



Final thesis

*The use of web technologies and
mobile devices to improve the
shopping experience*

Jaume Sala Soler

Computer Systems and Management Engineering

Supervisor: Tony Torp

Tampere, May of 2010

TAMK University of Applied Sciences
Department of Computer Systems Engineering

The use of web technologies and mobile devices to improve the shopping experience

Author: Jaume Sala Soler
Thesis supervisor: Tony Torp
Work order: TAMK University of Applied Sciences
Date: May 2010

Abstract (in English)

This final thesis was made for the department of Computer Systems Engineering of TAMK University of Applied Sciences. The main objective of this thesis is to improve the shopping experience.

In today's market, there are a vast variety of products to buy along with numerous places to buy them. This model of consumerism brings advantages to the client such as the ability to compare different products and stores.

On the other hand, this model also introduces some problems like choosing the appropriate product among the variety we have, choosing the store where to buy the products and the difficulty the customer may face while trying to locate the product inside the stores.

An efficient solution to these problems would be the use of new technologies such as mobile devices and the use of social networks and the Internet. This project aims to bring a solution using these technologies and explore the possibilities of indoor geolocation methods.

Keywords: Web technologies, mobile devices, indoor geolocation, shopping experience, AJAX, PHP5, MySQL

TAMK Universitat de Ciències Aplicades
Departament d'Enginyeria de Sistemes Informàtics

L'ús de tecnologies web i dispositius mòbils per millorar l'experiència de compra

Autor: Jaume Sala Soler
Supervisor del projecte: Tony Torp
Entitat ordenant: TAMK Universitat de Ciències Aplicades
Data: Maig 2010

Abstracte (in Catalan)

Aquest projecte final de carrera es va fer per al departament d'Enginyeria de Sistemes Informàtics de la TAMK Universitat de Ciències Aplicades. L'objectiu principal d'aquesta projecte és millorar l'experiència de compra.

En el mercat actual, hi ha una gran varietat de productes que comprar, juntament amb nombrosos llocs per comprar-los. Aquest model de consumisme porta avantatges per al client, com ara la capacitat per a comparar diferents productes i botigues.

D'altra banda, aquest model també presenta alguns problemes com triar el producte adequat entre la varietat que tenim, l'elecció de la botiga on comprar els productes i la dificultat que el client pot enfrontar en tractar d'ubicar el producte dins de les botigues.

Una solució eficaç a aquests problemes seria l'ús de noves tecnologies, com ara dispositius mòbils i l'ús de les xarxes socials i Internet. Aquest projecte pretén aportar una solució utilitzant aquestes tecnologies i explorar les possibilitats dels mètodes de geolocalització en interiors.

Paraules clau: Tecnologies web, dispositius mòbils, geolocalització en interiors, experiència de compra, AJAX, PHP5, MySQL

Preface

This work has been carried out at TAMK University of Applied Sciences in Tampere, Finland, during the academic year 2009/2010. The realization of this thesis belongs to the collaboration program ERASMUS between this university and University of Vic (Spain).

First and foremost, I want to thank my supervisor Tony Torp for allowing me to write this bachelor thesis at TAMK, also for his dedication and the help that I always needed.

Now I have a chance to express my gratitude to all the staff at the International Office, Mr. Ari Rantala and all the teachers at my home university, thanks for your confidence, offered opportunities and willingness to help me achieve my goals, without your dedication it would have been impossible to be here.

I would also like to thank all the people from UVic's IT Department for providing an excellent working atmosphere during my stay there, which helped me to improve my skills on this amazing world of computing.

Special thanks to the ones who have become my family during my *suomalainen seikkailu* in Tampere: my tutor Juho Jaakkola, all the other crazy tutors from TAMKO, the Santos family, the brothers and sisters from Lapinkaari, the "party people" from the trips to Russia, Mega-Lapland, Stockholm and the "Christmas in Lapland" team. Without you all, Tampere would have been just Tampere.

Finally, I owe special gratitude to my parents Jaume and M^a Dolors, who are "suffering" my absence from home since a long time ago. Thanks for your continuous and unconditional support; you always have encouraged me to go further.

Jaume Sala Soler
Tampere, May of 2010

Final thesis

*The use of web technologies and
mobile devices to improve the
shopping experience*

Jaume Sala Soler

Index

Abstract (in English)	iii
Abstracte (in Catalan)	iv
Preface	vi
1. Introduction	4
2. Objectives	5
3. System analysis	7
3.1 Definition	7
3.2 Classes diagram	8
3.3 Description of the classes diagram	9
4 System design	11
4.1 Definition	11
4.2 User's description	12
4.3 User interface information	14
4.4 System events	19
5 Database design	25
6 Program design	30
6.1 Objects creation	31
6.2 Inheritance	32
6.3 Code structure	34
7 Conclusions and future steps	36
Recommended books	39

1. Introduction

In today's market, there are a vast variety of products to buy along with numerous places to buy them. This model of consumerism brings advantages to the client such as the ability to compare different products and stores.

On the other hand, this model also introduces some problems like choosing the appropriate product among the variety we have, choosing the store where to buy the products and the difficulty the customer may face while trying to locate the product inside the stores.

An efficient solution to these problems would be the use of new technologies such as mobile devices and the use of social networks and the Internet.

This project aims to bring a solution using these technologies. They will help companies to advertise their products and stores which will help the user choose the correct item and help the clients in simplifying the task of shopping.

Obviously, if we are talking about consumerism, the whole project becomes larger. It is not just about clients, products and shops, there are plenty of areas such as product geo-location, price market and shopping preferences to explore. For that reason there is the need to divide the project in different stages.

In the next section I will describe what the main objectives of the project are which will be my focus and later on what objectives should be done in consecutive steps.

2. Objectives

As we said previously, this project aims to solve different problems related to consumerism. To describe those, first we have to talk about the consumers that are going to use the system.

Mainly there are three different types of users:

- **Producers:** These are the users responsible for adding the products to the system. In other words, producers are the companies that manufacture the products that later on are going to be bought by the clients.
- **Sellers:** These are the users responsible for adding the shops and their products to the system. Again, in other words, sellers are the owners of the store(s) and for that reason they know which products they have and where they can be found inside the store(s).
- **Clients:** These are the users that take advantage of all the information provided by the producers and sellers, searching for their preferred shops and products and creating shopping lists. As an extra they are able to add reviews of the shops to the system and so other users can take advantage it.

From the description related to the users using the system we can abstract more elements we need to be able to manage; from the producers we notice the **products**, which most likely have a **category** associated with the item in order to help to order the products.

