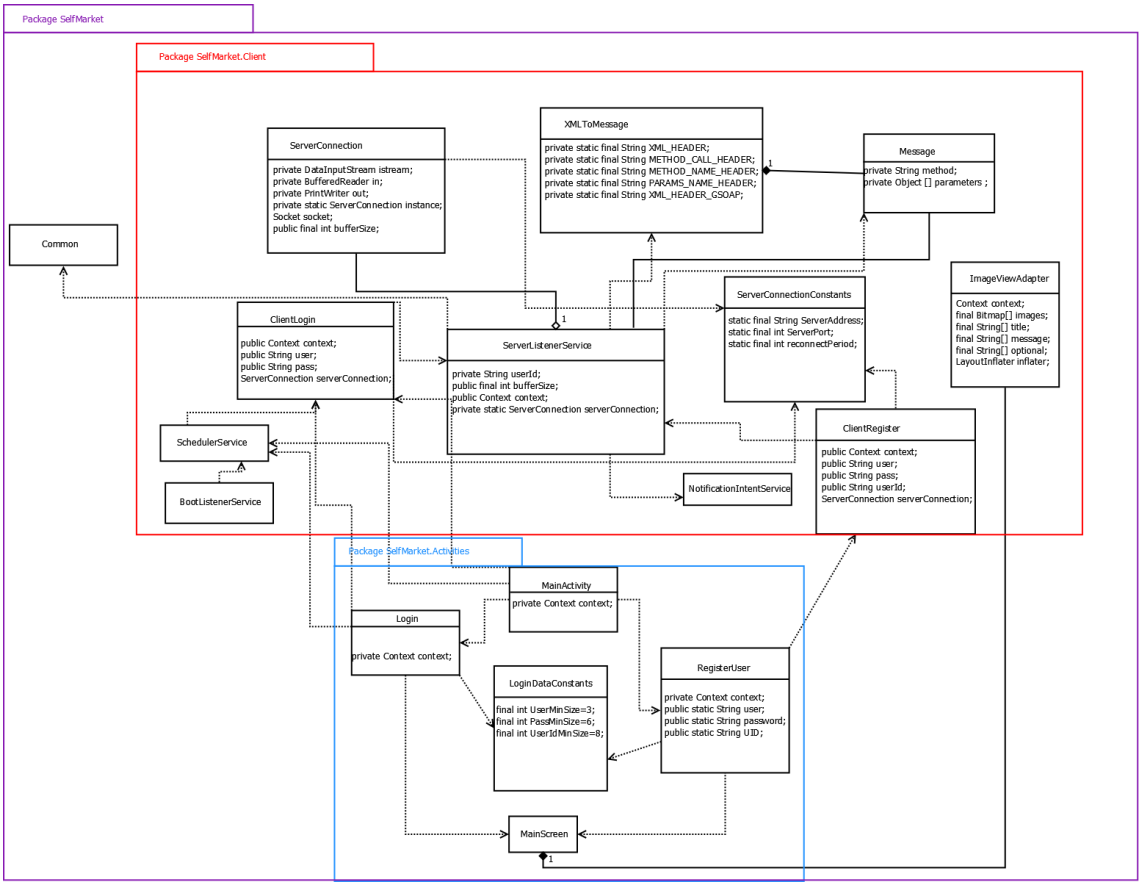


Figure 10. Android Application Architecture

The Android application is the module that will be given to the users and because of that, it's compatibility with many multiple devices and it's interface are very important, also, we want to take a look to the resources required for the application to run and how much it increases the battery drain of any device.

The Android applications depends entirely of the server and because of that we want also to make sure that once the application has established a connection for the first time, it is able to recover this connection every time. Also, knowing that a device now a days is able to connect to Internet in many different ways (3G, LTE, WiFi, etc.) we want the service to be able to recover from a connection lost because it can occur that a device changes its internet connection for example from LTE to WiFi if the store has a free WiFi etc. and since the application will be running in a portable device we have to expect this connection loses to occur. We have to remember that our system is time dependent as we mentioned earlier and because of that we need to recover the connection to the server as early as possible to guarantee that the system will work in the time frame that we have.

Also, since we will need to indentify the users, we have provided them an account system and to facilitate the things we want that once an account has been registered and the user has logged in with that account, to remember the account credentials to avoid having to ask the user for the account credential every time the application is started.

Finally, with the implementation of the Android application we have also taken into account the possible interrupts that an Android device could have when the application is running.

In short, as discussed below, the application is able to recover from a device shutdown, reboot, app kill, connection lose, etc. without any problems, maintaining the service always active in background and without wasting almost any resource.

Starting with the code, we find that all the classes are divided in two different packages. The first one is the package "SelfMarket.Client" and the second one is the package "SelfMarket.Activities". The first package contains all the classes that provide the functionalities needed to communicate with the server and act as a server client, while the second package contains all the classes that implement an Android Activity, that is the classes that represent a graphical screen in the application.

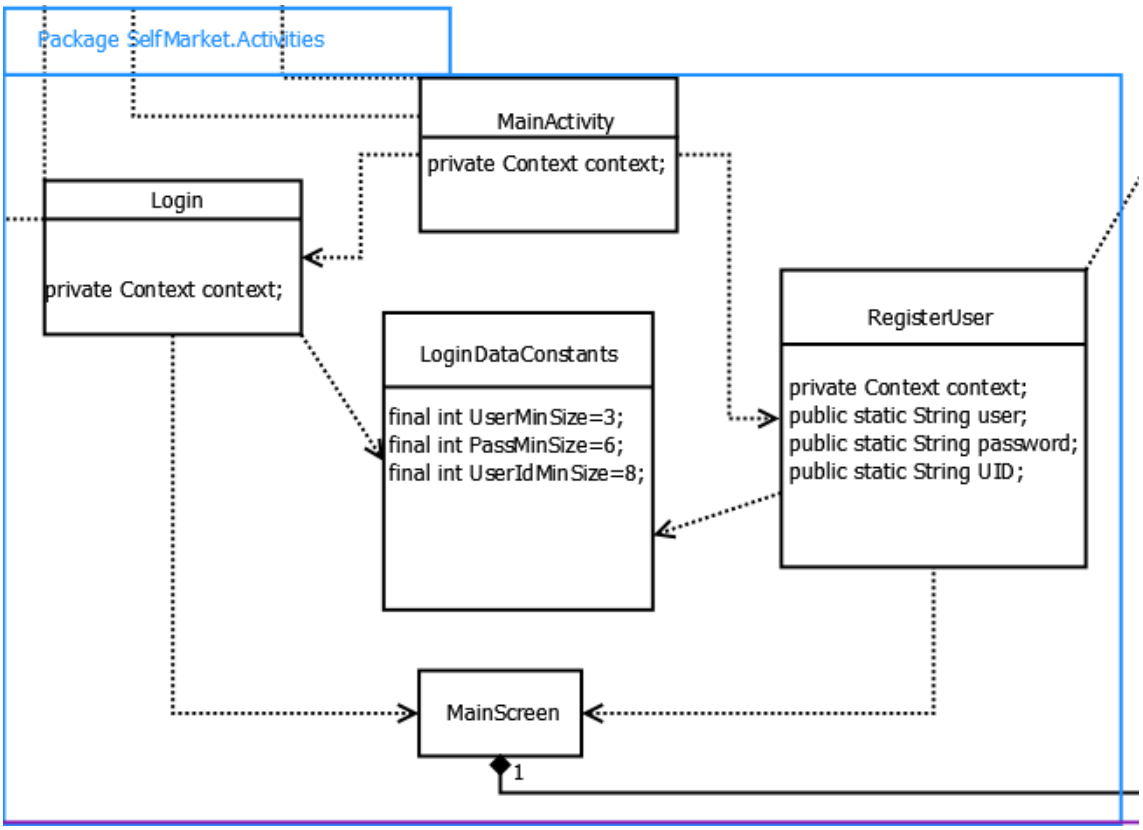


Figure 11. Package Activities from the Android app code (cut fragment from original diagram)

