

Supermarket Guidance in a PN System

Delphine Blondeau and Alexandre Gimenez

Published in June 2008.

INTELLIGENT MULTIMEDIA 10TH SEMESTER
Supermarket Guidance in a PN System

PROJECT PERIOD:
4th of february 2008 -
4th of June 2008

PROJECT GROUP:

1070

GROUP MEMBERS:

Delphine Blondeau
Alexandre Gimenez

SUPERVISORS:

Lars Bo Larsen
Alexandre Fleury
Rasmus L. Olsen

NUMBER OF COPIES: 4

REPORT PAGES: 130

APPENDIX PAGES: 49

TOTAL PAGES: 179

SYNOPSIS:

This report describes the development of a whole "Intelligent Supermarket" System. Similar systems have already been implemented in several supermarkets but the main difference with our project is that it is totally user-centric whereas the other ones are supermarket-centric.

This project uses Magnet beyond technology, which is a research project based on Personal Network concept. The user is able to interact with the system thanks to his mobile device (Nokia N800 or N810).

Our system can be divided into four parts. Firstly, the shopping list that the user can input into a specific application of his mobile device. This shopping list is shared between all the members of his family who have their own mobile device.

The second part concerns the guidance of the customer inside the supermarket.

Thirdly, the user is able to order the products he wants and to prepare his own bill by scanning the barcodes of the items.

Finally, at the end of the shopping, the user will get a virtual waiting line ticket. If this person is a handicapped person or a pregnant woman, he will be considered as a priority person.

Our objectives are to make an analysis of this system, to design an ergonomic interface, to perform usability tests and finally to implement a prototype of this system to be able to achieve a complete demonstration of our "Intelligent Supermarket".

We succeeded in making a usable system. The user can build his own shopping lists, being guided in the supermarket and the waiting queue handling is working properly. We also ran some usability tests and some tests of the final system in order to improve it the most possible. Unfortunately, the system is a bit slow and we did not have enough time to implement the scan of barcodes and the use of Magnet Technology to share the shopping list between the members of the family.

Contents

Abstract	13
Dedication	15
Preface	17
Introduction	19
Motivation	19
Problem Statement	19
Main Objectives	20
1 Project Presentation	21
1.1 Purpose of the Study	21
1.2 Scenarios	21
1.2.1 Shopping list	22
1.2.2 Arrival at the supermarket	23
1.2.3 Supermarket guidance	23
1.2.4 Waiting queue handling	24
1.3 Delimitation of the project	25
1.3.1 Update of the system	25
1.3.2 Location of the user in the supermarket	25
1.3.3 Choice of certain products	25
1.3.4 Queue Handling	26
I Analysis Part	27
2 Pre-Analysis	29
2.1 Personal Network and Magnet Technology	29
2.1.1 PN Organization	29
2.1.2 PN Federation (PN-F)	30
2.1.3 Context and Services	31
2.1.4 PN, Magnet and our project	31
2.2 Tablet PC and Maemo	34
2.2.1 Tablet PC	34
2.2.2 Development tools for Nokia Internet tablets	34
2.3 Existing Intelligent Supermarkets	36
2.3.1 Intelligent Shopping Assistant: The Shopping Buddy	36
2.3.2 Concierge System	36
2.4 The location of the customer in the supermarket	37
2.4.1 GPS	37
2.4.2 RFID technology	38
2.4.3 Bluetooth	39
2.4.4 Magnet Location System	40
2.5 The automatic bill	41
2.5.1 RFID technology	41
2.5.2 Barcode technology	41

3	Analysis	45
3.1	User definition	45
3.2	Use cases and conceptual Model	45
3.2.1	Shopping List Manager	46
3.2.2	Guidance System	47
3.2.3	Queue Manager	48
3.3	Tasks analysis and inherited features	49
3.3.1	Shopping List Manager	49
3.3.2	Guidance System	49
3.3.3	Queue Manager	49
3.4	Ideal System	50
3.5	Real System	51
II	CHOICES AND IMPLEMENTATION	53
4	design	55
4.1	Global Architecture	55
4.1.1	The application on the mobile device	55
4.1.2	The Supermarket System on the server	56
4.2	Application on the mobile device	57
4.2.1	Shopping List Manager	57
4.2.2	Guidance in the supermarket	65
4.2.3	Checkout	68
4.3	Supermarket System - the server	70
4.3.1	Guidance in the supermarket	70
4.3.2	Checkout	78
4.4	Supermarket System - the check-out application	80
5	Implementation	83
5.1	Choice of development language	83
5.1.1	Mobile programming	83
5.1.2	Server programming	85
5.1.3	Connexions between the server and the mobile application	85
5.1.4	Database	86
5.2	Application on the mobile device	86
5.2.1	Shopping list Manager	86
5.2.2	Guidance in the supermarket	90
5.2.3	What about the localisation of the customer?	92
5.2.4	When is the shopping finished?	92
5.2.5	Checkout	93
5.3	Supermarket System	94
5.3.1	Common class: the Database Manager	94
5.3.2	Guidance	96
5.3.3	Checkout	98
5.4	Supermarket System - the check-out application	98
6	Tests	101
6.1	Usability tests	101
6.1.1	Usability tests and user centred design	101
6.1.2	Low-Fidelity Prototype	101
6.1.3	Organization of the tests	102
6.1.4	The tasks list	104
6.1.5	Results of the tests	114
6.1.6	Results of the survey	117
6.1.7	Decisions	117
6.1.8	New Low-Fidelity Prototype	120

6.1.9	Conclusion of the usability tests	120
6.2	Tests of the implemented system	121
6.2.1	Organization of the tests	121
6.2.2	The tasks list	124
6.2.3	Results of the tests	125
6.2.4	Results of the survey	127
6.2.5	Conclusion of the tests of the implemented system	128
7	Conclusion and perspective	129
7.1	Conclusion	129
7.2	Possible Improvement	129
7.3	Perspective	130
III	APPENDIX	131
A	Extra Documents	133
A.1	Usability tests	133
A.1.1	Presentation of Usability tests	133
A.1.2	First Low-Fi Prototype	137
A.1.3	Updated Low-Fi Prototype	150
A.1.4	Answers to the survey	152
A.2	Tests of the implemented system	156
A.2.1	Introduction text	156
A.2.2	Answers to the survey	159
A.3	Implementation of the saving of shopping lists	167
A.4	Design of the map display via xpm data	170
A.5	Code of the server - guidance	171
A.6	Database	172
B	Glossary	175
C	Bibliography	177
C.1	Personnal Network and Magnet Technology	177
C.2	Tablet PC	177
C.3	Intelligent supermarkets	177
C.4	The location of the customer in the supermarket	177
C.5	The automatic bill	177
C.6	Choice of development language - Mobile Programming	178
C.7	Design and implementation of the mobile application	178
C.8	xmlrpc	178
C.9	The server	178
C.10	The report: building and orthographe	178
D	References	179

List of Figures

2.1	Personal Network [PN/IMAGE])	30
2.2	Personal Network Agent [PN/Agent/IMAGE])	31
2.3	Personal Network Federation [PN/FED/IMAGE]	32
2.4	An example of slate tablet PC: Nokia N800 [TabletPC/N800]	34
2.5	An example of convertible tablet PC: Nokia N810 [TabletPC/N810]	35
2.6	The Concierge System [CONCIERGE]	37
2.7	a RFID tag [RFID/TAG]	38
2.8	Barcode with 'Lars Bo Larsen' encoded in code 128B	41
2.9	Barcode with 'Lars Bo Larsen' encoded in code 128A	42
2.10	Barcode with 'Lars Bo Larsen' encoded in code 39	42
2.11	2D barcode with 'Lars Bo Larsen' encoded in Aztec code	42
2.12	2D barcode with 'Lars Bo Larsen' encoded in Codablock-F code	42
3.1	Use case - Shopping List Manager	46
3.2	Use case - Guidance System	47
4.1	Global Architecture	56
4.2	Shopping list manager	58
4.3	Sequence Diagram: Launch application and select the shopping list	59
4.4	Sequence Diagram: Remove a product	59
4.5	Sequence Diagram: Remove a product when no product has been selected	60
4.6	Sequence Diagram: Edit notes	60
4.7	Sequence Diagram: Edit notes option when no product has been selected	61
4.8	Add a product screen	61
4.10	Sequence Diagram: Lists Manager	62
4.11	Sequence Diagram: Lists Manager when the current list has not been saved	62
4.9	Sequence Diagram: Add a product in the shopping list	63
4.12	Sequence Diagram: Save the shopping list	64
4.13	Sequence Diagram: Exit	64
4.14	Sequence Diagram: Exit when the current list is not saved	65
4.15	Sequence Diagram: Initialization of the connexion	65
4.16	Next Area Screen	66
4.17	Shopping List Screen	67
4.18	Vegetables Area	67
4.20	Check-out - Waiting ticket 5	68
4.19	Sequence Diagram: Guidance in the supermarket	69
4.21	Check-out - Waiting ticket 0	70
4.22	Sequence Diagram: User and checkout	70
4.23	Sequence diagram: Initialization of the connection between client and server	71
4.24	Sequence diagram: Retrieving of the products of the shopping list	72
4.25	map with the directions to an area	72
4.26	Sequence diagram: getting the map	74
4.27	Zoomed map	75
4.28	Sequence diagram: getting the zoomed map	75
4.29	Map of an imaginary supermarket	76
4.30	References of products in the database	77
4.31	References of category in the database	78