From the sellers we see the need to manage the **shops** and also the location of the products within them. A way to accomplish that is through the use of **floor plans** so we can position the products somewhere inside the shop which will help the customer locate the product.

As a client we said they are able to create shopping lists and give shop's reviews, so we have also the need to manage both **lists** and **punctuations**.

Summarizing, the system will be able to manage the next list of tasks:

- Create, modify and delete users (clients, sellers and producers).
- Create, modify and delete products and the categories of those.
- Create, modify and delete shops, their floor plans, the location of the products inside those and the client's shop punctuations as well.
- Create, modify and delete shopping lists.

Obviously the system can be accessed by these 3 different types of users, each of those have the right to manage different parts of the application. For that reason a login system should be implemented to take care of the security in all the application.

By now, in the first stage of the project, I will not take care of the security as I'm implementing all the base code the application need to work, the user interface is going to be modest and not ready for public use. The user interface that I am implementing is going to be a proof of concept of all the things the system will be able to manage but without specifying which user does what.

3. System analysis

3.1 Definition

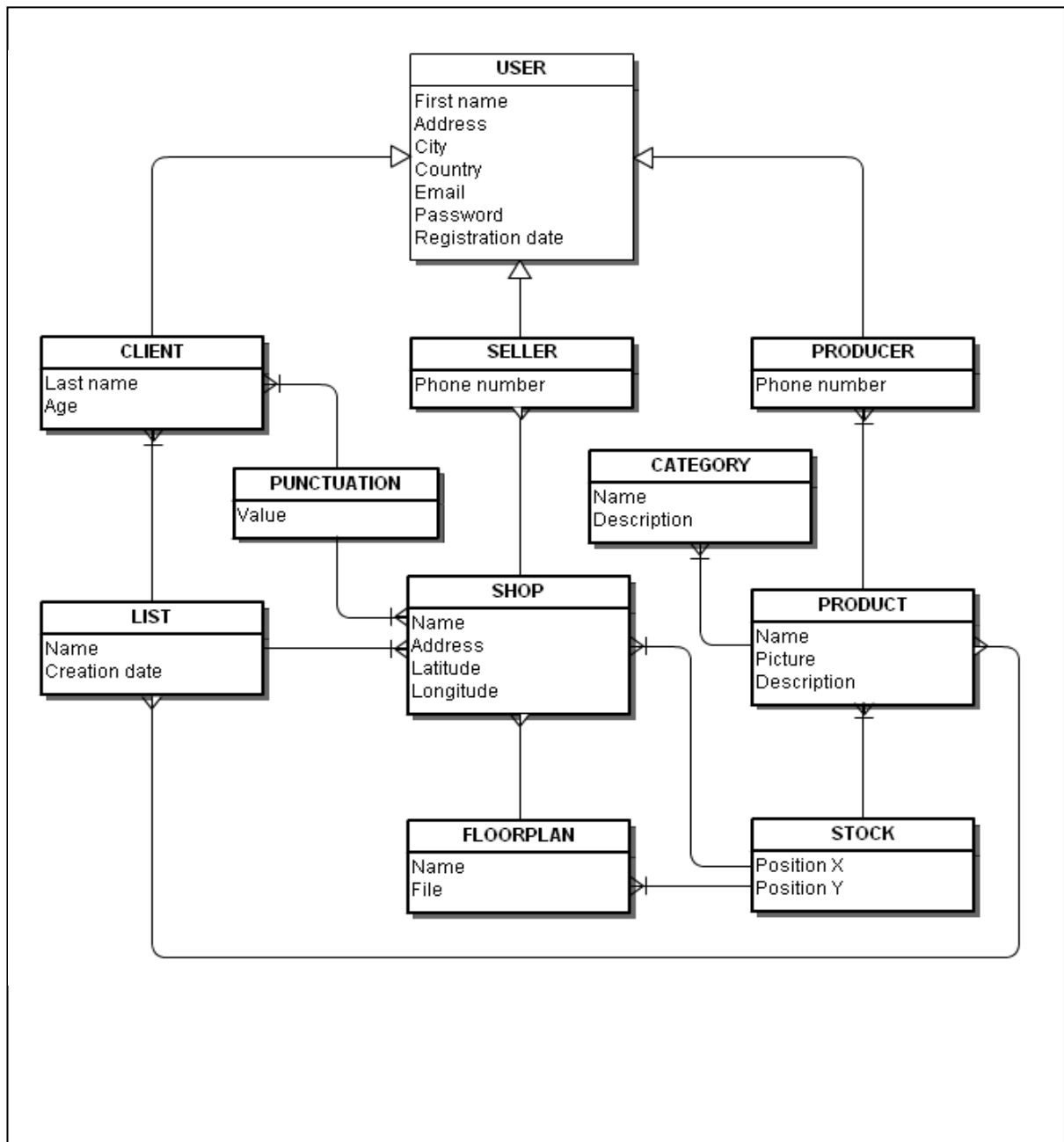
In the system analysis it is essential to know the characteristics of the elements that are part of the project and what kind of relationship exists between them and how this will affect the creation of the database and the development of the application.

Within the analysis we find the class diagram whose function is to display the different objects (classes) involved in the system and how they are interrelated. From each object we will describe their attributes and their services that are going to be implemented. What is not shown in the diagram are those services that are considered simple like create, delete and standard queries. What it will describe are all those that modify any class, or modify the information.

The diagram below will be helpful to know which tables should be included in the database and which attributes each table must have, also it will show which tables we have to access to if, as an example, we want to delete some entry and we want to maintain the referential integrity of the data.

3.2 Classes diagram

Below is the diagram generated from the system requirements followed by the description.



3.3 Description of the classes' diagram

- CATEGORIES

Name: name of the category, *varchar(30)* = "".

Description: detailed information referring to the category, *text* = "".

- FLOOR PLANS

Name: descriptive name of the floor plan, *varchar(20)* = "".

File: name of the floor plan file, *varchar(40)* = "".

- LISTS

Name: descriptive name of the list, *varchar(40)* = "".

Creation date: day and time when the list was created, *datetime* = "".

- PRODUCTS

Name: descriptive name of the product, *varchar(40)* = "".

Picture: name of the product's picture file, *varchar(30)* = "".

Description: detailed information referring to the product, *varchar(200)* = "".

- PUNCTUATIONS

Value: numerical value of the punctuation, *tinyint(2)*, "".

- SHOPS

Name: descriptive name of the shop, *varchar(40)* = "".

Address: local address of the shop, *varchar(50)* = "".

Lat: numerical value of the latitude for geolocation, *float(10,6)* = "".

Lng: numerical value of the longitude for geolocation, *float(10,6)* = "".