Starting with the "SelfMarket.Activities" package that is zoomed above, we encounter four classes and one interface. The classes are:

- Login
- MainActivity
- MainScreen
- RegisterUser

The Interface "LoginDataConstants" as his name seems to tell, it's an interface that provides an easy access to some constants we use for the login or register functionalities.

The "MainActivity" class corresponds to the first activity that starts when launching the app. The "MainScreen" however corresponds to the screen where the user will be able to see the promotions received and is named is like this because is the screen where the main functionality of the application will be displayed.

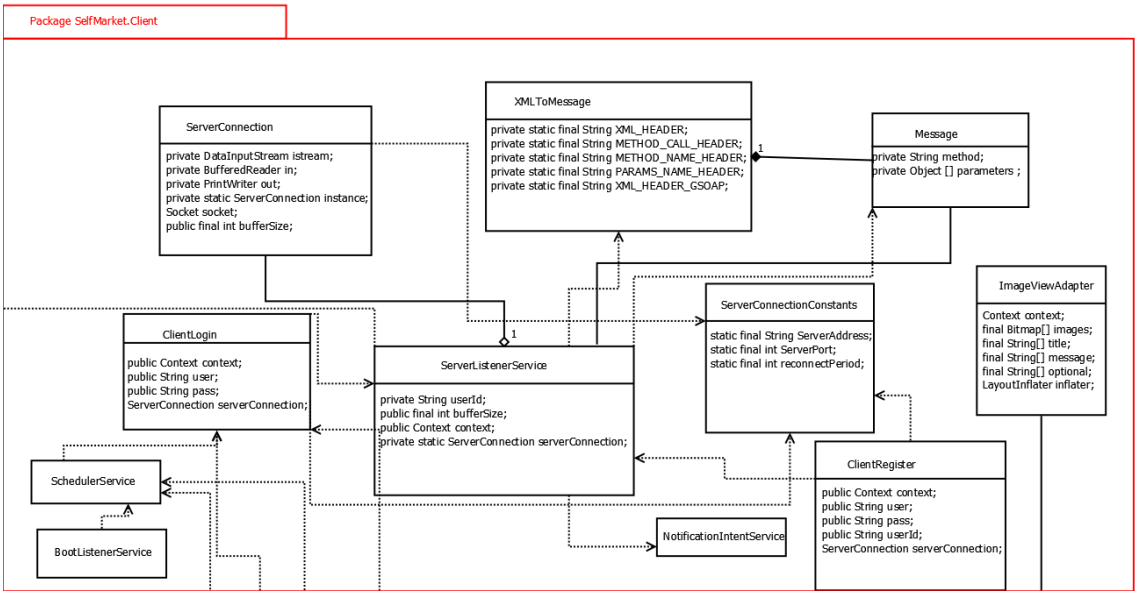


Figure 12. Package Client from the Android app code (cut fragment from original diagram)

To continue, in the package "Selfmarked.Client" we find a total of ten classes and one interface. The classes are:

- ClientRegister
- Message
- ImageViewAdapter
- NotificationIntentService
- XMLToMessage
- ServerListenerService
- ServerConnection
- ClientLogin
- SchedulerService
- BootListenerService

The Interface "ServerConnectionConstants" as his name seems to tell, it's an interface that provides an easy access to the constants we use to connect with the server, that includes the server address and the server port and finally the interface has one more constant that refers to the reconnect period (or timeout period) and we will take about that later on.

Since the Android application will be connecting with the push notification server, as you will probably have noticed, some of the classes inside this package are named exactly the same as some classes inside the push notification server code. In fact, these classes are almost exactly the same as the ones that are in the push notification server code.

To start with the classes inside the package "Selfmarked.Client", to be able to communicate properly with the push notification server, the code contains some specific java classes that provide the needed functionalities to adapt, parse or transform the data to different formats to be able to send and receive the data in the same format that the server is using. These classes are:

- Message
- XMLToMessage
- Common

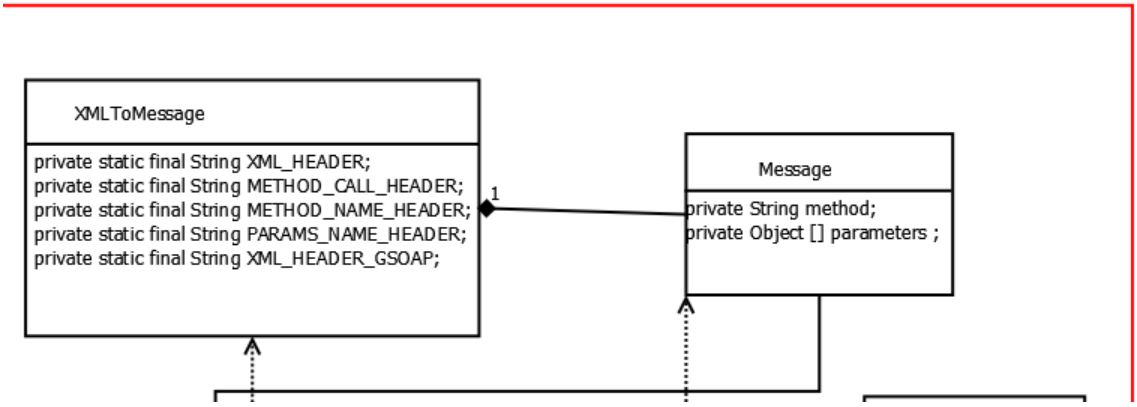


Figure 13. Data format classes of the Android app (cut fragment from original diagram)

The last one is a class that is not inside the package "SelfMarket.Client" but inside the global package "SelfMarket". That is because this class contains functionalities that can be useful in any situation. These functionalities are one to be able to decode an image file parsed using base64 enconding (the same one that the push notification server uses) and one to be able to generate from a Message object an String that follows the XML-RPC format (again, the same one that the push notification server uses).