4.32 Shopping list in the database	78
4.33 Sequence diagram: check-out	79
4.34 Sequence diagram: check-out and client	80
4.35 Check-out in the database	80
4.36 Sequence Diagram: Checkout Assistant	81
5.1 Shopping List - Load a list	87
5.2 Shopping List - Home Page	88
5.3 Shopping List - Choice of category Screen	89
5.4 Shopping List - Choice of products Screen	89
5.5 Guidance in the supermarket - Go to area Screen	91
5.6 Guidance in the supermarket - Pick up products Screen	92
5.7 Guidance in the supermarket - Waiting in the queue	93
5.8 Guidance in the supermarket - End of the queue	94
5.9 GUI for the check-out assistant - check-out closed	99
5.10 GUI for the check-out assistant - check-out open	99
6.1 Shopping list manager	102
6.2 Guidance in the supermarket	102
6.3 Do you often go shopping?	103
6.4 How comfortable are you with the use of mobile devices such as PDAs, Smartphones or Tablet PCs?	103
6.5 Home Page with an empty list	105
6.6 Add a product screen	105
6.7 Add a product screen - Vegetable Selected	106
6.8 Add a product screen - Potatoes selected	106
6.9 Add a product screen - Potatoes added	107
6.10 Add a product screen - Carrots selected	107
6.11 Add a product screen - Carrots added	108
6.12 Home Page screen	108
6.13 Home Page screen - Carrots selected	109
6.14 Edit Notes	109
6.15 Home Page screen - Carrots selected	110
6.16 Save the list screen	110
6.17 Detection of the system	111
6.18 Go to a specific area	111
6.19 Pick up the products	112
6.20 Check the notes of Carrots	112
6.21 The list is empty	113
6.22 Number 5	113
6.23 Check out	114
6.24 Home Page with an empty list	115
6.25 Home Page with the notes	115
6.26 Guidance in the vegetables area	116
6.27 End of the guidance	117
6.28 Updated home Page with the table	118
6.29 Updated Guidance Home Page	119
6.30 Updated Screen for a specific Area	119
6.31 Updated Screen for a specific Area	120
6.32 Did you participate in the usability tests of this system	121
6.33 Do you often go shopping?	122
6.34 How comfortable are you with the use of mobile devices such as PDAs, Smartphones or Tablet PCs?	122
6.35 The library	123
6.36 The library transformed into a supermarket	123
6.37 A user having a hard time holding the "products" and the mobile device at the same time	126

A.1	Shopping List Manager - Home Page 1	137
A.2	Shopping List Manager - Home Page 2	137
A.3	Shopping List Manager - Home Page 3	138
A.4	Shopping List Manager - Home Page 4	138
A.5	Shopping List Manager - Add a Product 1	139
A.6	Shopping List Manager - Add a Product 2	139
A.7	Shopping List Manager - Add a Product 3	140
A.8	Shopping List Manager - Add a Product 4	140
A.9	Shopping List Manager - Add a Product 5	141
A.10	Shopping List Manager - Add a Product 6	141
A.11	Shopping List Manager - Add a Product 7	142
A.12	Shopping List Manager - Edit Note	142
A.13	Shopping List Manager - Save List 1	143
A.14	Shopping List Manager - Save List 2	143
A.15	Shopping at the Supermarket - Detection of the system	144
A.16	Shopping at the Supermarket - Next Area Vegetables	144
A.17	Shopping at the Supermarket - Vegetables Area	145
A.18	Shopping at the Supermarket - Vegetables Area - Potatoes selected	145
A.19	Shopping at the Supermarket - Vegetables Area - Carrots selected	146
A.20	Shopping at the Supermarket - Next Area Drinks	146
A.21	Shopping at the Supermarket - Drinks Area	147
A.22	Shopping at the Supermarket - Drinks Area - Coke selected	147
A.23	Shopping at the Supermarket - End of the guidance	148
A.24	Queue Manager 1	148
A.25	Queue Manager 2	149
A.26	Queue Manager 3	149
A.27	Updated Shopping List Manager - Home Page	150
A.28	Updated Shopping List Manager - Home Page - Product Selected	150
A.29	Updated Shopping at the Supermarket - Next Area	151
A.30	Updated Shopping at the Supermarket - Area	151
A.31	Updated Shopping at the Supermarket - Shopping list screen	152
A.32	Answers to "The application is easy to use"	152
A.33	Answers to "The general design of the program is good"	153
A.34	Answers to "It is easy to browse through the application without getting lost"	153
A.35	Answers to "The functioning of the program is logic and easy to remember"	153
A.36	Answers to "The application is enjoyable to use"	154
A.37	Answers to "The application seems to be reliable"	154
A.38	Answers to "It is easy to build the shopping list"	154
A.39	Answers to "The way of building the shopping list is the best one"	155
A.40	Answers to "It is easy to understand the directions inside the supermarket"	155
A.41	Answers to "A map is a good way of providing directions"	155
A.42	Answers to "The way of providing directions in the supermarket is the best one"	156
A.43	Answers to "It is annoying not to be able to see the whole shopping list and to modify it while shopping"	156
A.44	Answers to "The application is easy to use"	159
A.45	Answers to "The general design of the program is good"	159
A.46	Answers to "It is easy to browse through the application without getting lost"	160
A.47	Answers to "The functioning of the program is logic and easy to remember"	160
A.48	Answers to "The application is enjoyable to use"	160
A.49	Answers to "The application works properly"	161
A.50	Answers to "It is easy to build the shopping list"	161
A.51	Answers to "It is easy to understand what I have to do in the supermarket"	161
A.52	Answers to "It is better to shop with this system than to shop on my own as I usually do"	162
A.53	Answers to "It is easy to shop and to hold/interact with the tablet PC at the same time"	162
A.54	The library	163
A.55	The library	163

A.56 A tester picking up a product	164
A.57 The application running on the tablet PC	164
A.58 A tester and the facilitator	165
A.59 A tester having difficulty to interact with the mobile device while holding the "product" .	165
A.60 A tester confused by mistakes on the detailed map and struggling to find a product	166

Abstract

Our project is the development of a whole "Intelligent Supermarket" System. Some of those systems already had been implemented in several supermarkets but the main difference with our project is that it is totally user-centric whereas the other ones are supermarket-centric. That means that our system is handling every shopping task of the user from the very beginning (the shopping list) until the end (the payment), always by trying to make easier the shopping time of the user. It is also using the user own devices, such as tablet PC.

Technologies used: This project needs a very high connectivity capacity for the communication between the different mobile devices themselves and with the supermarket system. That is why we decided to base it on Magnet Beyond technology which is a research project investigating Personal Network concept. The user is able to interact with the system thanks to his mobile device (Nokia N800 or N810). The application on the mobile device is developed in Python language. The server of the supermarket is in Java language.

Our system can be divided into four parts. Firstly, the user can input his shopping list in the mobile device. An application allows him to make it easily and quickly thanks to a predefined list of categories and products. One important characteristic of that part is that this shopping list is shared between all the members of the family who have their own mobile device, so that they all have the updated shopping list at every moment.

The second part concerns the guidance of the customer inside the supermarket: the mobile application gives him directions to every product present in his shopping list from his current location.

Thirdly, the user can scan the barcodes of the products he wants to buy with the camera embedded in his mobile device. In that case, he does not have to take a cart, he will get all the products he scanned when he will go to the cashier.

Finally, at the end of the shopping, the user selects that he wants to go to the checkout at that moment, he will get a virtual waiting line ticket. He does not have to wait in a line and the application will tell him when it will be his turn to go to the checkout. If he previously scanned every item he wanted to buy, an employee of the supermarket will bring a cart full of those and the checkout will directly know how much the customer needs to pay.

Dedication

We would like to thank every person who helped us during this semester, and particularly:

- Our three supervisors: Lars Bo Larsen, Alexandre Fleury and Rasmus L. Olsen. Through their experiences in PN Systems, in Magnet technology and in project management in general, they really helped us with advices, ideas and inputs.
- Charlotte Skindbjerg Pedersen, the IMM secretary, who is always very helpful for our student life at the university.
- The six students who accepted to participate in the usability tests of our system
- The librarian who allowed us to run our tests in the library
- The seven students who accepted to participate in the final tests of our system

Preface

This report is the project thesis report of the 10th semester project in Intelligent Multimedia at Aalborg University. It has been written by the project group IMM 08gr1070 during the first semester of 2008.

This report contains the overall description of the project and the choices made to realize it. The intended audience is not expected to have any prior knowledge about mobile systems, servers, or Human-Computer Interaction, but a general development knowledge is still recommended.

Figures and tables are marked by primary and secondary indexes. The primary index relates to the chapter number or the appendix letter, and the secondary one is a consecutive number for each chapter or appendix. For instance, Figure 1.3 refers to the third figure in the first chapter.

Report Outline

This document is structured with an introduction presenting globally the project, the motivation and the main objectives.

Then, the chapter Presentation of the project explains deeply the project with the scenarios and its delimitation.

Next, the report is divided into two parts, one which is only composed of the theoretical approach of the project and the other one which is a practical approach. The Analysis part is separated into two chapters: the pre-analysis with the description of every technology that interested us and the analysis part with mainly the description of the user, the ideal system and the real system.

The Choices and Implementation part describes the design and implementation of the system. There are also the tests, usability and final ones, done in this part.

We are ending the report with the conclusion, possible improvements and the perspective for this project.

Finally, there is the Appendix with the extra documents, the glossary and the bibliography.

Technology notice

This present document has been written using Latex and is published using TeXnicCenter software.

The images in this document have been retouched with the Gimp. The graphics have been made with Microsoft Office Visio Professional and Visual Paradigm.

The server application has been developed in Java, using the Eclipse environment.

The mobile application has been developed in Python, using Python2.5 and PyGTK library for the graphical elements. An Internet connexion was also required in order to communicate with the server.

The university lent us two mobile devices: a Nokia N800 and Nokia N810 that runs on Unix Operating System.

Authors and contributors

Content Writers The IMM 08gr1090 team, composed of Dephine Blondeau and Alexandre Gimenez.

Graphics, Style-sheets Dephine Blondeau, Latex.

Signatures:

Delphine BLONDEAU

Alexandre Gimenez

Introduction

Our second semester as a student in the Intelligent Multimedia Master (IMM) of Aalborg University is dedicated to our master thesis. For the first semester, we have been working on a mobile application allowing the user to get the shortest path by bus to his destination in Aalborg. As that project really interested us, we decided to continue with development of mobile applications for this master thesis in order to improve our knowledge in that domain.

AAU has currently several projects based on Personal Network. Personal Network (PN) is a new user-centric concept with a very high connectivity capacity. This allows the user to be able to interact with many connected devices surrounding him but also with people around him. Magnet beyond is a research project based on the Personal Network concept. Magnet beyond technology allows to create new systems or applications integrating directly the Personal Network concept. For more information about Personal Network and Magnet Technology, please refer to the section 3.1. Personal Network and Magnet Technology.

Motivation

Magnet beyond is a latest technology which seemed to offer unlimited possibilities of improvements for existing systems. More and more applications are based on Magnet Project which is getting more and more renowned. That is one of the reason why we decided to use this advanced technology in our master thesis.

Besides, Intelligent Supermarket concept also really motivated us because this is a system that we really would like to get for our own living.

Problem Statement

With the creation of supermarkets, the way of going shopping became something different: a lot faster, as we only have to go to one shop, but also a lot more stressful because supermarkets often are crowded and sometimes, products are difficult to find. That is why going shopping is considered as a chore by most people.