- STOCKS

Posx: x-axis of the product's location inside the shop, *smallint(4)* = "".

Posy: y-axis of the product's location inside the shop, *smallint(4)* = "".

- Users

First name: user's first name, *varchar(30)* = "".

Last name: user's last name, *varchar(40)* = "".

Address: user's postal address, *varchar(50)* = "".

City: City where the user lives, *varchar(40)* = "".

Country: Country where the user lives, *varchar(30)* = "".

Email: user's primary email address, *varchar(60)* = "".

Password: user's system password, *char(40)* = "".

- Client

Age: user's age, *tinyint(3)* = "".

- Seller

Phone number: user's phone number, *varchar(20)* = "".

- Producer

Phone number: user's phone number, *varchar(20)* = "".

Producers, Sellers and the Clients classes inherit the attributes of the Users class.

As we said, all the classes implement the next basic services:

- **add:** to instantiate an element.
- **modify*** (* = Data, Pwd, Picture, File): to modify the information of the object.
- **delete:** To eliminate the object from the system.
- **search:** To check if the object already exists and return the object information.
- **exists:** To check if the object already exists.
- **getJSON:** to obtain a JSON representation of the object information.

4. System design

4.1 Definition

The system design is done once the analysis is finished and is oriented towards implementation. We must ensure that the result is a simple application deployment and can be maintained easily so in the future, if we need to expand the project, the cost will be minimal.

To start working in this section first we must take into account the different type of users that will use the application and to know how they will interact with the application (depending on whether users are experts or not). Also, it is important that it is known which events that are performed as we have to know what we have to code. All of this information is also useful when designing the user interfaces that will communicate the users with the system. Simplifying how the application works will make the process much easier for the user as many will most likely not be experienced with such a system and therefore the interfaces have to be easily accessible and with a clear design.

Given all this, the system design can be divided into 3 major parts:

- GUI part, which are the communication interfaces with the user.
- The software that is developed to meet the requirements of the application.
- Database, which refers to the data organization.

4.2 User's description

CLIENT

Relation work/system: The client is the user that takes advantage of the whole system. He can create shopping lists that fits his requirements. Also he can query the system to get information from shops or products.

Needs and requirements: He must be able to access data that refers to him along with the lists. Also he has to be able to query information related to products and shops. All these operations have to be implemented in the simplest possible way, entering the minimum amount of data to perform the tasks.

Workflow: At Any time he can access the application, log in and create new shopping lists entering the data required. In addition, he can delete or modify existing lists. He can also search information for products and shops at any time.

SELLER

Relation work/system: The seller is the user that adds most of the information to the system. He creates all the information related to the shops and the products in those. In a future (next step of the project) he will be able to get statistics from the clients, like which are the most valued products according to the age of the client or where people prefer to buy products.

Needs and requirements: He must be able to access data that refers to him as well as manage the shops and stocks that he created. Like with the clients, these operations have to be implemented in the simplest possible way, entering the minimum amount of data to perform the events.

Workflow: He will be able to access the application at any time, although probably not as frequent as the client, in order to update the products and/or the products' location. To update this information a simple interface should be built as we have to take into consideration the concept of big shops where hundreds of products are stored. The way to

add the location of the products should be a simple process such as using a mobile device with geo-location capabilities.

PRODUCER

Relation work/system: The producer is the user that adds the base information to the system; he adds the product information. Each producer has to add his products to the system to spread itself into the market. As with the seller, in a future, he will be able to get statistics related to his products, like which are the best seller products, which city areas, how frequently users search for his products, etc...

Needs and requirements: he must be able to access data that refers to him; also he has to manage all the information that refers to his products. Like with the clients, these operations have to be implemented in the simplest possible way, entering the minimum amount of data to perform the events.

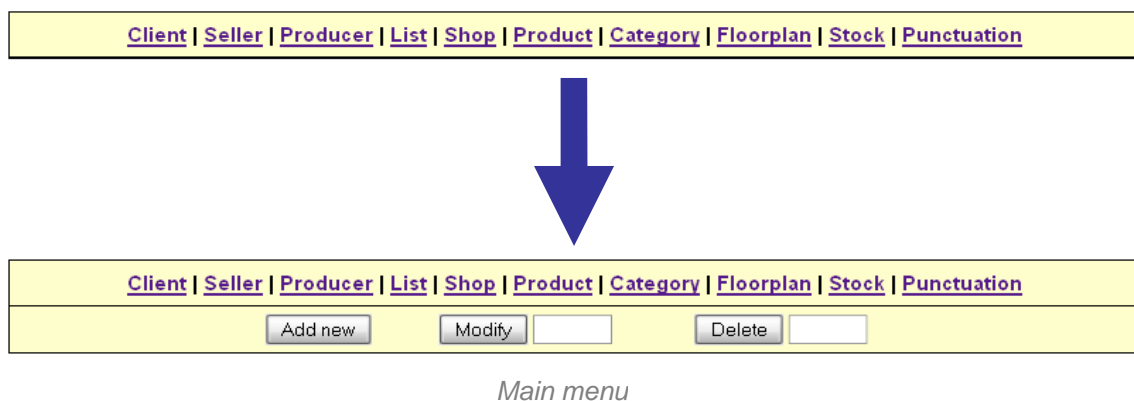
Workflow: At any time he can access the application, this user is the less active user in the system as he will only add information to the database when he wants to create, modify or delete some products. To update this information a simple interface should be build.

4.3 User interface information

As we specified in the objectives of this report, I focus in the base code that handles all the operations within the system, which means no large effort will be spent in order to build the user interface.

The user interface I will build is minimalist and simple, but lets you perform all the operations implemented in the system. Basically the operations of “adding” “modifying” and “deleting” have been implemented and can be performed using the interface.

The main interface is a horizontal navigation bar with all the objects in the system. Each element can be selected and this will show a second navigation bar under the first one with the three main operations we can perform (add, modify or delete).



Furthermore, we have the specific interfaces. The “add” interface is similar in all the “objects” where we ask for the information needed and at the bottom we find the button to validate and the button to reset the form. There are some exceptions like the list, shop and stock interfaces. In the last two (shop and stock) interfaces some of the inputs elements have been implemented dynamically, so it simplifies the task of adding a new element to the system.