The "Message" and the "XMLToMessage" classes are the same ones that we find in the push notification server and do exactly the same. That is, the Message class stores all the data that the Android application will send and receive into a "Message" object and provides the functionalities to easily retrieve all the data stored. The second one instead, transforms a received String that follows the XML-RPC format into a Message object.

To continue with the classes inside the package "Selfmarked.Client", we find some classes that are the ones in charge of creating and maintaining a connection with the server. These classes are:

- ServerConnection
- ClientRegister
- ClientLogin
- ServerListenerService

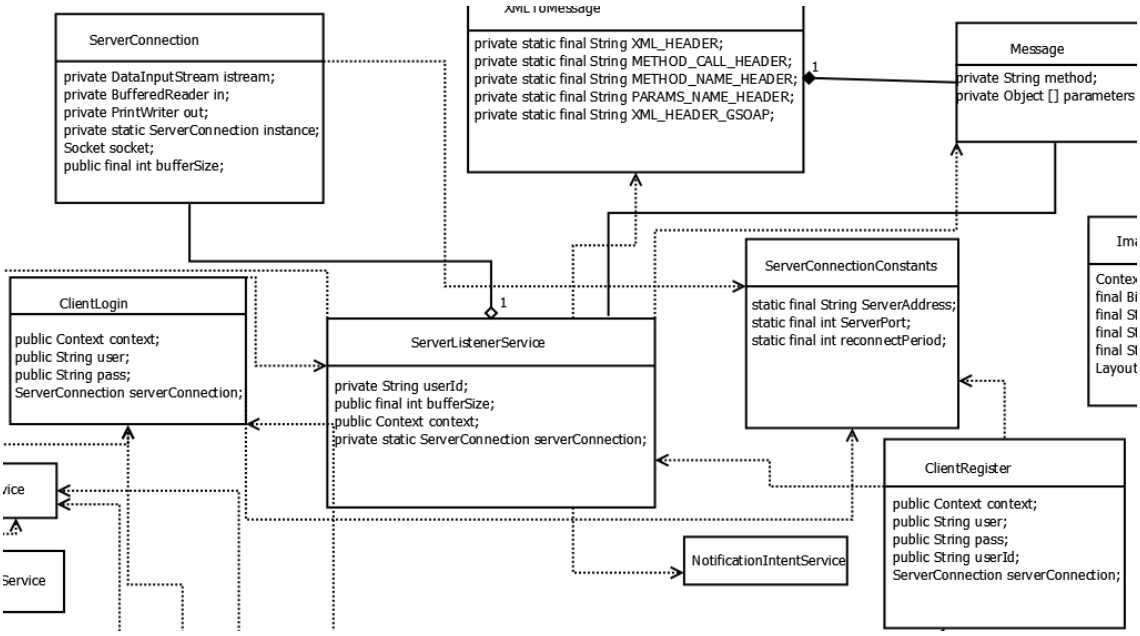


Figure 14. Connection managing classes of the Android app (cut fragment from original diagram)

The "ServerConnection" is a class that follows the singleton design pattern and contains all the parameters of the socket connection established with the push notification server. That is, all the buffers of the socket connection and the socket itself and a collection of "getters" and "setters" methods to be able to obtain and modify all the parameters of the established connection.

The "ClientRegister" and the "ClientLogin" classes are the ones that create and establish the socket connection with the push notification server in background using an Asynchronous Thread (Android API doesn't allow a normal thread to use socket connections, that's why we use asynchronous threats instead). Both classes create the socket and send the petition needed to start the communication with the server. After the communication has been established they store the sockets and all the parameters of the communication to the "ServerConnection" class to be able to manipulate and use the connection parameters in other parts of the code. The "ClientLogin" class is the most important of both because it is used a lot more frequently that the other one. Each time the Android Application will detect a connection lose or any other interruption the "ClientLogin" class will be instantiated to reestablish the communication with the server. However, the "ClientRegister" is only instantiated the moment the user wants to register a new account.
The only reason we are using two different classes to establish a communication with the server is that the register petition needs different parameters than the login petition. While the login petition only needs to send the username and password to the server, the

register petition needs one more parameter: the user ID. **Do not confuse** this user ID with the ID of the user in the MySQL database they are different! The user ID in this case is the in fact the epc of the RFID tag that will be attached to the user shopping card. Each user has only one unique epc and that epc will also be printed in the shopping card. When registering a new account the user will have to fill in that epc. That's the way we are able to identify the account that the user creates with the user itself.
To end with these two classes, once they have successfully connected with the server they instantiate the "ServerListenerService" class.

The "ServerListenerService" class implements an Android Service and because of that, is able to be started without launching the application and to run silently in background. The "ServerListenerService" listens and reads every data received from the server and depending on what is received executes one functionality or another. Its code is very similar to the "ClientThread" class inside the push notification server. With the "ServerListenerService" we parse the received data from the server in background and we are able to start the service that will create the push notifications to the device and to store the images that will be loaded in the screen.

To finish with, we have left four classes:

- SchedulerService
- BootListenerService
- NotificationIntentService
- ImageViewAdapter

The First two classes are also implementations of Android Services and they are **crucial** to be able to recover the connection once it gets lost or closed for any reason.
The first one "SchedulerService" is a service that starts every period of time and looks if the "ServerListenerService" is running. If it is not, it launches an instance of the "ClientLogin" class to reconnect the application with the push notification server. The time period which this service is triggered to start, to look for connection loses, is defined by the "reconnectPeriod" constant found in the "ServerConnectionConstants" interface.

To continue, to be able to reconnect even when the device has been shutdown or rebooted, we have the "BootListenerService" class. Once again, this class is an implementation of an Android Service and uses the "receive boot completed" permissions found in the Android API and declared in the manifest of the application in order to get this service started automatically after the device has been booted up. Once the service is started it automatically launches the "SchedulerService" that will recover the connection with the server.

After, we find the NotificationIntentService, that is an implementation of an Android Service and is the class that creates and loads the push notifications according to the data received from the server. This service is started by the "ServerListenerService" when the data received by the server contains the procedure call "notification" in the XML-RPC formatted data received. It is important to remember that both the server and the Android application are always sending all the data using that format, and with

that format, the "ClientThread" class of the push notification server and the "ServerListenerService" are able to easily distinguish which petition are receiving alongside with the data contained in the XML, by just looking into the procedure calls.

To end with the last classes, the "ImageViewAdapter" class provides the required functionalities to be able to display the promotions into an Android "ViewPager". This is useful to display the promotions received into the "MainScreen" activity.