The values we are using below come from a survey ¹ published in February 2008 by the french magazine Lineaires ². We noticed that people are spending an average of 58 minutes everytime they go to the supermarkets. That is a lot for people who do not have much free time. Moreover, 64% of the customers are taking a cart to shop: this means that most of people are going shopping only when they have many things to buy, they are not going shopping more than once a week. Finally, only 46% of people like shopping. These values show that it is necessary to change the way of shopping, to help people to save time and to appreciate the moment when they are shopping.

To make this task easier, some supermarkets implemented specific technologies to help customers to achieve their shopping more quickly (cf. 2.3. Existing Intelligent Supermarkets). But we thought that these systems were not enough and we decided to implement another one which besides would integrate

¹470 recorded interviews in 10 supermarket

²www.lineaires.com

the PN concept.

The idea is that every member of the family should be able to input, into their own mobile device, the products they want to buy in a common shopping list. Then, when one of these people is going shopping, the supermarket system would be able to access to this shopping list and would guide him through the shelves. Finally, he will save some time at the checkout thanks to a specific waiting queue handling.

The main difference between our system and the system already implemented in some supermarkets is that ours is user-centric, whereas the supermarkets' ones are supermarket centric. We are changing the viewpoint: in one case, the user can only use the system in the supermarket, in the other one, he can use it whenever and wherever he wants thanks to the shopping list handling. Our system is one generation further!

Main Objectives

Our main objectives are:

- to build a system that will really help people to shop more easily and more quickly
- to implement an ergonomic user-centric interface
- to learn more about mobile application
- to get skills about integrating Magnet beyond into a heterogeneous system

1

Project Presentation

1.1 Purpose of the Study

Our project aim is to propose a whole heterogeneous system for an intelligent supermarket project. The first devices part of the system are mobile devices used firstly to input the shopping list and secondly to interact with the supermarket system. The shopping list can be shared within several people (family or flatmate for example) so that they can have one single list containing every item needed. The interface of this application should be user-friendly and easy to use. The advantages of that list is that the customers can add an item to it whenever and wherever they want.

The second part of the system is the supermarket system. When the customer enters the supermarket, it gets the shopping list. Thanks to that, the user can be guided through the shelves to every product presents in his list. The user does not have to know anything about the locations of the shopping departments, that is taken care by the supermarket. The path suggested by the supermarket system is the quickest way so that the customer can save time during his shopping. The directions are displayed on the user's mobile device. Firstly, he picks up the item he needs and then he uses his mobile device to scan its bar code. Finally, when the shopping is over, the customer asks his own application to check out and the supermarket service prepares automatically the bill.

As the waiting queue is also handled by the system, the user then waits until his application tells him that there is a checkout available. He does not have to wait in a real queue and can do whatever he wants during this waiting time. Besides, if he is a handicapped person, he will be treated as a priority person in the waiting queue.

1.2 Scenarios

The scenarios presented below allow to get a more precise view of the project and also to show more clearly the potential offered by our system. These scenarios are realistic and cover the whole possibilities offered by the project.

Søren is a danish man. He has got a mobile phone and he recently installed our application on this phone.

1.2.1 Shopping list

Context: Søren is at home, at work, or anywhere else. He wants to input his shopping list in his Supermarket Guidance System. The list is initially empty.

Note: Products are organized by categories and sub-categories. An example of category would be *Vegetables*; it would include sub-categories (which are assimilated as a product) such as *Carrots*, *Courgettes*, etc. The list is actually made of sub-categories, not real products: for example, if the user wants to buy the product *A bag of two kilos of carrots*, he will add the *Carrots* sub-category to his list. Then, he will be able to add notes to this item: for example *"a bag of two kilos"*.

- **Søren wants to add several vegetables in his shopping list**

Søren launches his application on his mobile device. The first screen is the shopping list screen, which includes every item of the current list (so far, the list is empty) and gives the user the following options: add a product, remove a product. To add a product in the list, Søren selects the *"Add a product"* option.

The next screen is the product selection screen. Søren can browse through the products tree and select any sub-category he wants, in order to add it to the list. As he wants to buy potatoes and carrots, he will select the *Vegetables* category, then the *potatoes* and *carrots* items. He may want to add specific information about the item, by adding a note, such as *"two kilos"* on the potatoes item.

This task of adding an item to the list can be repeated until the list includes everything Søren wants to buy. As he needs to buy some beer, he will find it under the Drink category and add it to the list. When he is done, he can close the application and the current shopping list will automatically be saved.

- **Søren wants to remove a product of his shopping list**

Søren realizes that actually, he still has some potatoes and so does not need to buy some more.

The screen is displaying the shopping list. Søren selects *Potatoes* in the list, then selects the *"remove item"* button. A confirmation is required before the item is actually removed.

- **Søren's wife wants to add a product to the shopping list**

Søren's wife owns a Magnetized device which belongs to the same PN as Søren. The Supermarket Guidance application is also installed on this device. She opens the application and will automatically have an access to the shopping list Søren made, as it is shared within the PN. She can modify the list as her device is authenticated; she adds *apples* to the current list.

- **Optional scenarios: Søren checks in which supermarket every product of the list is available**

Before actually going shopping, Søren wants to make sure he is going somewhere he will be able to find everything he needs. He executes the *"scan availability"* command.

The system establishes a connexion to every known supermarket (that is, every supermarket Søren has been to and used the application at in the past) and queries for the availability of all product. Within a few seconds, Søren will be able to see a list of supermarket where all the products are available. He can also look into detail and see what is missing in each supermarket. This will help

him choosing where he will go shopping.

- **Optional scenarios: Søren wants to shop from home**

Søren selects the "Order now" command. Then, he will input the address where he wants his products to be brought. The system will then display approximately at what time he will receive them. A lorry will then deliver him the wanted products.

=> Søren is now ready to go shopping at the supermarket.

1.2.2 Arrival at the supermarket

Context: Søren already input his shopping list in his Supermarket Guidance System. He is going shopping and is entering the supermarket.

When entering the supermarket, Søren's mobile device will discover the Supermarket Guidance service and will propose Søren to use it. Another part of the system is running on the supermarket's server and communicates with every devices of current customers, and also with the checkouts.

- **Søren accepts**

The device will send Søren's shopping list to the supermarket's system.

- **Søren refuses**

The supermarket system will not interact with Søren's mobile phone anymore during his current shopping session in the supermarket.

1.2.3 Supermarket guidance

Context: Søren is in the Supermarket expecting directions to start shopping.

The supermarket's system includes a map of the shop that includes every category and sub-category of products. Usually, products of one category can be found in one and only one area, but this is not always true.

The system will determine the closest area from the entrance where Søren needs to go and will then send proper instructions to his device. There are two levels of instruction displayed: a general one and another one specific to each area.

When Søren is not yet in the proper area, a global map of the shop (sent by the supermarket) is displayed by the system. The path to the destination area is shown on the map. In our example, we will consider the next destination is the Drink area.

- **Søren needs only one product in the Drink area**

Søren is tracked by the supermarket system thanks to RFID tags. When he enters a new area, a RFID tag is read and allows the mobile application to know it has arrived in the proper area. It switches to the second level of instructions: a zoom-in of the map is displayed, focused on the Drink area. The list item (beer) is indicated with a cross on the map. The item's name is also displayed on the screen, including the possible user notes. Through the interface, it is easily possible to see

the discounts available for products in the area.

When Søren has found the beers, he executes the Next product command. At that moment, the system will display again the large map of the supermarket with the location of the next area Søren needs to go to.

- **Søren needs several products in this Vegetable area**

When Søren arrives in the Vegetable department of the supermarket, the zoomed-in map shows that he needs two items: Carrots and Salad. Their locations are both symbolized in the same map by crosses.

After Søren took those two products, as for the previous part, executes the Next product command to continue his shopping.

- **Søren finishes his shopping**

When Søren put every product he needed in his cart, he is proposed to go to the cashier.

Alternatively, if Søren wants to stop shopping and go to the cashier without buying every item of his shopping list, he can also click on the *Finish* Button.

- **Optional scenario: Søren can scan the barcodes of the products**

To save some time at the checkout, with the barcode reader embedded in his mobile device, Søren scans the barcode of every item he wants to buy. In that way, the system is building automatically his bill before he is going to the checkout. Søren does not even need a cart, the employees of the supermarket will fill a cart with all the items he will have chosen at the end of his shopping before he can go to check out. Besides, Søren will know before going to checkout the amount of his bill, which can avoid potential embarrassment during checkout, and secondly, his bill will be ready to pay instantly when he will arrive at the checkout.

1.2.4 Waiting queue handling

The cashier's system

The cashiers use another application, part of the whole system, which deals with the waiting queue. This application can run on several different devices depending on the supermarket. In this project, it will probably run on the same device as the user's application.

All the cashier has to do is run the application when he opens the cashier, and then notify every time he is done with a customer and ready for the next one. He also can notify the system that he closes his checkout. The server will use this information in order to notify the customers which checkout they can use.

Søren's scenarios

Context: Søren finished shopping and wants to check out.

When Søren wants to check out, he is asked if he wants to go to a regular checkout or to a fast checkout, in case he is buying less than 10 products. Søren selects the proper one regarding the content of his cart.

If he selects the fast checkout, he will go through the waiting queue faster than if he selects the regular one.

- **Søren is a standard customer**

The screen displays the number of customers that will pay before him. This number will decrease everytime a customer is going to a checkout. This number can also be incremented if there is a

handicapped person or a pregnant woman coming to pay as they are priority people.

When it is Søren's turn, his device displays the number (identifier) of the checkout he needs to go to pay.

Finally the system displays a Goodbye message and Søren can close the application.

- **Søren is a handicapped person**

As an handicapped person, Søren is a priority person. As soon as a checkout is available, he is invited to pay. The same process is applied to pregnant women or families with kids. This parameter is set in the options of the mobile application.

- **Optional scenario: Søren used the scanning of products barcodes**

If Søren decided to scan every product he bought, he does not have to empty his cart, which was prepared by the employees of the supermarket, but only to pay at the checkout the bill that was automatically created.

1.3 Delimitation of the project

1.3.1 Update of the system

Our system could be improved later on by some updates. Firstly, when there are new products available in the supermarket or when there are some products not anymore available, they should appear or disappear of the application. Secondly, regarding to the server, when the arrangement of the supermarket changes (location of the shelves for example), the modification should be taken into account in the map of the supermarket. And finally, when the products are moved in the supermarket, their locations should be updated in the server.

These updates are not yet developed in our final prototype but in a future improvement, the program will check from time to time (a survey needs to be done to find the best timing) if there is an update needed. We decided not to do it everytime the application is launched because it would take too much time as the user maybe only needs to add one product in the list, it needs to be done quickly. That means that if there is a change in the products or in the arrangement of the supermarket, there will maybe be a difference between the current configuration of the application and the real supermarket before the next update of the system.