[Client](#) | [Seller](#) | [Producer](#) | [List](#) | [Shop](#) | [Product](#) | [Category](#) | [Floorplan](#) | [Stock](#) | [Punctuation](#)

Add shop

Seller:

Shop Name:

Address:
[search location](#)

Location:

Add Shop menu

[Client](#) | [Seller](#) | [Producer](#) | [List](#) | [Shop](#) | [Product](#) | [Category](#) | [Floorplan](#) | [Stock](#) | [Punctuation](#)

Add stock


Shop:

Floorplan:

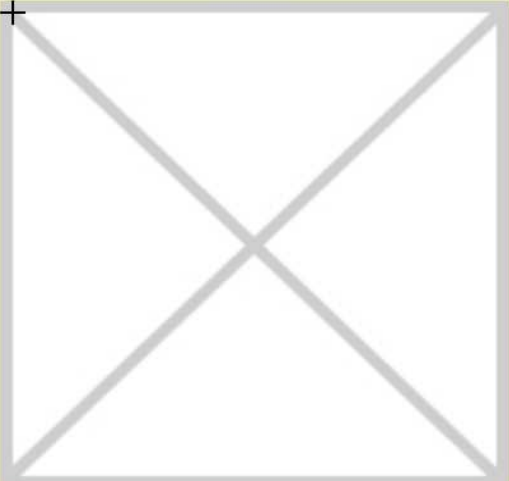
Producer:

Product:

Location:

Product name:	-
Category:	-
Description:	-
Picture:	

Second floor

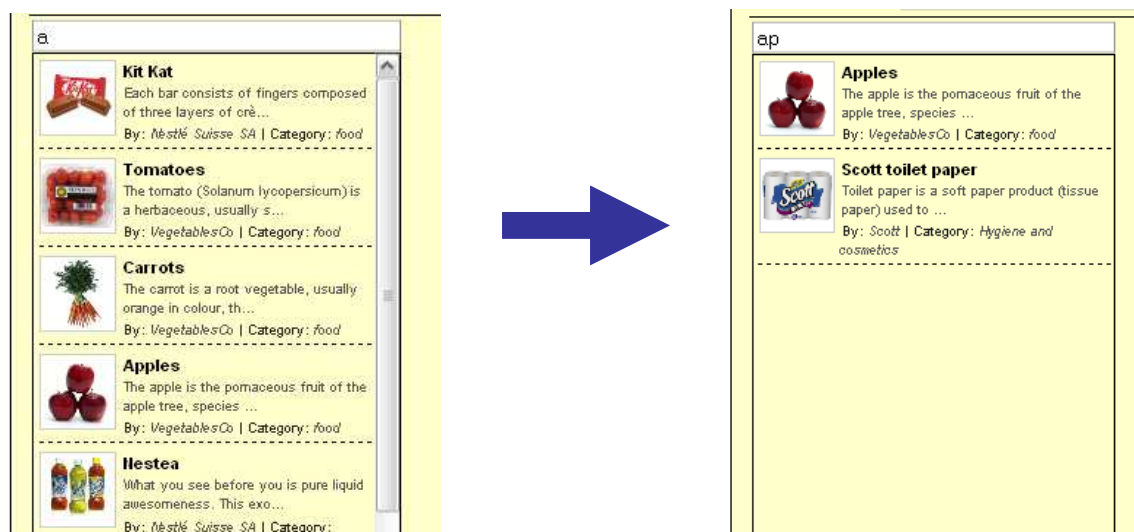


Add Stock menu

In the list interface I added a prototype of what I think the final application should look like for the client. It is really simple to use, with dynamic inputs and all the information displayed.

Add List menu

As we can see, a three-column interface has been implemented. The left column displays all the search results the user performs. If we click one of those results the information (of this product) is displayed in the middle column, showing the product's full information.



Dynamic (ajax) search input to filter the results

Next, if we want to add this product to the shopping list we only have to click in the “add this product to the list” button and automatically the product is added to the right column. This column shows all the products the list contains. As in the left column, if we click any of the elements in the right column the product’s full information will be displayed in the middle column as well.

3-column view. The final shopping list is displayed in the right column.

Finally, the “modify” and “delete” interfaces are practically the same, both show the element information (after querying for it). The “modify” interface lets you modify the information of the object and update it while the “delete” interface only lets you look at the element information and confirm the delete operation.

[Client](#) | [Seller](#) | [Producer](#) | [List](#) | [Shop](#) | [Product](#) | [Category](#) | [Floorplan](#) | [Stock](#) | [Punctuation](#)

Add client

First Name:

Last Name:

Age:

Address:

City:

Country:

Email:

Password:

Password again:

[Client](#) | [Seller](#) | [Producer](#) | [List](#) | [Shop](#) | [Product](#) | [Category](#) | [Floorplan](#) | [Stock](#) | [Punctuation](#)

38

Modify client

First Name:

Last Name:

Age:

Address:

City:

Country:

Email:

Password:

Password again:

[Client](#) | [Seller](#) | [Producer](#) | [List](#) | [Shop](#) | [Product](#) | [Category](#) | [Floorplan](#) | [Stock](#) | [Punctuation](#)

38

Delete client

First Name:

Last Name:

Age:

Address:

City:

Country:

Email:

Password:

Password again:

Add, modify and delete menu interfaces for the client.

4.4 System events

CLIENT

Event: Add client

Response: Add a client to the system.

Description: After all the data has been filled, we add the user to the system.

Services: client.add

Event: Modify client

Response: modify the client data

Description: After the actual data has been displayed we can update it.

Services: client.modifyData, client.modifyPwd

Event: Delete client

Response: Delete a client from the system.

Description: Deletes the client from the system.

Services: client.delete

SELLER

Event: Add seller

Response: Add a seller to the system.

Description: After all the data has been filled, we add the user to the system.

Services: seller.add

Event: Modify seller

Response: Modify the seller data.

Description: After the actual data has been displayed we can update it.

Services: seller.modifyData, seller.modifyPwd

Event: Delete seller

Response: Delete a seller from the system.

Description: Deletes the seller from the system.

Services: Seller.delete

PRODUCER

Event: Add producer

Response: Add a producer to the system.

Description: After all the data has been filled, we add the user to the system.

Services: producer.add

Event: Modify producer

Response: Modify the producer data.

Description: After the actual data has been displayed we can update it.

Services: producer.modifyData, producer.modifyPwd

Event: Delete producer

Response: Delete a producer from the system.

Description: Deletes the producer from the system.

Services: producer.delete

LIST

Event: Add list

Response: Add a list to the system.

Description: After all the data has been filled, we add the list to the system.

Services: list.add

Event: Modify list

Response: Modify the list data.

Description: After the actual data has been displayed we can update it.

Services: list.modifyData

Event: Delete list

Response: Delete a list from the system.

Description: Deletes the list from the system.

Services: list.delete

SHOP

Event: Add shop

Response: Add a shop to the system.

Description: After all the data has been filled, we add the shop to the system.

Services: shop.add

Event: Modify shop

Response: Modify the shop data.