Database Architecture:

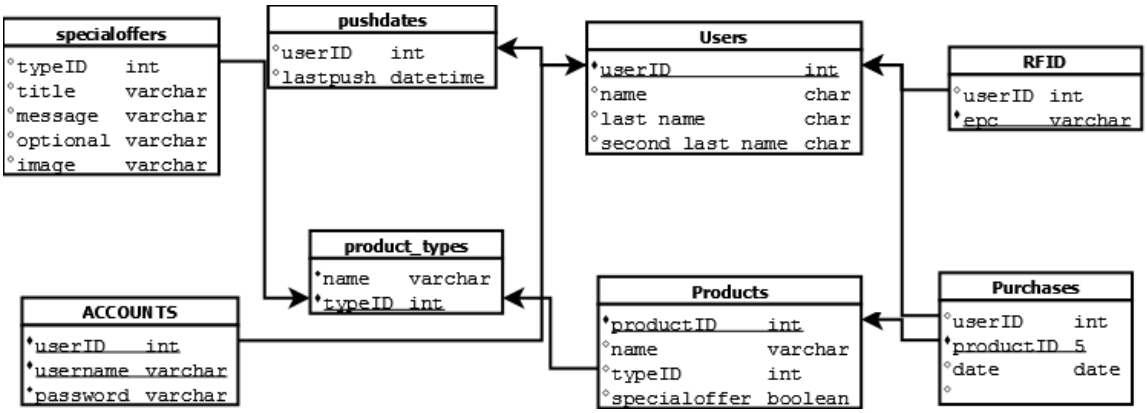


Figure 15. Database Architecture

The MySQL database is the module that stores all the dependent data from the whole system to work properly.

Some parts of the database simulates the tables that could be found in any database of a store and can easily be changed or modified by only having to adapt some tiny parts of the code of the push notification server (concretely we would only have to change the code of the "ScannerWindow" that is in charge of finding the appropriate promotions for each user). On the contrary, the other tables are almost mandatory in order to make the system work, because the system is highly dependent on them.

To begin with, the tables that are simulating those tables that could be in a database of any store are:

- Purchases
- Products
- Product_types
- specialoffers

To start with these tables, the "purchases" table represents a table containing a list of the purchases done for each shopper of the store. With this table we are able to discover the preferences of each user. The system will be more precise the more each shopper buys products to the store and fills more rows to the table. We have to take into account while the last affirmation is true, there is another variable that will make the targeted promotions more or less precise and is the consistency in the number of buys, because the algorithm that fills the targeted preferences of each user as we described in the "general architecture" section, only takes into account the products each user has bought in the last six months.

To continue, the table "product_types" contains all the different categories that a product of the store could belong to. The products can only belong to one category,

because of that as we described in the "general architecture" section, the design of this table is very important.

To continue, the table "products" contains all the products that a user can buy in the shop and it relates them with the "product_types" table to be able to identify in which brand the product belongs.

With this lasts two tables and using the "purchases" table, we are able to identify which is the "product type" that each shopper buys products from it the most.

Finally, we have the table "specialoffers" that contains all the promotions that are running in the store. For each running promotion, we have to fill which product it belongs to, a title of the promotion, a description, an additional message that we use to specify the period validation of each promotion and the path of the picture of the promotion.

Coming up next, we have the tables that contain the highly dependent data for the system and some others. These tables are:

- users
- RFID
- pushdates
- accounts

First, the table "users", contains all the data of the users that are registered in the shop and have a store card from them. With this table we can identify each different user and link the created accounts and their shopping card number with them.

Second, the table "RFID" is the table we use to link each user with their own epc from the RFID tag attached to their shopping card. With this table we are able to identify when a user that has an account registered is entering the store.

Next, the table "pushdates" is the table that contains the date of the last time each user has received new promotions to their device. With this table we are able to avoid collapsing the push notification service, adding a restriction to only send promotions to users that haven't received anything in the last 24h. With that way, we avoid first spamming a user that crosses multiple times the entrance or stays it at there and secondly we are able to avoid the push notification server from doing useless work, since we want the server to consume as less resources as possible.

Finally, the table "accounts" is the table that contains the accounts registered from the Android application of each user. It is with this table that the server is able to know in which device it has to send the promotions once a user has been detected in the store. To be able to link one account to one user, we demand the users to introduce their shopping card number the first time they are registering an account. With the table "RFID" we will then be linking the account with the user itself.

4.3.2 Interface design

Android App interface design:

In order to be able to connect to the server and receive promotions and notifications, the user installing the app will have to fill some steps using the graphical interface provided once the application is started.

The screens that the user will need to interact with at first are shown below:

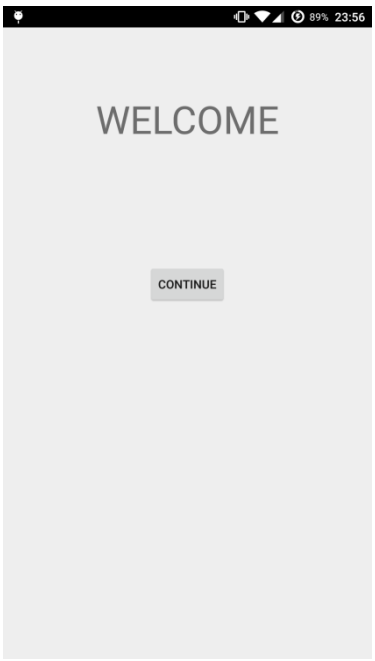


Figure 16. Android app welcome

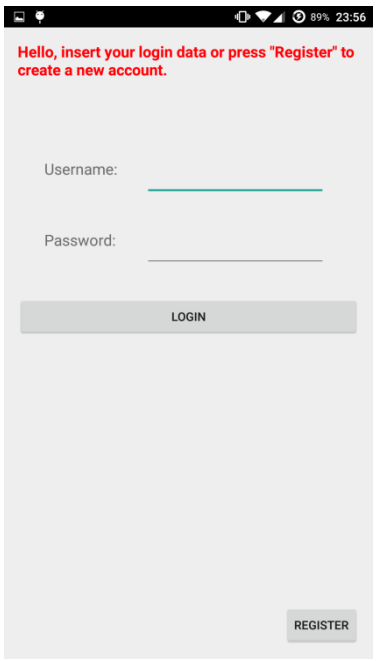


Figure 17. Android app login

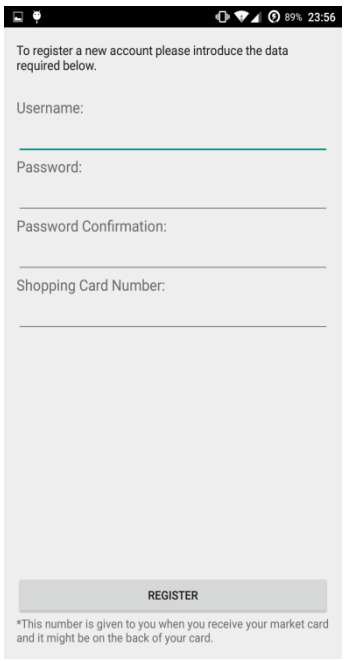


Figure 18. Android app register

The first screen will always be shown when starting the application, pressing the "continue" button will have different behaviors depending on the state of the application. If the user has logged in before to the server with an account, if the application is not currently connected with the server, it will attempt to connect to the server and will display right after the "MainScreen" if it connects successfully, however, if it doesn't, it will erase the login credentials that are saved locally and will display the "login screen" contained in the "Login" activity class in the code, right after. However, if the user has never been logged in with any account or there are not any login credentials saved, the application will just display the "login screen" contained in the "Login" activity class in the code.