1.3.2 Location of the user in the supermarket

To locate the user in the supermarket, we are using a specific technology the "Magnet Location System". This technology is currently tested in different laboratories in Aalborg University. That means that we are not sure about the actual accuracy of this system and about the delay to receive the location.

1.3.3 Choice of certain products

With the barcodes system, the user does not have to take a cart with him but only to scan the barcodes of the products he wants. That brings a problem for some kind of products that generally, the user wants to choose himself. For example fruits, vegetables or cheeses.

The case when the user wants to use both system cart for the products he prefers choosing himself and barcodes for the others is not handle yet. So with the current system, he cannot choose himself these kinds of products.

1.3.4 Queue Handling

Some parameters in the queue handling specify whether the user is a priority person or not. He can be either a handicapped person or a pregnant woman. There is obviously no way to check on this characteristic in the mobile application so some people could try to cheat and to input that they are priority person whereas they are not, in order to go more quickly to the cashier. But the problem can be handle at the cashier lever as the check-out assistant can check on the status of this person.

Part I

Analysis Part

2

Pre-Analysis

2.1 Personal Network and Magnet Technology

Magnet Beyond, a continuation of the Magnet project, is a worldwide R&D project in which 30 partners among 15 countries are involved. Aalborg University is one of those partners, among other universities, research centers, industrial partners and SMEs.

The context of the project is as follow.

As stated on the official website, the purpose of the project is "to improve the quality of life for the user, and this will be done by introducing new technologies that are more adapted to the needs of the user"¹. People own more and more high-technology devices, mobile or not, such as computers, mobile phones, smartphones, PDAs and many more we could imagine in the future (fridges, cupboards, etc.). Those devices are able to communicate one or more interfaces such as Internet-on-air (Wifi, GPRS), bluetooth, infrared connexion, etc. Also, different devices provide different services: a computer may be able to edit documents, a mobile phone may be able to place a call, etc. Unfortunately, it's not possible, from a given device, to use a service provided by another device. However, it is technically an achievable challenge: this is what Magnet Beyond is all about.

2.1.1 PN Organization

The purpose of the Magnet Beyond project is to create a user-centric network architecture that would allow the user to easily use a service available within his Personal Network (PN). The Personal Network (cf. Figure 2.1) is the network that connects every device of one specific user (hence the PN is user-centric). A PN is made of clusters, which are Personal Area Networks (the home cluster which is made of the devices that are located at home, the office cluster, etc.), which are interconnected, in most cases through Internet, or within another PN. A Personal Area Network (or PAN) is a computer network made of devices such as computers, smartphones, PDAs, etc. The devices can be connected over-the-air (with bluetooth, infrared connexion, etc.) or in a wired way. Usually, the range of a PAN is of a few meters.

The fact that two devices are not part of the same cluster should not be a barrier for the use of the service; ideally, which device provides a certain service and where it is located should be transparent to the user. The type of connectivity is also transparent: wifi, GPRS, bluetooth... The system determines among the available types of connectivity, which one is the best, by considering various parameters including bandwidth, price, etc.

¹<http://www.ist-magnet.org/technicalapproach>, may 2008

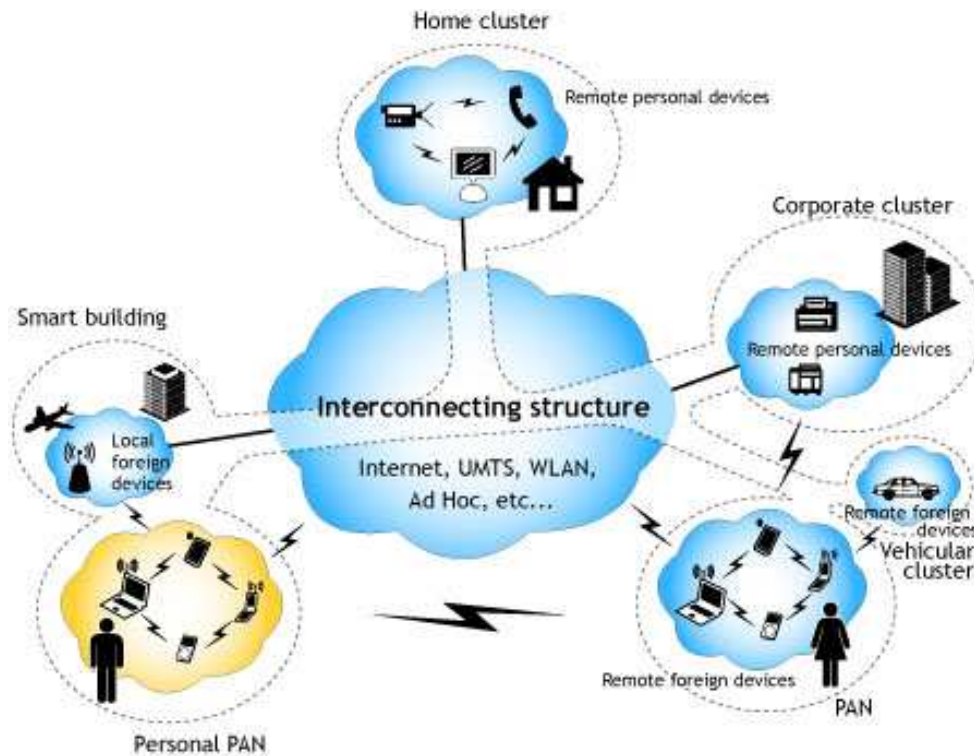


Figure 2.1: Personal Network [PN/IMAGE])

Some devices of the PN are also nodes. A node is a communicating entity which implements the IPv4 and/or IPv6 protocol. Each cluster has a Gateway Node which allows the communication with other clusters in the PN, usually through Internet.

Finally, there must be a PN agent available at any time to make the PN functional. The PN agent (cf. Figure 2.2) acts as a server that enables the different clusters to interconnect, by using gateways.

2.1.2 PN Federation (PN-F)

A PN Federation is a federation of Personal Networks that are interconnected and share services (cf. Figure 2.3). There can be public PNs; for example, a device which belongs to a private PN could be able to use a service of a device that belongs to a public PN, if they are part of the same PN-F.

On the connectivity point of view, a PN-F works the same way as a PN: it is also made of clusters which are interconnected through Gateway Nodes.

The previous figure shows how several PNs can merge into a PN-F. Let's have a closer look at this figure and take the example of User 1, who works in a Hotel. His PN is made of two clusters: his Hotel Cluster and his Home Cluster (inside the red area). But he belongs to a PN-F which includes User 2 and User 3: probably they are friends and they decided to merge their own PNs in a larger PN-F so they all have an access to services belonging to other PNs. As a result, User 1 has an access to the public services within the orange area, a larger area that represents the PN-F.

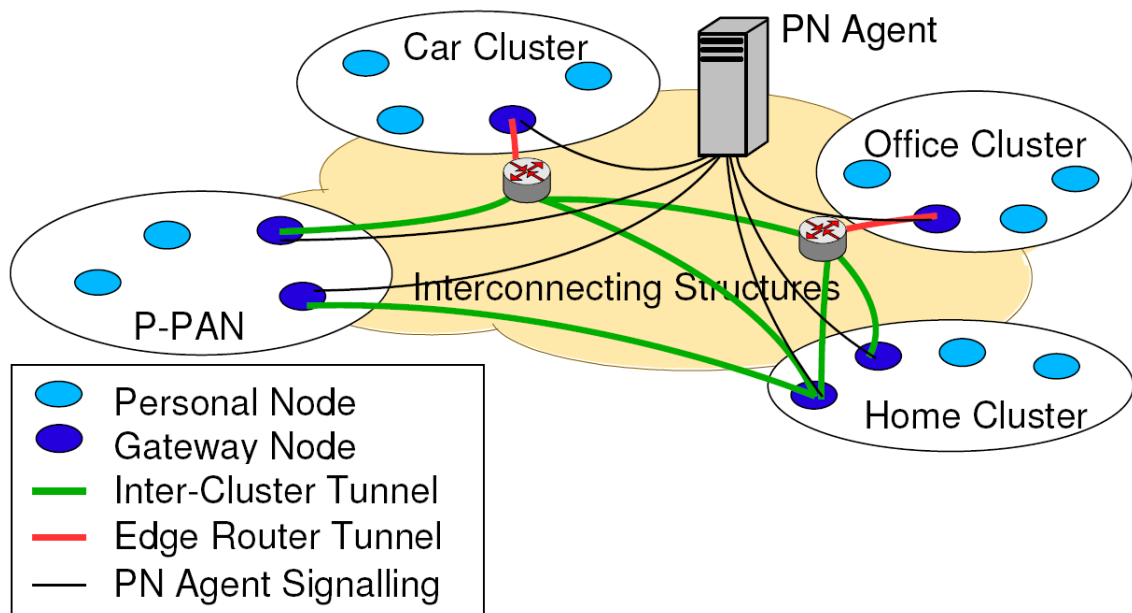


Figure 2.2: Personal Network Agent [PN/Agent/IMAGE])

2.1.3 Context and Services

The Magnet system is highly context-sensitive. Actually, the context is such an important part of the Magnet system that it has its own framework, the Secure Context Management Framework (SCMF). The SCMF is made of several context agents that runs on all nodes, and can keep track on the context of the PN such as the status of the devices, the services they provide, etc.

Services are a very important part of the Magnet project. A service can be of many different types, can be run on various devices and can be offered to a user depending on the context, thanks to the Service Discovery process. The Service Discovery is run within the PN and enables the user's device to be aware of which services other devices of the PN or the PN-F can offer.

Actions can be initiated depending on the context. For example, in our project, we can imagine that the user is in the same PN-F as the supermarket server, which is part of a public PN. When the user's device is detected, the fact that the Supermarket Guidance System is installed on the mobile device of the customer will decide if the user will be offered to use the Guidance System service to help him shopping.

2.1.4 PN, Magnet and our project

We can see that our project fits with the concept of PN network and Magnet services.

Our system will run on an Internet tablet, which will be a device that belongs to a user, and is part of his PN.

No matter which device is used to create the shopping list, it will be saved within the PN which means that it can be read and modified by another device which is part of the same PN. This feature is useful if someone within the user's family wants to add something to the family's shopping list: he can use his own device as long as he is part of the same PN, and he probably is.