Description: After the actual data has been displayed we can update it.

Services: shop.modifyData

Event: Delete shop

Response: Delete a shop from the system.

Description: Deletes the shop from the system.

Services: shop.delete

PRODUCT

Event: Add product

Response: Add a product to the system.

Description: After all the data has been filled, we add the product to the system.

Services: product.add

Event: Modify product

Response: Modify the shop data.

Description: After the actual data has been displayed we can update it.

Services: product.modifyData, product.modifyPicture

Event: Delete product

Response: Delete a product from the system.

Description: Deletes the product from the system.

Services: product.delete

CATEGORY

Event: Add category

Response: Add a category to the system.

Description: After all the data has been filled, we add the category to the system.

Services: category.add

Event: Modify category

Response: Modify the category data.

Description: After the actual data has been displayed we can update it.

Services: category.modifyData

Event: Delete category

Response: Delete a category from the system.

Description: Deletes the category from the system.

Services: category.delete

FLOORPLAN

Event: Add floorplan

Response: Add a floorplan to the system.

Description: After all the data has been filled, we add the floorplan to the system.

Services: floorplan.add

Event: Modify floorplan

Response: Modify the floorplan data.

Description: After the actual data has been displayed we can update it.

Services: floorplan.modifyData, floorplan.modifyFile

Event: Delete floorplan

Response: Delete a floorplan from the system.

Description: Deletes the floorplan from the system.

Services: floorplan.delete

STOCK

Event: Add stock

Response: Add a stock to the system.

Description: After all the data has been filled, we add the stock to the system.

Services: stock.exists, stock.add

Event: Modify stock

Response: Modify the stock data.

Description: After the actual data has been displayed we can update it.

Services: stock.modifyData

Event: Delete stock

Response: Delete a stock from the system.

Description: Deletes the stock from the system.

Services: stock.delete

PUNCTUATION

Event: Add punctuation

Response: Add a punctuation to the system.

Description: After all the data has been filled, we add the punctuation to the system.

Services: punctuation.exists, punctuation.add

Event: Modify punctuation

Response: Modify the punctuation data.

Description: After the actual data has been displayed we can update it.

Services: punctuation.modifyData

Event: Delete punctuation

Response: Delete a punctuation from the system.

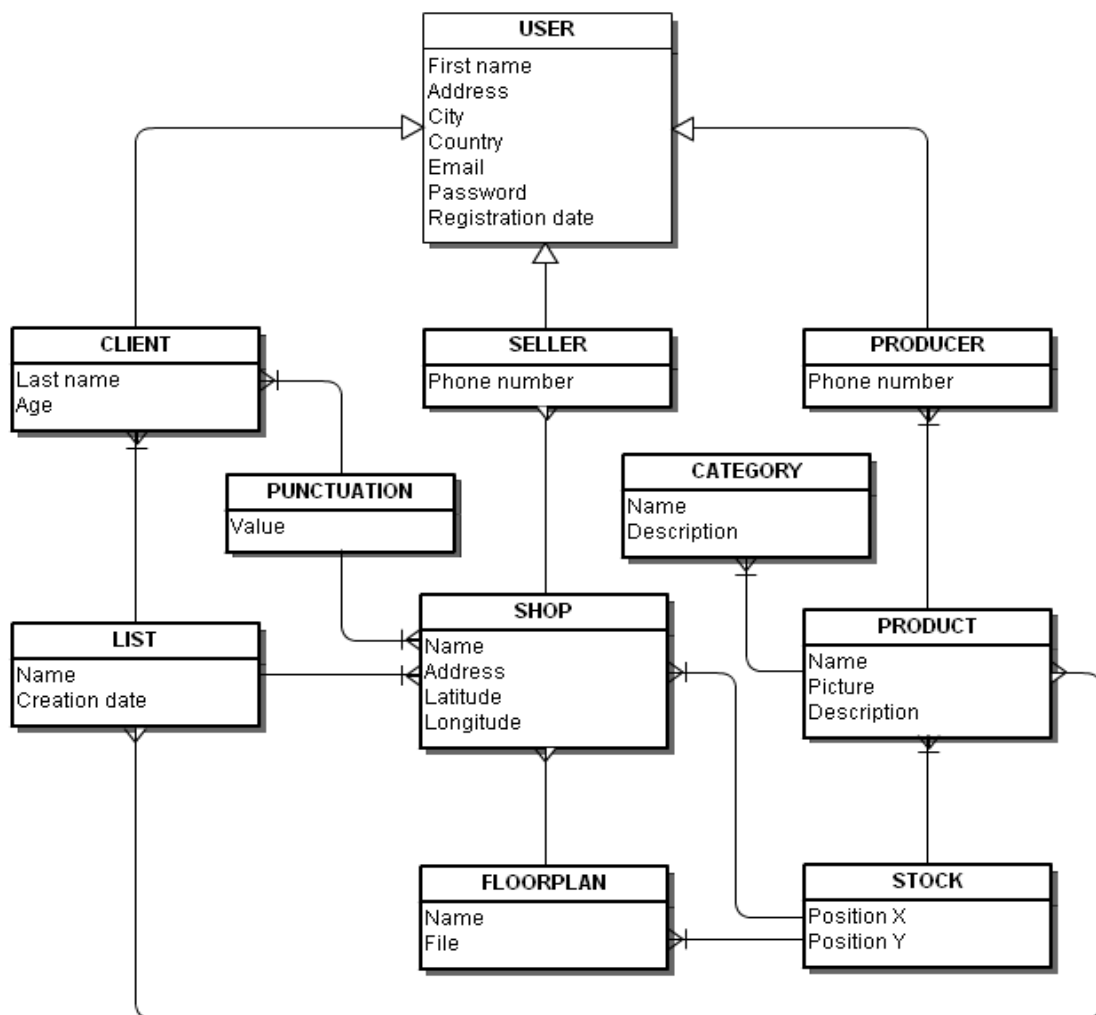
Description: Deletes the punctuation from the system.

Services: punctuation.delete

5. Database design

This section explains how the translation from the "class diagram" to the "relational model" was carried out.

Starting with the class diagram, we can translate each of the classes into a new table in the database. If we look at the diagram, the tables may be initially as follows:



CLIENT (First name, Last name, Age, Address, City, Country, Email, Password, Reg date)

CATEGORY (Name, Description)

FLOORPLAN (Name, File)

LIST (Name, Creation date)

PRODUCER (First name, Age, Address, City, Country, Email, Password, Phone number, Reg date)

PRODUCT (Name, Picture, Description)

PUNCTUATION (Value)

SELLER (First name, Age, Address, City, Country, Email, Password, Phone number, Reg date)

SHOP (Name, Address, Latitude, Longitude)

STOCK (Position X, Position Y)

But we must bear in mind that each class is in relation with other classes, and those relations contain participation restrictions, these are very important in order to make a good translation of the database.