To continue, with the "Login Screen" the user will be able to login to the server using an account if it has registered any before or will be able to go to the "register screen" if it doesn't instead. To login to the server with an existing account the user will have to introduce the credentials of his account using the fields "username" and "password" and

press the button "login". Once the "login" button is pressed, the application will attempt to connect to the server using the credentials introduced only if the username introduced has the minimum size (this number is found in the "LoginDataConstants" interface in the code and currently is three) and if the password introduced has also the minimum size (also found in the "LoginDataConstants" interface and is currently six). If the minimums are not meet the application will display an error message instead. While trying to login to the server the application will display a message that informs the user that the application is trying to connect. After four seconds the application will verify if it has connected successfully and if not it will display a message describing why it has not connected (timeout, credentials wrong, etc.). However, if it has successfully connected it will display the "MainScreen".

Following with the "register screen", the user will be provided with everything needed to create a new account in this screen. To create the new account, the user will have to fill the fields present in the screen.

For the "username" field, the user will have to fill a username with a length equal or superior to the minimum (found in the "LoginDataConstants" interface in the code and currently three) and that has not been used before or else it will get a specific error message displayed for each case.

Continuing with the "password" and "password confirmation" fields, the user must introduce the same password in both fields and it's length has to be equal or superior to the minimum (found in the "LoginDataConstants" interface in the code and currently six) or else it will get a specific error message displayed for each case.

To finalize with the fields, in the "shopping card number" field the user must introduce the epc that has been assigned to him and will be printed in the shopping card, since the RFID tag is attached to it. This epc should not have been registered into any account or else the user will get an error once trying to register the account. The epc also has a minimum length (found in the "LoginDataConstants" interface in the code and currently equal to eight) or else the user will get also a specific error message displayed.

To end with the registration process, once the "Register" button is pressed if the fields introduced are ok, the application will attempt to register a new account to the server, if not the application will display any of the errors mentioned before. Four seconds after the "register" button has been pressed and the register petition has started, the application will verify if the registration process has been completed and in any of the cases it will take the user to the "login screen" with the difference that if the registration process has not been completed it will also show a message with the error occurred.

Once the user has been logged in to the server, it will get the "Mainscreen" displayed. This screen displays the promotions that the user receives each time the server sends them to the Android application. It displays the picture, the title, the description and the valid date ("optional" field in the "specialoffers" table at the MySQL database). All this information of the promotions is displayed in a "ViewPage" that the user is able to scroll side by side to show the next or the anterior promotion. The user can also use the arrows found at the bottom of the screen.

The first time the user accesses this screen and while the user has not received any promotion before, the screen will display the logo of the application and "missing" in the promotion fields. That is the default state of the screen.
In the next page you can see the different states of the screen before and after receiving promotions.

States of the "MainScreen":

1.

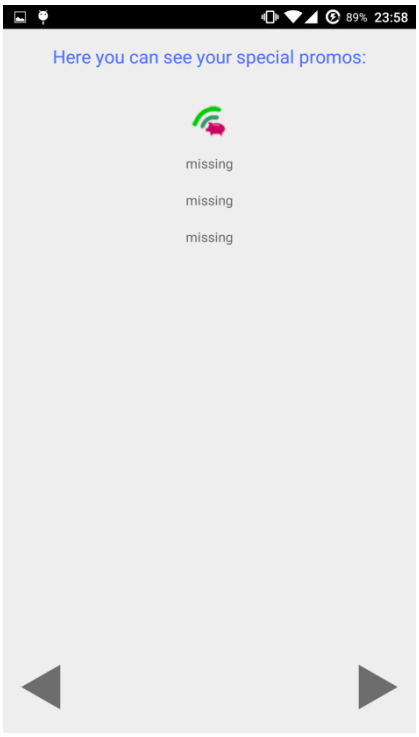


Figure 19. Android app main screen empty

2.

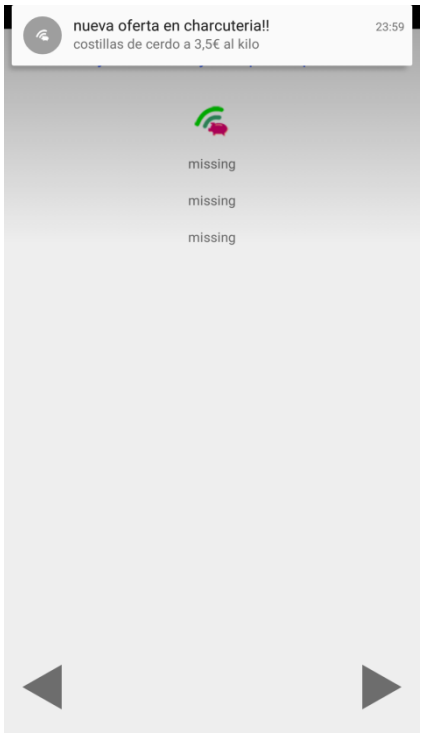


Figure 20. Android app notification in main screen

3.

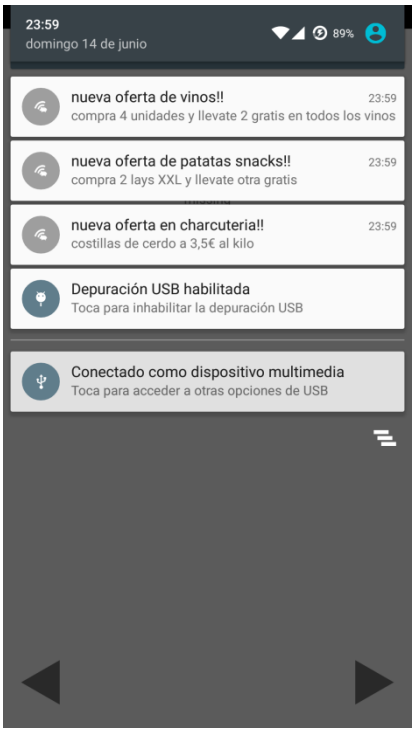


Figure 21. Android app notifications

4.