About the shopping part, a supermarket would be part of a public PN-F that any user can join. Each supermarket part of this PN-F would propose a Magnet service, which consists in helping the user shopping. The user will be asked if he wants to use this service depending of the context: he will only be

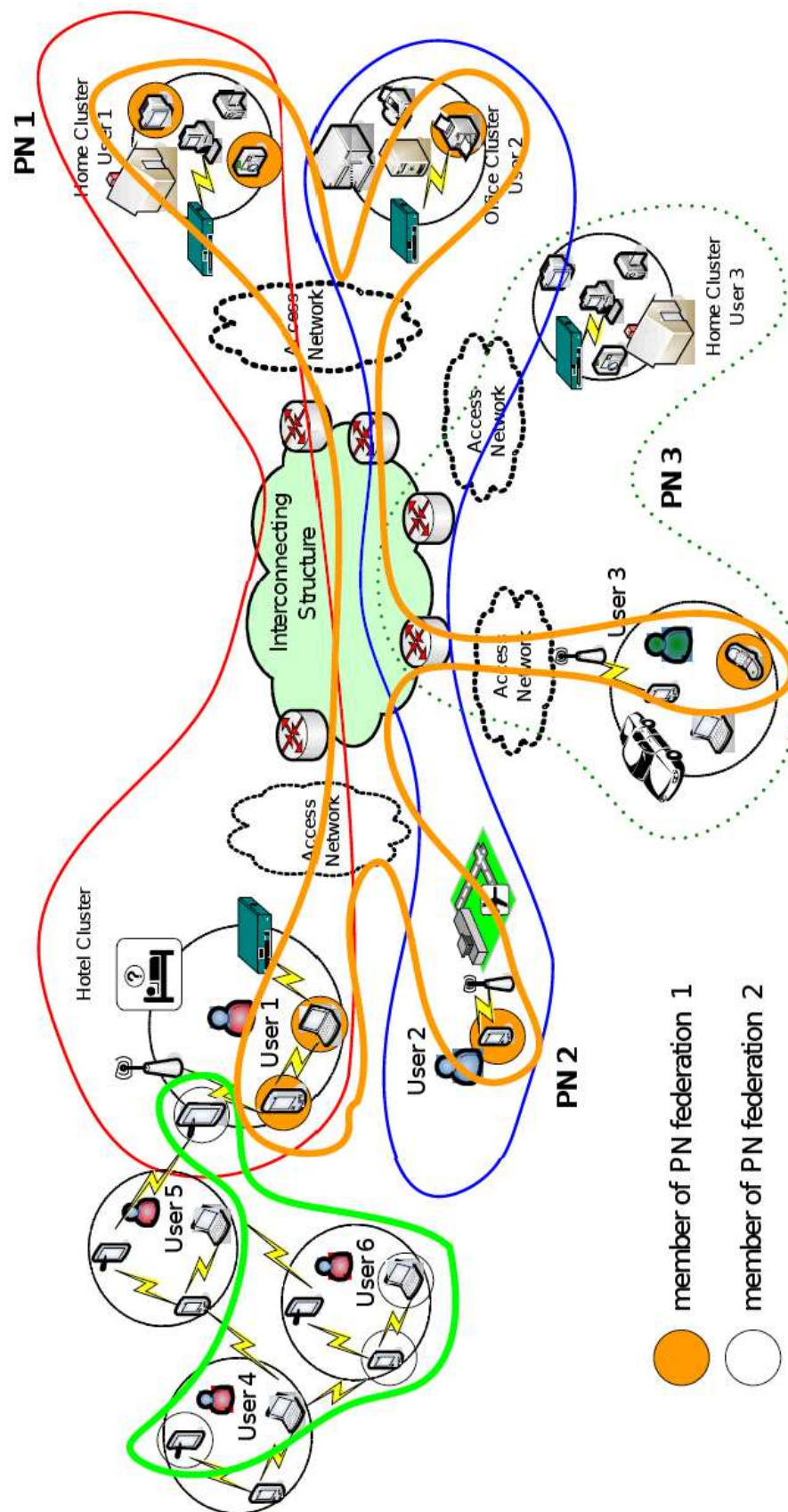


Figure 2.3: Personal Network Federation [PN/FED/IMAGE]

asked if he enters a supermarket and if he has the application installed on his device. This relies on the context sensitivity feature of Magnet services.

2.2 Tablet PC and Maemo

2.2.1 Tablet PC

Tablet PC has been invented in the early 90s by GRiD Systems but it was popularized only in 2002 by Microsoft with the launching of Windows XP Tablet PC Edition.

Tablet PC can run on many different operating systems (Windows or Unix Kernel). It is a kind of notebook, a slate-shaped mobile computer. It is equipped with a stylus in order to allow the user to interact with it; this stylus replaces the mouse used for classical computers (the user can also use his own finger). Some tablet PCs also allow the interaction through vocal recognition. People sometimes assimilate tablet PC to PDA (Personal Digital Assistant) but they are not the same. Firstly there is a difference about the screen; tablet PC's screen is usually bigger and in landscape format whereas PDA's screen is smaller and in portrait format, but above all, the main difference is that tablet PC is able to run every kind of software depending on the operating system installed on it, whereas PDA only is able to run some specific ones.

There are mostly two features of tablet PC: Slates and Convertibles.

- Slate tablet PC looks like traditional "writing slate", they do not have a keyboard. So there is a "virtual keyboard" appearing on the screen and the user is using the stylus on this keyboard in order to input some text. It is also possible to connect an external keyboard via wireless or USB connexion.



Figure 2.4: An example of slate tablet PC: Nokia N800 [TabletPC/N800]

- Convertible tablet PC have a keyboard attached, it usually can be slided under the main screen. They are usually bigger and heavier than the slate ones.

For our project, AAU lent us two tablets PC: one Nokia N800 and one Nokia N810.

2.2.2 Development tools for Nokia Internet tablets

Maemo² is an open source development platform for Nokia Internet tablets that runs under Linux. Accessible to everyone, it allows the developer to work on a computer in an environment identical to the

²<http://maemo.org/>, may 2008

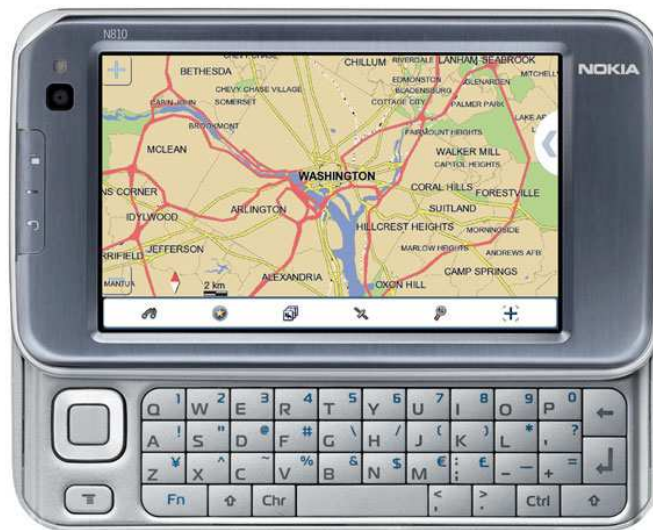


Figure 2.5: An example of convertible tablet PC: Nokia N810 [TabletPC/N810]

actual device. Several programming languages are supported by Maemo, such as C, Python or Java.

The Maemo SDK is a useful tool but moreover the developer will greatly appreciate how helpful the Maemo community is. Countless tutorials and other types of documentation are available for free, various secondary tools have been created for many purposes (such as using another programming language) and the forums are full of talented programmers devoted to help others.

All this makes developing on Nokia tablet PCs easy, when sometimes it is very complex to even get started in developing on a mobile device.

2.3 Existing Intelligent Supermarkets

Here are two examples of existing intelligent supermarkets that inspired us at the beginning of the project, this list is not exhaustive but representative of this kind of system. They allowed us to find the best way of helping customers in the supermarkets. We noticed many of existing intelligent supermarkets are quite similar.

2.3.1 Intelligent Shopping Assistant: The Shopping Buddy

The Intelligent Shopping Assistant was created in 2004 by IBM. This system can be adapted to every supermarket. The Shopping Buddy is the one implemented in Stop & Shop grocery stores in the USA. It is composed of a cart with a tactile screen (tablet PC) which is able to guide the customer regarding to his shopping list. The guiding is possible thanks to beacons placed on the shelves of the supermarket. Three procedures allow the system to know the content of the shopping list, or at least to predict it:

- The customer can send an email to the supermarket with his shopping list. In that case, it is also possible for him directly to get a cart full of the products he wants to buy.
- The customer can input the name of the wanted product.
- The user slides his client card in the cart. This card contains his previous bought products and also his favorite ones.

The user also has the possibility to prepare his own bill by scanning the bar codes present on the products he is buying. A bar codes reader is part of the cart.

This system, initially created to make easier the shopping experience of the customer, is supposed to boost the sales of the supermarket. It is due to the fact that it is developing loyalty but also because the cart is conjointly displaying promotions and product reminders thanks to a high targeted program ³.

2.3.2 Concierge System

The company Springboard Networks in Canada created Concierge. It is a cart with a GPS and LCD tactile screen embedded which is able to tell the customer when he is approaching an item of his shopping list and also some promotions. This system is also handling an automatic payment thanks to a bar code reader installed on the cart to prepare the bill and a wireless payment by swiping a bank card. Moreover, the cart stores information every time a customer is shopping and will make suggestions to the client at the next shopping time ⁴.

Notes: We were really surprised to discover that this system is using a GPS device to locate the user whereas it is used indoor. We do not understand how it can communicate with the satellites and so, get the exact location of the cart in the supermarket. Our guess is that the brochure of Concierge system used in a wrong way the term "GPS" which actually meant a location system that is not specified...

³According to IBM brochure: <http://www-03.ibm.com/industries/retail/doc/content/bin/stop26shopfinal.pdf>, april 2008

⁴According to Concierge brochure: http://www.springboardnetworks.com/assets/resources/TechPoint_BR_2007.pdf, april 2008



Figure 2.6: The Concierge System [CONCIERGE]

2.4 The location of the customer in the supermarket

2.4.1 GPS

Definition of GPS

GPS means Global Positioning System. It was developed by the United States Department of Defence at the end of the 70's. It is working thanks to a constellation of orbit satellites that transmit microwave signals.

GPS receivers are composed of an antenna, a highly-stable clock and processors. Nowadays, many of those also have a screen to display its position on a map for the user who needs to travel.