All the relations "N to 1" are easy to translate as we only have to add a table column to the "1-side" relation being the primary key of the class "N-side".

For example, PRODUCT is in relation with PRODUCER, and PRODUCT has the 1-side relation. This means the PRODUCT table should have a column with the foreign key to the PRODUCER table. Thus, the PRODUCT table is as follows:

PRODUCTS (Producer_id, Name, Picture, Description)

Below, we can see the primary key and the foreign key in italics. Now we do the same for the rest of the "1 to N" relations that exists in the diagram. Following is what we get:

CLIENTS (ID, First name, Last name, Age, Address, City, Country, Email, Password, Reg date)

CATEGORIES (ID, Name, Description)

FLOORPLANS (ID, Shop_id, Name, File)

LISTS (ID, Client_id, Shop_id, Name, Creation date)

PRODUCERS (ID, First name, Age, Address, City, Country, Email, Password, Phone number, Reg date)

PRODUCTS (ID, Product_id, Category_id, Name, Picture, Description)

PUNCTUATIONS (User_id, Shop_id, Value)

SELLERS (ID, First name, Age, Address, City, Country, Email, Password, Phone number, Reg date)

SHOPS (ID, Seller_id, Name, Address, Latitude, Longitude)

STOCKS (Product_id, Floorplan_id, Shop_id, Position X, Position Y)

The “N to M” restrictions forces us to create a new table to store the relations between classes. In the class diagram we can see a relation of this type between the classes LIST and PRODUCT. To solve this problem, a new table named ITEMS is created which will contain two foreign keys, one for the list id and the other for the product id. Both keys combined create the primary key of this table.

ITEMS (list_id, product_id)

Finally, if we look to the diagram we can see that there is a class generalization which can be solved grouping all the class' siblings into one and adding an attribute that serves to distinguish the type.

CLIENTS (ID, First name, Last name, Age, Address, City, Country, Email, Password, Reg date)

PRODUCERS (ID, First name, Age, Address, City, Country, Email, Password, Phone number, Reg date)

SELLERS (ID, First name, Age, Address, City, Country, Email, Password, Phone number, Reg date)

Thus, applying this solution, the resulting table is as follows:

USERS (ID, type, First name, Last name, Age, Address, City, Country, Email, Password,
Phone number, Reg date)

Where "type" may be **c**: client, **s**: seller or **p**: producer.

So the final model of the database tables is as follows:

CATEGORIES (ID, Name, Description)

FLOORPLANS (ID, Shop_id, Name, File)

ITEMS (list_id, product_id)

LISTS (ID, Client_id, Shop_id Name, Creation date)

PRODUCTS (ID, Product_id, Category_id, Name, Picture, Description)

PUNCTUATIONS (User_id, Shop_id, Value)

SHOPS (ID, Seller_id, Name, Address, Latitude, Longitude)

STOCKS (Product_id, Floorplan_id, Shop_id, Position X, Position Y)

USERS (ID, type, First name, Last name, Age, Address, City, Country, Email, Password,
Phone number, Reg date)

Below is a short description of each table in the database:

CATEGORIES: Stores the different categories the system let you tag a product. Its primary key is ID.

FLOORPLANS: Stores the information related to the files that are used to show the floor plans of the shops, it doesn't store the file itself. The primary key is ID.

ITEMS: Stores the relations between the tables list and products. The primary key is the combination of both ids, list_id and product_id.

LIST: Stores the information about the list, who created the list, name of the list, it also stores the shop_id to make it more consistent. The primary key is ID.

PRODUCTS: Stores all the information related to the products, the primary key is ID.

PUNCTUATIONS: Stores all the punctuations the users use to rate the shops. The primary key is shop_id and user_id.

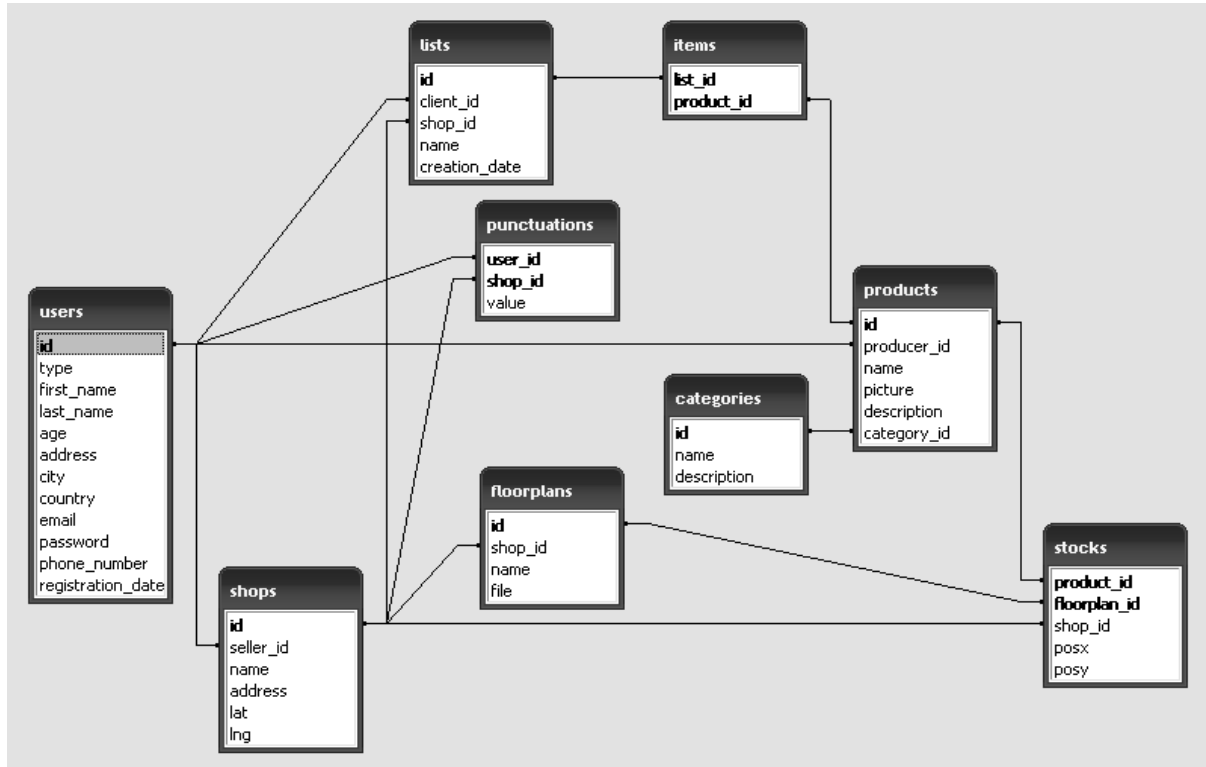
SHOPS: Stores the information about the shops, including the latitude and longitude. The primary key is ID.