Figure 22. Android app main screen with promotions

Server interface design:

In order to make the push notification server to work, the administrator will have to fill some steps using the graphical screens provided, once the application is started.
The first screen displayed will be the "WindowFrame", that provides the interface below:

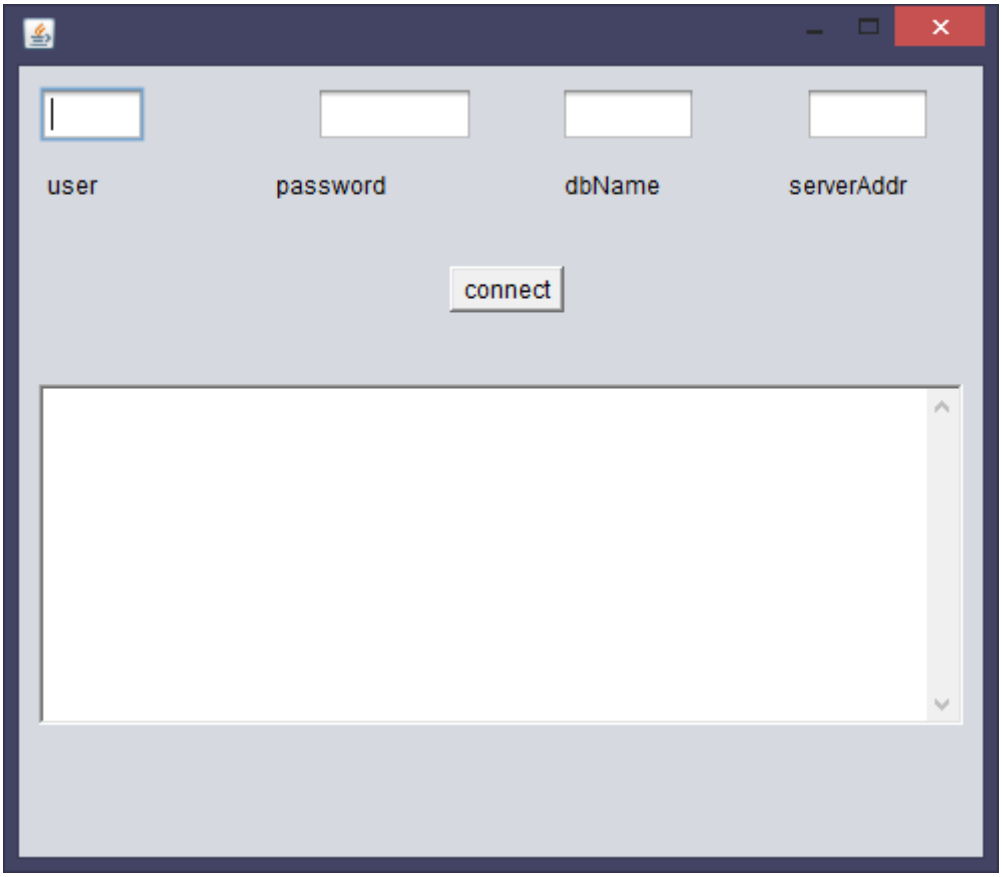


Figure 23. Server First Screen

This screen allows the push notification server to save and use the parameters needed to connect to the MySQL database. The administrator will have to fill the first two labels with the credentials to access the database the third with the name of the database and the fourth with the remote address of the database. After all the labels all filled pressing the button "connect", will load the display of the next screen and start the part of the server that manages the communications with the Android applications ("PushServer") only if the server is able to connect to the database successfully, if not it will display any error occurred in the text area bellow the "connect" button instead and not launch anything.

The next screen is the main screen of the server and is named as "ScannerWindow" in the code. This screen will display everything the administrator needs to know while the

push notification server is running: the detected users, the detected tags that do not belong to any user (if any) and also the possible errors during the launch (if any). The interface of the screen is shown below:

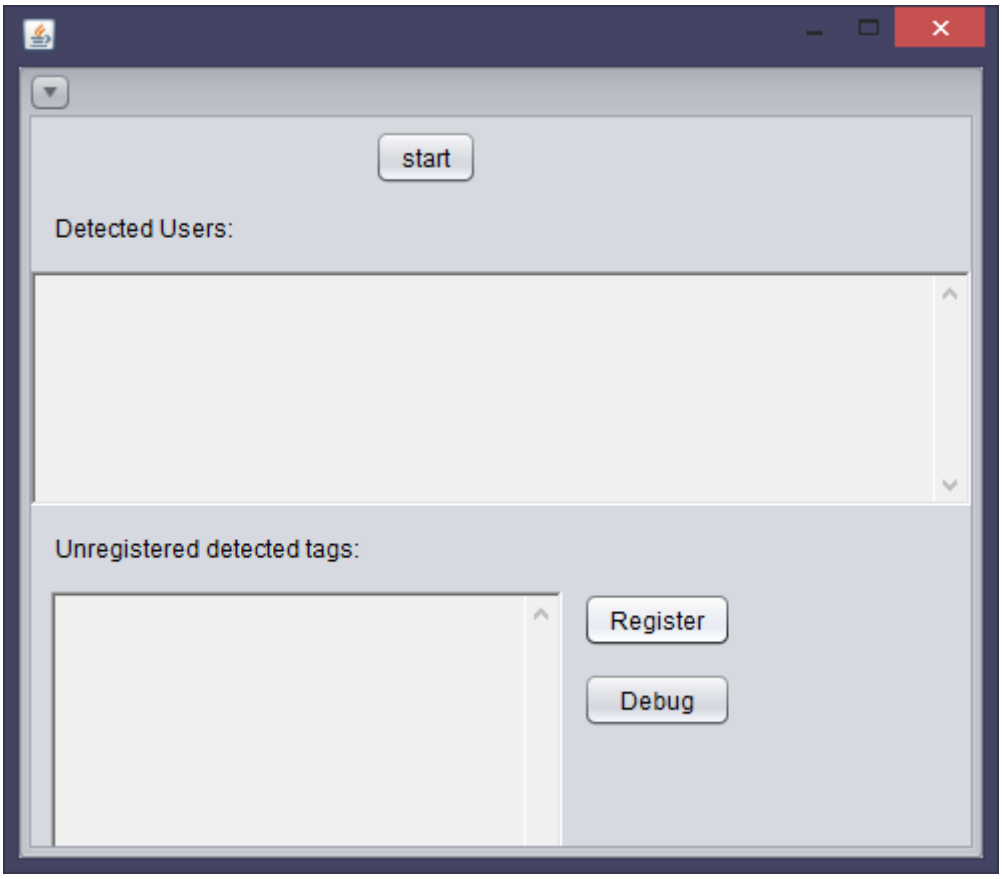


Figure 24. Server Main Screen

Beginning with the text areas, the text area at the top is the one that will be displaying the detected user using the RFID system. The text area below however, will be the one displaying the detected RFID tags that doesn't belong to any user (if any).

As for the buttons, the button "start" will launch the main algorithm, that will make the server read for the detected users or unregistered tags and do all the intended behavior that is designed for (query the database, send push notifications etc.). The button "Register" will not stop the main algorithm if it has started and will only display the "register" screen contained in the "RegisterWindow" class in the code. The button "debug" will also not stop the main algorithm of the push notification server and will only display the "Debug" screen contained in the class with the same name in the code.