A GPS receiver is able to calculate its location but also its speed, direction and time, with the signals of three or more satellites. Those calculations are based on the location of the satellites and on the atomic clock contained in each satellite whose time is compared to the time of the receiver's clock. That comparison allows the receiver to know its distance from this satellite. The receiver needs to do these computations at least three times to be able to triangulate its position. Generally, receivers need to do it more than three times to have a better accuracy as some clocks are not totally accurate.

How can we use GPS in our system?

If the mobile device of the customer is equipped with a GPS receiver, it could allow the mobile device to locate the customer in the supermarket on a map the supermarket system already provided. Unfortunately, GPS needs to be able to communicate with satellites so being in a supermarket, i.e. indoor, can really be a problem for these needed communications. Concierge System (cf. 2.3.2. Concierge System) locates the cart of the customers thanks to a GPS device embedded in the cart, so originally, it seemed possible to use that technology. Nevertheless, we could not find any information about how this GPS is working indoor and we guessed that Concierge System was not using GPS technology but another location system that was wrongly described in the brochure as "GPS technology" probably because it is a term known by most people...

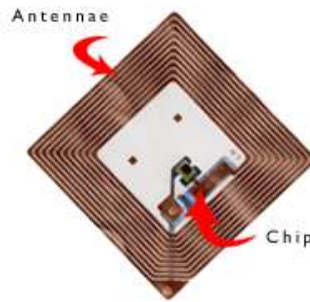


Figure 2.7: a RFID tag [RFID/TAG]

Which decisions did we take?

We decided that this technology was not appropriate to our system because supermarket is indoor and GPS devices are not supposed to work indoor...

2.4.2 RFID technology

Definition of RFID technology

RFID means Radio Frequency IDentification. This is part of automatic identification technologies. It is the successor of bar codes. Bar code allows to identify optically a product whereas RFID allows to store some more data. RFID is used to identify rapidly an object, an animal or a person that are tagged with a RFID tag. RFID is used with radio frequency signals. These frequencies are chosen depending on the size of the magnetic field needed.

There are three type of RFID chips. The low frequency tags (100-500 kHz) and middle frequency (10-15MHz) are working with energy coming from electromagnetic field that "wakes them up", they are also called the passive tags. They can be detected up to several centimeters for the low frequency tags and as far as 50-80 centimeters for the middle frequency tags. The third category of RFID tags is working on high frequency (2,4-5,8 GHz). This one can be detected up to several meters but it needs an internal battery.

A RFID system is composed of three parts:

- A scanning antenna
- A transceiver with a decoder to interpret the data
- A transponder (the RFID tag) that has been programmed with information

Radio-frequency radiation is the means of communication between the transponder and the scanning antenna but it is also the means of providing energy to the RFID tag so that it can communicate (for the passive RFIDs).

When a RFID tag passes through the field of the scanning antenna, it detects the activation signal from the antenna that "wakes up" the RFID chip and it transmits the information on its microchip to be picked up.

The guidance of the customer with RFID technology

We thought about using the RFID technology in the guidance of the customer in the supermarket. The mobile device of the user needs to be equipped with a RFID reader and RFID tags are put on every shelf

of the supermarket.

Notes: This reader could also be embedded into the shopping cart but as the final aim of the project is also for the user to get rid of the cart, we focused on embedding it into the mobile device.

Certain RFID readers are specific for PDAs or Internet tablets. This kind of reader generally has a length between 80 and 150 mm, a width between 40 and 110 mm and a thickness of 20mm. It costs around 2000 kroner.

After the supermarket system calculates the path that the customer should follow in the supermarket to take his products, it sends it to the mobile device of the user. The mobile device also has got a complete map of the supermarket and the relations between the shelves and the tags. So the mobile device is able to locate the customer in the supermarket by scanning the RFID tags around it put on the shelves, in this way, it can tell the user where he needs to go and which directions he should take.

What did we decide about that technology?

Firstly, we noticed that we could only use the active tags in our project as the passive tags do not have a distance capability large enough to allow the customers to be located easily in the supermarket.

We decided not to use it firstly because we would have needed to add a RFID reader to the mobile device and that kind of technology seemed to be complicated to be Magnetized⁵. Secondly, we thought that it would probably not be the most efficient because nowadays many products have their own RFID tags embedded which would have been also detected by our reader. Even if we could configure a filter in the reader, we thought that there still would be a waste of time for the user waiting for a reaction of his device.

2.4.3 Bluetooth

Definition of Bluetooth

Bluetooth was created by the Swedish producer Ericsson in 1994. It was invented in order to be a new way of communicating and exchange information between computers, printers, keyboards, mobile phones, etc. without a cable.

Bluetooth is working with radio waves. The bandwidth used is from 2.4 to 2.4835 GHz. There are three type of Bluetooth devices depending on their power and range:

- Class 1: 100mW with a range of 100 meters
- Class 2: 2,5 mW with a range of 10 meters
- Class 3: 1 mW with a range of 1 meter

The most common class is the third one.

How can we use Bluetooth in our system?

The mobile device of the user is already containing bluetooth technologies. The process of locating the user is quite similar to the one we thought about with the RFID technology. That means that by putting Bluetooth access point on every shelf of the supermarket, the mobile device can locate the user in the supermarket.

Which decisions did we take?

We decided that this technology was very appropriate for our project but after many hesitations with Magnet Location System (cf. 2.3.4 Magnet Location System), we decided to use the second one.

⁵In the whole report, the term "to Magnetize" means to set up the Magnet Technology for the concerned device

2.4.4 Magnet Location System

How does Magnet Location System work?

This system is a technology which is currently in R&D in Aalborg University.

There are the initial conditions:

- The person we need to locate is carrying a "target box".
- Three other boxes are placed strategically in the supermarket.
- A fourth one is connected to the server.

The target is always sending packages that the three other boxes receive. These three boxes take into account the signal strength of the packages received and then communicate with the one connected to the server which will triangulate the location of the target.

This system needs between 2 and 3 seconds to send the location to the server.

Note: The shelves are not blocking the packages sent.

Which decisions did we take?

We decided to choose this technology because the accuracy of these boxes corresponds to our needs but moreover because it already had been Magnetized in the past so that will be easier for us to implement it in our system in comparison to the other technologies. As this technology is currently in R&D, we also thought that it would be an interesting challenge to discover it.

2.5 The automatic bill

For the automatic bill part, we need that every product can be identified easily by the mobile device of the user. In that way, thanks to the connexion with the supermarket system, the mobile device will be able to store information about the chosen product and to get its price. Consequently, at the end of the shopping, it will be able to prepare automatically the bill.

2.5.1 RFID technology

Which use do we have for RFID technology in the automatic bill part?

Nowadays, more and more products have RFID tags to identify them. The general advantages of RFID tags are firstly that it does not need a direct contact with the reader and secondly, that it is able to store more information than bar code, the price of the product for example.

This second advantage allows to limit the number of requests needed with the supermarket system to get the prices of the products. Unfortunately, the first advantage did not particularly seem interesting to us because it could be the cause of many false positive if the reader is near from several products.

Which decisions did we take?

To avoid the false positives due to the reading of RFID tags without contact, we would need to take a specific care on reducing the most possible the range of the RFID tags. And, again, we were thinking about the problems which could occur with the integration of a RFID reader in our system.

2.5.2 Barcode technology

Classical barcode

Barcodes were invented in 1952 by Joseph Woodland and Bernard Silver. A barcode is an image of succession of parallel lines of different widths and spacings. It is used to store information and more generally, to identify the item tagged with it. Bar codes can be read by barcode readers, which are actually optical scanners, but also with special software.

The relation between the lines and the information is made with the code used at the creation of the barcode. There are several codes standards available to generate a barcode (Code 39, Code 128A, Code 128B, Code 128C), they are called symbologies. The choice of the code depends on the needs, some are only working with numbers information, some others can only take into account some symbols, etc. A bar code can be divided into five parts: a quiet zone, a start character, information characters, a stop character and a quiet zone.

There are some examples of symbologies:



Figure 2.8: Barcode with 'Lars Bo Larsen' encoded in code 128B



Figure 2.9: Barcode with 'Lars Bo Larsen' encoded in code 128A



Figure 2.10: Barcode with 'Lars Bo Larsen' encoded in code 39

2D barcode

As an improvement of barcodes, 2D barcodes were invented. They are also called matrix codes or 2D codes.

The rule is the same as the one for traditional barcodes but the information is represented in two dimensions as a grid of squares. 2D code is able to encode more information in a small space than classical barcode.

The images following are some examples of symbologies for 2D codes:



Figure 2.11: 2D barcode with 'Lars Bo Larsen' encoded in Aztec code

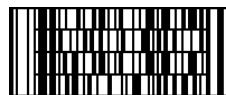


Figure 2.12: 2D barcode with 'Lars Bo Larsen' encoded in Codablock-F code

Which use do we have for barcode?

Bar codes can contain the references of a product and its price, so it makes possible the implementation of automatic bills. To use this technology, the user will need to have a barcode reader on his mobile device.

Some applications already exist to allow the reading of barcodes if a camera is integrated in the mobile device.

Which decisions did we take?

The advantages of using barcodes instead of RFID technology is firstly that the mobile device we are working on owns a camera which can be used as a barcode reader. That means that we do not need to integrate another component in our system whereas we need a RFID reader to use the RFID tags.

Secondly, our decision was also influenced by a matter of cost as a barcode is very cheap in comparison with a RFID tag. So we decided that we preferred using barcode.

3

Analysis

3.1 User definition

At the beginning of the project, we searched for information about the typical customers of the supermarkets to be able to adapt our application to a specific profile if there is one. The numbers and percentages we are using below come from a survey¹ published in February 2008 by the french magazine Lineaires².

The only prerequisite characteristic needed for the user of the Supermarket Guidance in a PN System is that he should own an Internet tablet.

The range of characteristic ages of people going shopping is mainly between 12 and 80 years old and the average age is 42 years old. These people are either men or women at almost the same proportion. That means that almost all the population could theoretically be using our application. But we noticed two points about the gender differences: 37% of men considers that shopping is a chore whereas this percentage is only 30% for women. Additionally, 33% of women and 37% of men refuse to spend more than 30 minutes shopping.

These percentages mean that our system is a little more important for men than women as men are the ones who want the most shopping quickly (7% more than women) and who are the majority who do not like shopping (4% more than women). But in another hand, as the differences between these results are not that important, only 7% and 4%, we finally decided to focus on both genders. However, in a perspective of commercializing our product, we realized that the advertisements should probably focus on males.

During the design of our system, we focused on building an application usable by any inexperienced users in mobile application. So after having defined the user, in order to achieve that design, we organized some usability tests (cf. 6.1. Usability tests).