STOCKS: Stores the information related to the position of the products inside the shops. The primary key is product_id and floorplan_id but also contains a foreign key to the shop it belongs to (shop_id).

USERS: Stores all the users information, it doesn't matter the type of user, the attribute "type" takes care of what kind of user is stored and what data is necessary.

Finally notice that as we are talking about a relational database, it requires an identifier to differentiate the elements of this, so id's were added to all tables that need it.

Referring to the database, a MySQL 5 database has been used. All the tables are encoded in UTF8 and the tables' engine used is MyISAM.



6. Program design

After the system design comes the program design where the programming language and the techniques used to code the application are described.

We talked about the database which is a MySQL 5 with a MyISAM engine for the tables, now it's time to talk about the programming languages used. As we are talking about a web application a server side and a client side exists. Their descriptions are as followed:

Server side: To code the common gateway interface in the server side, PHP 5 is used. PHP has been chosen because of his potential and flexibility. It processes all the events and handles the connections to the database. All the code have been modularized, the objects have been translated to PHP classes, the database connection have been separated from the rest of the code (`mysqli_connect.php`), and the forms' processing is handled in another file (`mgmt-cgi.php`) like the DB's asynchronous connections as well (`mgmt-ajax.php`).

Client side: The client side has been coded in HTML with a doctype "XHTML 1.0 Transitional". All the HTML code is located in the same file (`mgmt.php`). A CSS style sheet file (`mgmt.css`) and javascript (`mgmt.js`) have been used to create a dynamic interface.

Asynchronous connections, using AJAX techniques, have been used to handle all the queries to the database to retrieve information. All the code to process the AJAX connections are located separately (`mgmt-ajax.php`) from the rest of the application code.

6.1 Objects creation

Objects are instances of each class, so each new one comes out with their attributes and services. As we said a translation has been carried out to create the PHP classes, each class has its attributes and services. Those need to be specified to indicate which parameters are required. To better understand these concepts an example with the PRODUCT class is given.

File: product.php

```
class product
{
    //attribute definition
    private $id;
    private $producer_id;
    private $name;
    private $picture;
    private $description;
    private $category_id;

    //service definition
    function __construct(...){...}
    function add(){...}
    function modifyData($id) {...}
    function modifyPicture($id) {...}
    function delete($id) {...}
    function search($id) {...}
    function getJSON($id) {...}
}
```

From any other object or file we can access the class' services including a reference to the class file and calling the service with a simple syntax. As an example let's say we want to search for a product. The code needed to achieve this is:

```
product->search(product_id);
```

6.2 Inheritance

The inheritance consists in grouping attributes and/or common services of different classes. That way, when you define a new class you only have to indicate which class you use to inherit attributes and services.

In the case of our application, we have a generalization of classes for the users, having a global class (USER), which contains all the attributes and common services, and the CLIENT, SELLER and PRODUCER classes that inherit from this one. The USERS class are coded as followed:

```
abstract class user {  
  
    private $id;  
    private $type;  
    private $fname;  
    private $lname;  
    private $age;  
    private $address;  
    private $city;  
    private $country;  
    private $email;  
    private $password;  
    private $phone_number;  
    private $registration_date;  
  
    function __construct($vtype='', $vfname='',...){...}  
  
    function add(){...}  
    function modifyData($user_type, $id){...}  
    function modifyPwd($user_type, $id){...}  
    function delete($user_type, $id){...}  
    function search($user_type, $id){...}  
    function getJSON($user_type, $id){...}  
  
}
```

As we can see, all the common services and attributes are coded here but we declared the class as **abstract**, this is because the system has no users with type “user”, instead we have

clients, sellers and producers and for that reason these three classes inherit from the abstract class USER. To understand the inheritance let's take a look at the class CLIENT and see how it makes use of the inheritance.

```
require_once ('user.php');

class client extends user {

    function __construct($vfname = '', $vlname = '', ...){
        parent::__construct('c', $vfname, ...);}

    function add(){ return parent::add(); }

    function search($id_client){
        return parent::search('c', $id_client); }

    function getJSON($id_client){
        return parent::getJSON('c', $id_client); }

    function modifyData($id_client){
        return parent::modifyData('c', $id_client); }

    function modifyPwd($id_client){
        return parent::modifyPwd('c', $id_client); }

    function delete($id_client){
        return parent::delete('c', $id_client); }
}
```

As we can see, there is no need to recode the attributes as all of them are handled by the class USER. We only have to call to their parent services with the appropriate arguments the same way all the services inherit from the class USER.

6.3 Code structure

By now we have discussed the code implementation. We will now talk about code organization, or in other words, how the files are organized in the server.

This is a web application resulting in the likely case that there will be a lot of users regardless the fact there is no user login yet, files in the server have been organized thinking of the future.

All the files are placed in the root directory named “aproot”. Inside this we can find two folders, private and public.

Private: Inside this folder we can find all the classes and database connection files. This folder and its content should not be seen by outside users as connection queries and important code implementations are stored in it. Following is the list of files that can be found inside and a short description of those.

- **category.php**: Category object class.
- **client.php**: Client object class, inherits from user.php.
- **db.php**: Contains all the database queries to retrieve list of objects (i.e.: list of clients).
- **floorplan.php**: Floorplan object class.
- **listBuy.php**: List object class.
- **mysqli_connect.php**: Contains the database access information.
- **producer.php**: Producer object class, inherits from user.php.
- **product.php**: Product object class.
- **punctuation.php**: Punctuation object class.
- **seller.php**: Seller object class, inherits from user.php.
- **shop.php**: Shop object class.
- **stock.php**: Stock object class.
- **user.php**: User object class.