The debug screen is a screen that provides all the tools that the administrator of the server needs to test if the server is able to send any data to a connected device. For it, the screen contains a lot of labels to fill in order to create a valid notification that will be sent to the user specified. The interface of the screen is shown below:

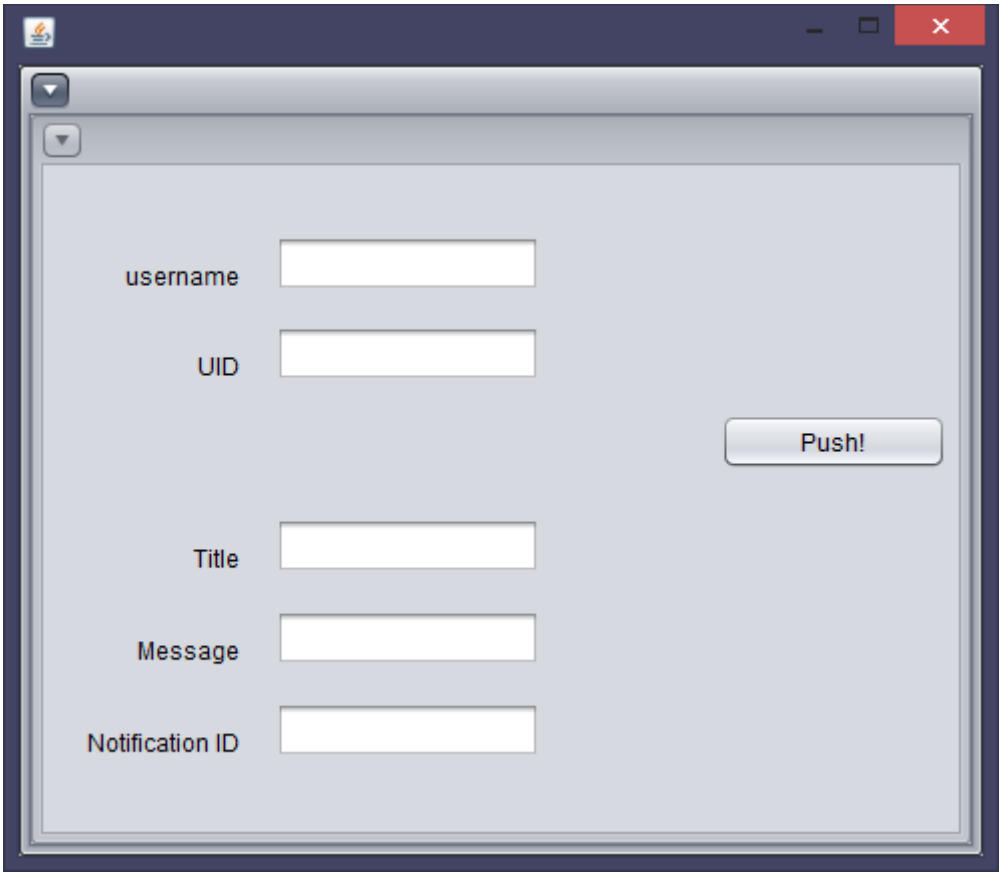


Figure 25. Server Debug Screen

Starting with the first two labels, the administrator has to fill at least one of them, being able to leave the other one blank (optional). They are used to target the notification that will be generated to a specific user that has to exist in the database and be connected to the server with the Android application. The "UID" field refers to the ID of the user that you want to send the custom notification, **inside the MySQL database** (the administrator will have to look at the "accounts" table to know which one is the ID); the "username" field however, refers to the account username of the user that you want to send the custom notification.

Finally, the last three labels are used to specify the data you want to be displayed in the custom notification that will be generated. The "title field is used to write the title of the notification that will be sent, the "message" field is used to write the message text that will be displayed in the notification and finally, the "notification ID" is the id of the notification that the Android application will take to distinguish that notification from others in the same application. In short, the notification ID is used not only to display multiple notifications in the app but also to know in which order place the pictures that are sent. For example, if you send a debug notification with the ID '1' the picture that is sent with the notification will be displayed first in the application. It is important to note that in the debug case, the picture that it is sent is a picture found in a very specific path relative to the project: "src/ID.png" where "ID" is the number introduced in the "notification ID" field. Another important note is that currently the number of images

that can be displayed in the android app is capped at 3 (this of course can be changed easily) so any number higher than 3 will not work.

Finally, the last screen left is the "Register" screen that is contained inside the "RegisterWindow" class in the code of the push notification server. The "register" screen provides the functionality to easily create a new user in the database and assign a specific epc of an RFID tag to him. The interface of the "Register" screen is shown below:

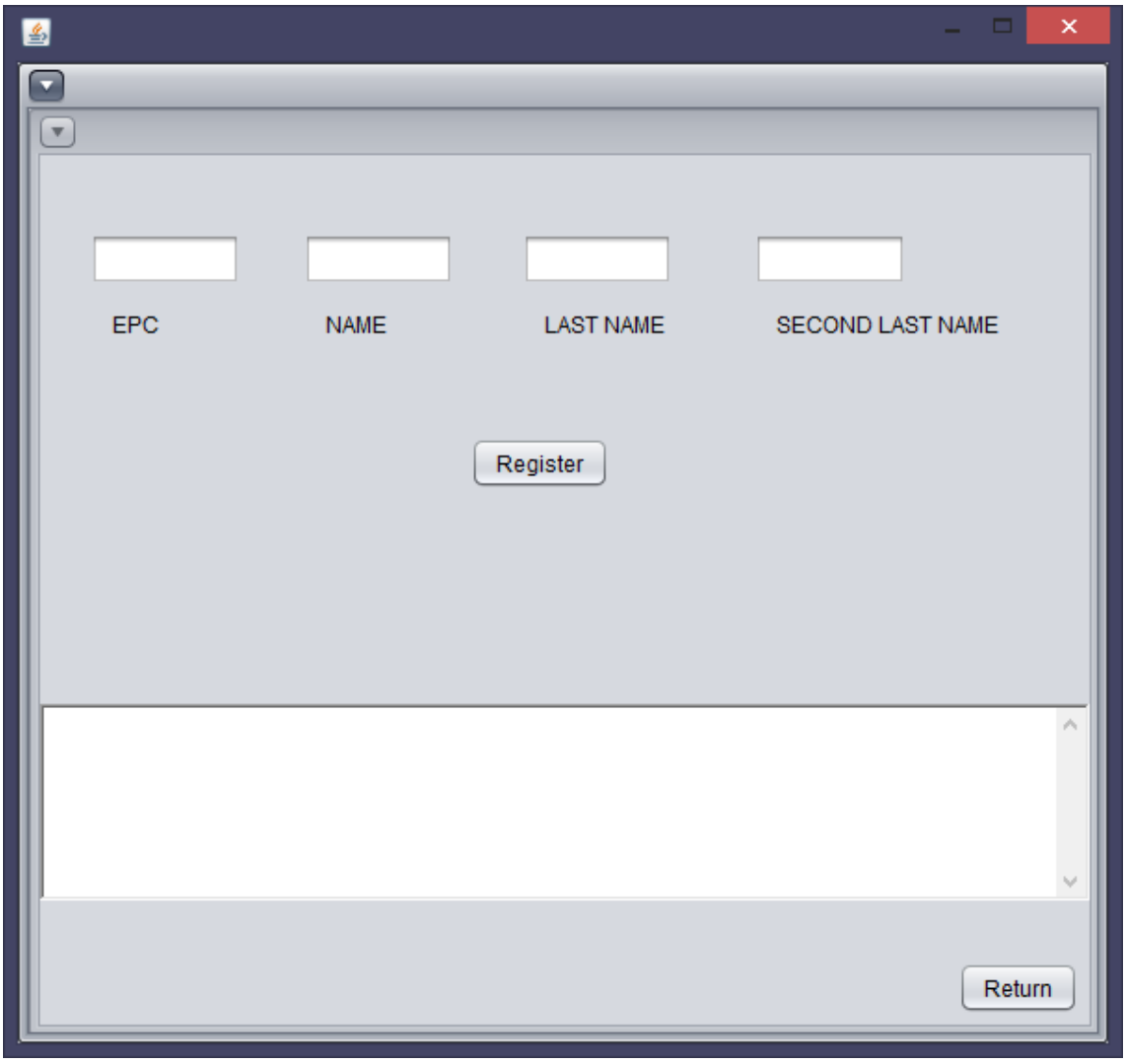


Figure 26. Server Register Screen

The field "EPC" is used for the epc that will be assigned to the user being created (this is used to create a new row in the "RFID" table). However, the "Name", "Last name" and "Second last name" are used to specify the user real name, last name and second last name (these fields are used to create new row in the "USERS" table").

5. EXPERIMENTAL RESULTS

5.1 Tests on the UbiCA lab laboratory

During the project, we have performed multiple simulations in the UbiCA lab laboratory. We can differentiate the simulations performed in two categories: the basic simulations and the real world scenario simulation.

Basic simulation:

Before doing a real world scenario simulation, we have been doing some basic testing during the development of the project multiple times. In these basic simulations we were using a standalone RFID antenna (model 'ADAN-p12EU-FL-200') and testing the detection of RFID tags in multiple situations. The data gathered is shown in the tables below:

The first table represents the mean (from multiple tests) of the distance where the tags were detected while setting different numerical power to the antennas with the tags being inside and outside a wallet:

Antenna power (dbm)	Distance inside wallet (cm)	Distance outside wallet (cm)
21	-	20
23	-	30
25	-	50
27	15	75
30	30	125
31.5	40	200

Table 3. Basic simulations

Real world scenario simulation:

Figure 27. Real world scenario simulation

At the end of the project, when everything was finally implemented we have performed at the UbiCA lab laboratory some simulations of the system using a simulation of a real world scenario with a platform simulating the gates of a store with an RFID antenna attached to it in the top side of the gates.

The table below show the mean (of multiple tests) of the maximum distance where the RFID tags were detected by the antenna with different output powers and with the tags being inside a wallet or outside a wallet. All the tests have been performed with users coming from multiple directions and going through the simulated gates where the antenna was attached:

Antenna power (dbm)	Distance inside wallet (cm)	Distance outside wallet (cm)
21	-	20
23	-	100
25	150	200
27	200	250
30	250	350
31.5	350	350

Table 4. Real world simulations

5.2 Conclusions about the tests

As we can conclude with the tests, in a real world scenario our system performs well and can be implemented without many constraints. As we can see from the tables, for the real world scenario, there is a fair margin about the output power of the antenna even when the tags are inside a wallet. The antennas can run from 25dbm to 31.5dbm if we want the users to be able to have the cards inside their pockets and they can even run to a minimum of 20dbm if we want them to have to approach their shopping cards. Below 20dbm the antennas were not able to detect any tags in any of the simulations done, even if putting them touching the antennas. Also, with the maximum output power the tags where being detected even if putting them in contact with the floor inside a wallet and covering them with clothes.

Another important thing that we found during the tests is that a user does not need to cross the gates in a concrete way or direction to be able to detect its tag, in fact the tags where detected with the same precision coming from any direction while crossing the gates. This also shows us that the project could really be implemented in a real world scenario.

As for the results of the basic simulations, we can see that if a store does not want to install the antennas on its entrance, we would then have to require the users to put their shopping cards near a simple antenna installed in the store in order to send the targeted notifications to them because the ranges of a single antenna are a lot shorter and we were actually having difficulties to detect the tags that where inside a wallet if we were not using the maximum output power.

6. FUTURE WORKS

For the future we have thought about some improvements:

- May add social sharing functionalities to the Android application.
- May add the functionality to be able to order the promotions.
- May add more functionalities and screens to administrate the server (removing a user, be able to manage accounts, ...).
- Possible improvements for the debug functionality with more options to fill (pictures, description...).
- Adding the option to store the pictures inside the external storage of the device.
- Possible improvements for the look and feel of the Android application
- Possible improvements for the recommendation algorithm adding more variables. We could store more date about the users and use it for the recommendation (age?, location?... all that can be used too)

7. CONCLUSIONS

After finishing the project and having experimented with it in multiple scenarios, we can conclude that our project could be implemented in a real world environment without many constraints.

First of all, our project successfully meets the condition of being able to send push notifications in the time-lapse needed. Also, the RFID hardware is able to perform successfully in a real world scenario, only requiring a bit of tune in to find out which power is better because too much power leads to detect the users too far away and too few power leads to users not being detected at all. To continue, our android application is compatible with most smartphones and with the majority of screen sizes beacuse the information has been adapted to feet even with small screens. Finally, our server implementation is able to perform well even in low end computers as we have been using a laptop during our tests and the server was consuming a 0% of CPU time when idle and 1% of CPU time during the sending of push notifications even when there were multiple devices connected.

To finish, while as described our system has the capabilities needed to be implemented in the real world, it only has the minimum functionalities and it is perfect for it to be the basis of a bigger system but before even consider implementing it outside, a more advanced system would require to be implemented, as the system meets only the basic functionalities as it is a research project and for the system to be really representative outside, we would need to add some more functionalities that are mandatory on a commercial system these days as for example social sharing, fill user interface standards, etc.

Bibliography

1. M. Shakila Banu, P. Sasikala, V. Kavitha, G. Yazhini, Lavanya Rajamani and Aruna Dhanapal: RADIO FREQUENCY IDENTIFICATION (RFID): STATE OF THE ART AND ITS APPLICATIONS IN FOOD PROCESSING (2011)

2. A. Rubio Martínez, J. Vales Alonso: SISTEMA PARA LA AMPLIACIÓN DE CONTEXTO BASADO EN RFID PARA TERMINALES MÓVILES (2013)

3. D. Karali: Integration of RFID and Cellular Technologies, UCLA-WINMEC-2004-205-RFID-M2M (2004)

4. Chia-Chen Chen, Tien-Chi Huang, James J. Park, Neil Y.Yen: Real-time smartphone sensing and recommendations towards context-awareness shopping, February 2015, Volume 21, Issue 1, pp 61-72

5. Bruce Dickson, Mary Dickson: Method and system for providing shopping assistance using RFID-tagged items, US 09/859,292 (2002)

6. Ivan Schrodt, Jeffrey Lindsay, Jennifer Marvin, Michael O'Shea, Gregory Benrud: RFID system and method for managing out-of-stock items, US20050149414 A1 (2005)