3.2 Use cases and conceptual Model

For the use cases, we will divide them into three parts: shopping list manager, guidance system and queue manager.

¹470 recorded interviews in 10 supermarket

²www.lineaires.com, may 2008

3.2.1 Shopping List Manager

In this part, the main use cases of the Shopping List Manager will be detailed.

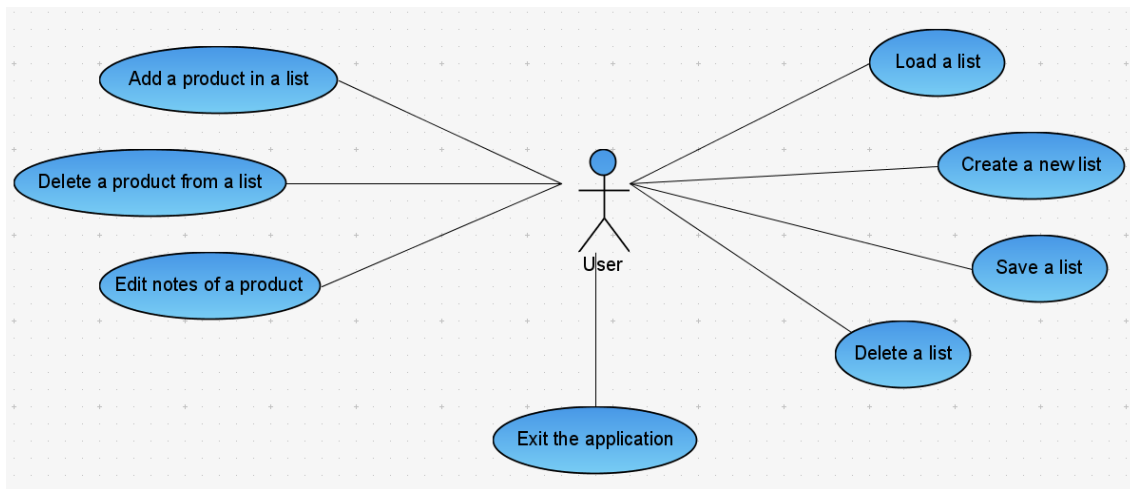


Figure 3.1: Use case - Shopping List Manager

The user wants to create a new list

The user selects the list manager and then selects "New list". The system checks if the current list has been saved, if not, it asks if the user wants to save it before loading the new list. If the answer is yes, it will be saved (cf. 3.2.1.4 The user wants to save the list) and then load the new list. Otherwise, a new list is displayed.

The user wants to add a product in the list

The user already has a shopping list open. He asks for adding a product. Then, he needs to specify the category of this product. Then he will choose the product in the list. He also can add some notes for this product (such as 2kg for example). Then he confirms that he wants to add it in the shopping list.

The user wants to delete a product

The user selects the product he wants to delete in the list displayed and then asks for deleting it. The system will ask for a confirmation. If he confirms, the product will be deleted. Otherwise, the product will remain in the list.

The user wants to edit the notes of a product

The user selects the product whose notes are the one he wants to see. Then he asks for editing them. If he modifies them, the system will ask for a confirmation. If he accepts, the notes will be changed. Otherwise, the notes will remain the same.

The user wants to save the list

When the user is done with the modification of his shopping list, he selects Save the list. The system will then ask him about the name of this list. At that moment, the user can still give up saving the list. The names of the pre-existing list are displayed so the user can choose them or create a new one. If he does

not want to give up, he has to confirm that he wants to save it.

The user wants to load another list

The user selects the list manager and then selects that he wants to load a list. The system checks if the current list has been saved, if not, it asks if the user wants to save it before loading the other list. If the answer is yes, it will be saved (cf. 3.2.1.4. The user wants to save the list) and then load the other list. Otherwise, every pre-existing list are displayed, the user has to choose one and to confirm that he wants to open it.

The user wants to delete a list

The user selects the list manager and then selects that he wants to delete a list. The system displays every pre-existing list on the screen, the user has to choose one and to confirm that he wants to delete it. He will then be back on the main screen.

The user wants to exit the application

The user can exit the application whenever he wants. If he selects that he wants to exit and if the current list has not been saved yet, the system will ask him if he wants to save it or not before exiting (cf. 3.2.1.4 The user wants to save the list).

3.2.2 Guidance System

In this part, the main use cases of the guidance system will be detailed.

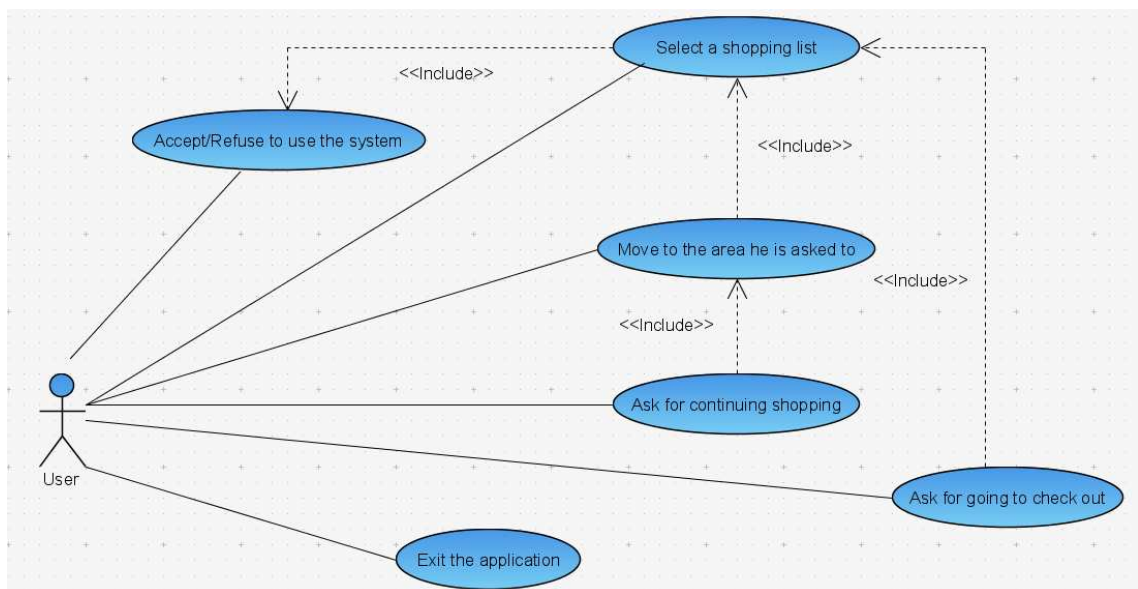


Figure 3.2: Use case - Guidance System

The user accepts to use the guidance system

While the user is entering the supermarket, the supermarket system detects that he has the guidance system application installed on his mobile device. The application is launched and asks if the user wants

to use it. If he wants to use the guidance system, he accepts.

The user refuses to use the guidance system

In the same conditions as in the previous part, if the user does not want to use the guidance system, he refuses. He can still launch it manually if he changes his mind.

The user selects the shopping list he wants to use for shopping

The first step is to choose the shopping list which he wants to use to shop. After the application is launched, the list of every available shopping lists is displayed and the user can choose the one he wants.

The user arrives at the area specified by the application

After the guidance started, it asks the user to go to a specific area (vegetable, meat, etc.). He is guided thanks to a map representing the whole supermarket with arrows to the wanted area. The system is following virtually the user thanks to a location system. When the user arrives at the right area, the system displays a new map representing the shelves of the area and showing where the products of this area that belong to the shopping list are situated.

The user wants to continue shopping

When the user took every wanted product of the area where he stands, he can ask for continuing shopping. If there still are some products in the list, the system will ask again to go to a specific area (cf. 3.2.2.4. The user arrives at the area specified by the application). If it is over, the user can finish shopping (cf. 3.2.2.6 The user finishes shopping)

The user finishes shopping

When the user finished shopping, he can ask to go to check-out.

3.2.3 Queue Manager

In this part, the main use cases of the Queue Manager will be detailed.

The user is a regular customer

When the user asks to checkout, the system checks on the status of the user. If he is a regular customer, he will get a waiting number corresponding to the number of people who will check-out before him. Everytime another customer is going to pay, his waiting number will decrease of one. When the number is zero, he can go to the checkout specified.

The user is a priority person

When the user asks to checkout, the system checks on the status of the user. If he is a handicapped person or a pregnant woman, he is a priority person. He will be the next one to check-out.

3.3 Tasks analysis and inherited features

The whole system is composed of many different tasks. They are divided into three parts: Shopping list manager, guidance system and queue manager.

3.3.1 Shopping List Manager

In the shopping list manager, the main tasks are the following ones:

List Handling

Firstly, the Shopping list manager needs to handle the shopping lists. At the beginning and at every moment the user can ask to open a list, a new one or a previous shopping list. It is also able to save a list and to delete one.

Building of a shopping list

The second main task of the Shopping list manager is to allow the user to build his shopping list. The user can add or delete a product and also add or delete a note in the list.

3.3.2 Guidance System

In the Guidance system, the main tasks are the following ones:

Choice of a shopping list

The Guidance System allows to choose the shopping list which the user will shop with in the supermarket.

Guide to the areas

Then, it is able to know from which department belong each product of the shopping list. For each category of products, it will guide the user through the shelves to the needed area. It will use a map and directions to guide the user. Some information will be stored in the mobile device. Some others will be sent by the supermarket system.

Show the products to take

Everytime the user arrives at a wanted area, the Guidance System will tell him which products he has to take in this department and where they are located in the shelves. Some information will be stored in the mobile device. Some others will be sent by the supermarket system.

Launch the Queue Manager

When the user wants to check-out, the Guidance system is able to launch the Queue Manager.

3.3.3 Queue Manager

In the queue manager, the main tasks are the following ones:

Check the status of the user

Firstly, it is able to check on the status of the user to know if he is a priority person or not.

Numbers handling

Secondly, it is able to ascribe a waiting number to the user regarding to his status (cf. 3.3.3.1. Check the status of the user) and to decrement it in relation to the number of people going to check-out.

3.4 Ideal System

This section describes our system as it could be designed in perfect conditions of technology, time and workforce.

The Supermarket Guiding System would run on any Internet tablet, not only on Nokia N800/N810, with a similar layout. The user would build a shopping list from a standardized list of products that could be updated automatically on every device of a PN-F when a new version is available in the PN-F.

A shopping list can be accessible either by the user from the devices that was used for its creation, or from any device that belongs to the user's PN, depending on if he chooses the list to be private or shared within the PN.