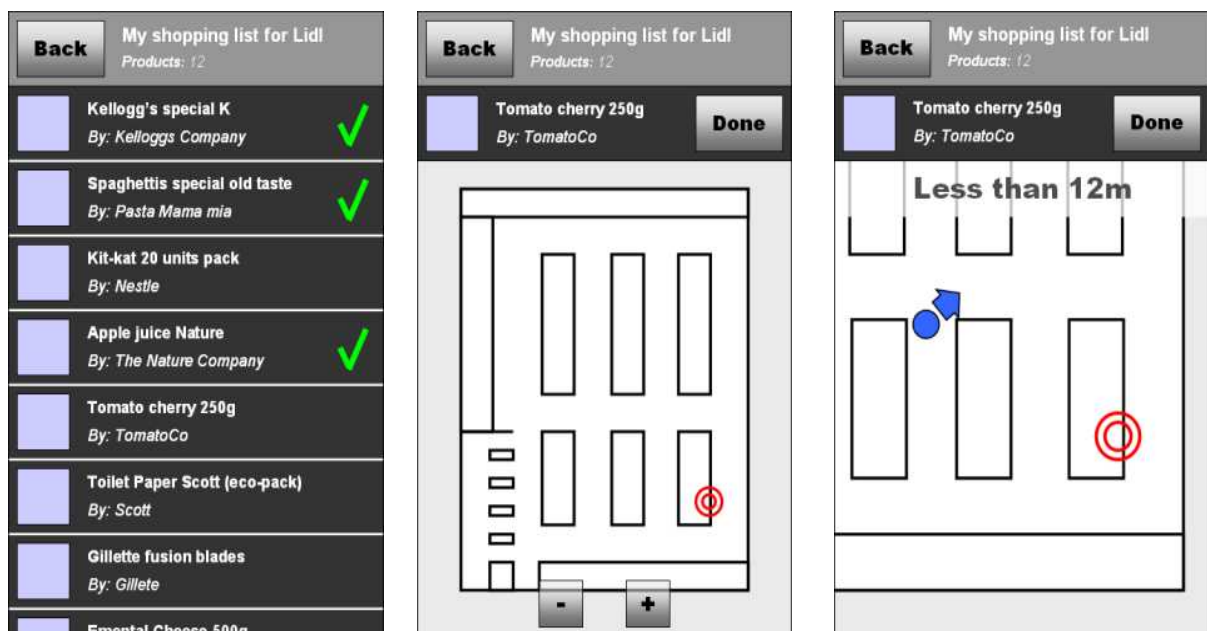
Public: This folder contains files that can be accessed by web browsers, form processing files and different subfolders that store different types of images the system uses. When the application is ready to use, the web domain should point to the file `mgmt.php` as this is like the `index.html` of a web page right now. Following is the list of files and folders that can be found inside and a short description of those.

- **floorplan_images (folder):** Contains all the floorplan images the sellers upload.
- **product_images (folder):** Contains all the product images the producers upload.
- **web_images (folder):** Contains images used in the web page (style).
- **mgmt-ajax.php:** Contains all the functions to handle AJAX connections to the server.
- **mgmt-cgi.php:** Handles all the form requests, all the system events.
- **mgmt.css:** Style sheet of the web.
- **mgmt.js:** Here we can find all the Javascript code the application use.
- **mgmt.php:** The main file, this is the web page to point at if we want to use the application.
- **reset.css:** Style sheet to reset all the default styles the different browsers have.

7. Conclusions and future steps

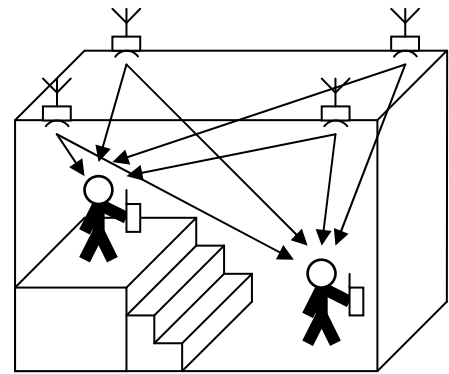
As described earlier, this is just the beginning of a big project, many implementation decisions have been taken during the design and the probable case that some of those will be rewritten or extended in a future. To finalize this thesis by means of conclusion, next is a list of ideas that came to my mind during the analysis but have been discarded because of the time or the complexity but nonetheless are valid ideas.

- Build a friendly user interface, adapted to the different types of users. Clients and lists, sellers and shops, producers and products. Web 2.0 showed us the need to building friendly user interfaces to approach the power of the Internet to the final user.
- Create a mobile interface to help the clients find the products in the shops. By now, the indoor's GPS systems are not mature and for that reason the application uses floor plans, but in the future it is possible that this technology will be mature enough to replace the actual method. Following are some pictures to illustrate what I'm talking about.



UI mock-up for mobile devices. (Left) A list with all the products we selected previously at home. (center) Showing the position of a product with the actual technical implementation. (Right) Showing the position of a product using indoor GPS technology.

This technological problem can be split into another thesis where computer and electrical engineers can work in collaboration to find a solution. Nowadays there are different approaches to solve this problem, some of those are based in a Wi-Fi/GPS hybrid method. Another possible solution could be the use of PANs (Personal Area Networks) to keep track of objects which would require multiple GPS antennae to be setup atop buildings and other obstructions, which would be wired to an indoor RF repeater system that directional receivers could tap into. The indoor segment would not only repeat the signals, but it would reportedly amplify them as well to ensure a solid connection.



- Extend the system to generate statistics to help sellers and producers to sell their products. Things like, best seller products, preferred shops, most active users (by ages), etc...
- As with the clients to find the products inside the shops, a tool to tag the products' coordinates could be helpful for the sellers as they have to input this information to the system for all the products.
- Like in any web application, implement a login system to secure all the data and accounts.
- The last but not the least, this is a huge project and involves the collaboration of different entities, not only end users but shops and companies. One area that could help as a bridge to a full implementation could be the use of these indoor technologies to find the shops (and not the products on those) inside a shopping mall.

As an example, near Tampere there is Ideapark, a really huge shopping mall with more than 150 stores. What if a system to show where the shops are is implemented? What if we go further and a system to show where the products are inside the shops and at the same time advertise other products within the place? Here is where my thesis ends but new ones are started...

Recommended books

Here is a list of books I used to create the application and also some books I recommend for the future steps. Some of those are graphic design books with no connection with computers, but still useful for the creation of future user interfaces.

- BUDD, Andy (2009). *CSS Mastery: Advanced Web Standards Solutions* (2nd Edition). Friends of ED.
- CEDERHOLM, Dan (2007). *Bulletproof Web Design: Improving flexibility and protecting against worst-case scenarios with XHTML and CSS* (2nd Edition). New Riders.
- FREEMAN, Eric & Elisabeth (2004). *Head First Design Patterns*. O'Reilly Media.
- KEITH, Jeremy (2007). *Bulletproof Ajax*. New Riders.
- KEITH, Jeremy (2005). *DOM Scripting*. Friends of ED.
- KRUG, Steve (2006). *Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition*. New Riders.
- ULLMAN, Larry (2008). *PHP 6 and MySQL 5*. Peachpit Press.