Once the list is established, an online routine will check if every product is available in the supermarkets that belongs to the user's PN-F, so the user can choose the place he is going to shop to accordingly. For example, when his shopping list is loaded, he could be displayed the logo of the nearby supermarkets that currently propose all the products of his list.

About the second part of the system, when the user enters the supermarket, the application would automatically be launched on his device, thanks to the context sensitivity that Magnet services can use.

Then, the supermarket will compute the shortest path for the whole shopping process, from the entrance of the user to the checkout, going through every area the user must visit to complete his shopping.

The user is able to modify the shopping list while shopping: the server adapts and recomputes the shortest path.

The user is equipped with a location system which allows the server to keep track on his location while he is shopping. This is useful for two reasons. Firstly, the server detects the arrival of the customer in the area where he was told to go to. For example, if the server asks him to go to the Breakfast area, it can also detects when he actually enters the Breakfast area. Then, the device will automatically go to the next step and show the zoomed map of the current area, with the location of the product the user wants to buy. The second reason is that if, for some reason, the user goes off his normal route and unexpectedly enters another area of the shop, then the server can compute the shortest path again and update the directions. Then, the user scans the bar code of the item he wants to buy. The scanning is done via the camera embedded in the tablet PC and a specific application installed on it. The user does not need to put any product in his cart; actually, he does not need to carry any cart. The system will note that the user wants to buy this specific product and will notify the server.

When the user is done shopping, he will ask to go to check out. At this moment, the server will ask an employee of the supermarket to fill in the customer's cart with the list of product he scanned inside the shop.

While the user is waiting for a checkout to become available, he does not have to stay in line; instead, he can wander in the shop or sit somewhere. He can have a good idea of how long he has to wait as he is

given his place in queue, so he knows how many people are ahead.

When it's his turn, the application indicates him to go to a specific check-out. There, he has to pay, which can be done either as usual, or by paying automatically if a mean of payment is recorded in his mobile device (for example, his credit card information).

Once he has paid, he can pick up a cart that has been filled with the products he bought (he may have to wait a little time until the cart is ready).

3.5 Real System

If today's technologies could eventually allow to develop such a system, it would take an amount of time we cannot spend for our current project. Anyway, the aim of the project is rather about creating a prototype which consists in a proof-concept.

This section will mostly emphasize on the differences between what we would like to do and what we actually did, and the reasons of those differences will be discussed.

The list of all products available in supermarkets is actually standardized, as there is only one version within the "world". However, it cannot be updated automatically, it has to be updated manually.

The Magnet storage system could allow the list to be updated automatically very easily; unfortunately, we were not able to use the Magnet packages because the project is at an early development step. Indeed, there is no such thing as a documented Magnet API that would allow us to easily use the Magnet capabilities. Instead, we can access the source code of the projects, whose documentation is far from perfection. In practice, this makes using the Magnet packages very difficult as we have to figure out how it works on our own. These difficulties became very obvious and that is why our prototype does not include the use of Magnet packages.

About scanning the bar code of a product instead of actually picking up the product, we feel that this is a very good idea, likely to be used in the real world in a near future. However, it technically implies a work in image processing, in order to read the bar code. Considering the general size of the project and the fact that there is only two people in the team, it was decided to stick to a more usual way of shopping, which allows us not to develop this image processing part, a costly part to realize in terms of work time.

The server provides the directions to the closest product, so the path is computed for every category, instead of computing it once for all at the beginning and make sure we give the user the very shortest path. There are two main reasons for that. The first reason is that this type of computation would take too long on the server as it uses a lot of computation power. The second reason is that it is not the main purpose of our project to optimize shortest path algorithms, so we are not going for the perfect algorithm here, just for a good one: the difference will be barely detectable anyway.

Also, we do not let the user modify the shopping list while he is shopping, because it is hard to keep the application simple, and we think simplicity is definitely what we are looking for as the user is already busy carrying around his cart.

The automatic payment is a similar function as we think it really could be implemented in the future, but implied a lot of extra work. Besides the cost in term of work time, there were also huge security issues that had to be dealt with. As security is a big part of the Magnet project, it will be possible to implement this function when the Magnet project will be a few steps further, which will make it easier to use.

Part II

CHOICES AND IMPLEMENTATION

4

design

This chapter will describe how theoretically our system was meant to be built. All these design decisions were taken after all the pre-analysis and many considerations. We already decided theoretically the technologies we will use to build our system.

Remember that the system which will be described below will surely have some differences with the one that will finally be developed. We tried to take into account every limitation given by the technologies we choose to use but we will probably run into some problems that we did not think about when we will start to build it.

4.1 Global Architecture

The Supermarket Guidance System is divided into two main entities, which are made of a few subsystems.

An entity is a part of the system that runs on a specific device.
The entities of the global systems are:

1. The application on the mobile device
2. The Supermarket System on the server

Both entities sometimes need to communicate with each other, especially inside the supermarket. This is done via a wifi connexion (the one of the supermarket), using XML-RPC protocol. It is technically possible to use many other types of connexion instead of wifi: Internet could be an option.

The next sections describe each one in detail.

4.1.1 The application on the mobile device

The client is an entity that runs on the customer's device. It runs in Python.

The client is made of three main subsystems, which are:

- The Shopping List Manager

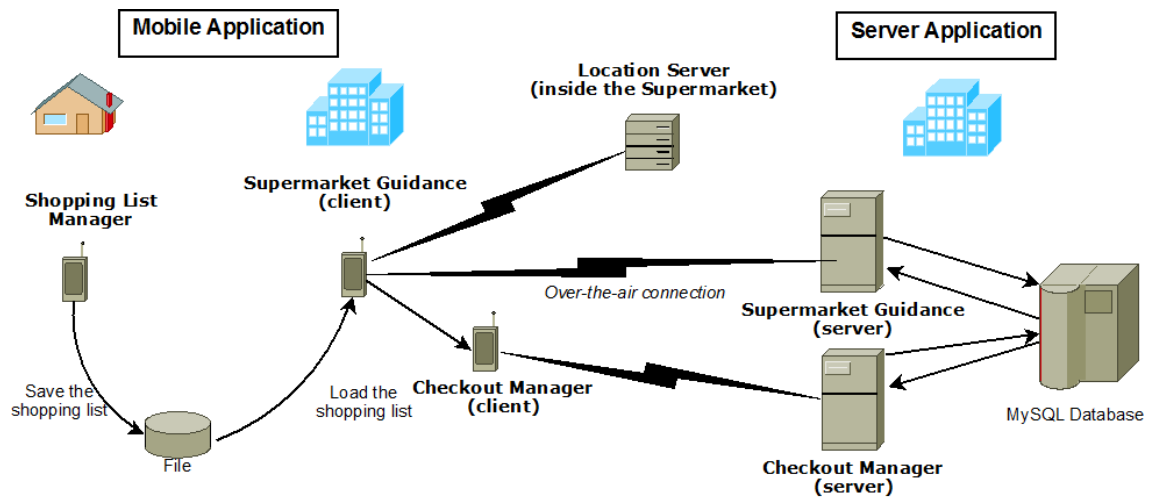


Figure 4.1: Global Architecture

- The Guidance in the supermarket
- The Checkout Manager

The shopping list manager is the part of the application that is run by the customer when he is still at home. It can run offline and it is used to build the shopping list. The shopping list is saved as a .txt file and it will be loaded by the second subsystem, the Supermarket Guide.

The Supermarket Guidance is run when the customers enters the supermarket, and it mainly consists in a GUI. This part is in close relation with the Supermarket server as it asks it for directions during the whole shopping process. The role of this Supermarket Guidance is to retrieve directions from the Supermarket Server and display them to the user in the best possible way; then to keep track on what the customer is doing and ask the server for the next directions regardingly.

There is a last part in the application on the mobile device: the one that deals with going to checkout. Indeed, when the customer is done shopping, he will select the "go to checkout" option in the Supermarket Guide. Then, the Checkout Manager will ask the server to join the queue for checking out. It will let the user know how many people are ahead of him in the queue, and when he is ready to check out, according to the data returned by the server.

4.1.2 The Supermarket System on the server

The Supermarket Server is an entity that runs on a server in the supermarket. It runs in Java, and does include a MySQL database.

It is made of two main subsystems, which are:

1. The Guidance in the supermarket
2. The Checkout Manager

The guidance subsystem is a big part of the project: it deals with the customers within the shop. It will retrieve their shopping list, will determine their path in the supermarket accordingly, then will

send them the different maps they need to see, and many more details that are discussed later in this document. Data are stored in a MySQL database.

The Checkout Manager handles the virtual queues to the checkouts. It keeps track on which checkouts are open or close and which ones are free to receive a new customer. It also keeps track of the customers waiting for a free checkout, and let them know how many people are ahead. Also, when it is their turn, it will let them know which checkout they have to go to.

4.2 Application on the mobile device

This application is developed in Python language in a object oriented way. It is divided into three parts: the shopping list, the guidance in the supermarket and the checkout part. The first part is independant of the others whereas the two last parts are depending on each other as at the end of the guidance, the checkout application will be launched. The Shopping list part is linked to the Guidance part as the Guidance will need to access the shopping list files created during the shopping list part.

About the interface of the mobile application, we started by creating a low-Fi prototype which was tested with Usability tests (cf. 6.1. Usability tests). This low-Fi prototype was rebuilt taking in consideration the results of these tests.

In this chapter, we will talk about the design of the application (from the reviewed low-Fi prototype) and about the functioning of the application.

4.2.1 Shopping List Manager

The Shopping List Manager is the part where the user can input all the products he wants to add in his shopping list. When he will go to the supermarket, the system will use one of the shopping lists he previously built with the Shopping List Manager.

A dynamic system

The system contains initially the list of products available in the supermarket and their associated categories. This list is stored in a .txt file. The system was made with the thought that when there is a new product or a new category available, it could be updated. It is also working when a product or a category is not available anymore. The only prerequisite is to modify this .txt file since the interface is built dynamically, based on the content of this .txt file.

How are the shopping lists saved?

The shopping lists are composed of two data types. Firstly the products and secondly the notes associated to each product.

We thought about two possibilities for saving the shopping lists: a database or a .txt file. We chose to use the second one because it was totally adapted to our needs and at the same time, it seemed to be really simpler to implement.

About the structure of this file, it should contain the names of products or identification numbers for each product with boolean values telling if the corresponding product is part of the shopping list (true) or not (false). Each product should also have a value corresponding to its category number and the notes the user input. The category number will only be helpful during the Guidance part.

Supermarket Guidance System – Shopping list manager

Produit	Notes
Potatoes	2kg

Home

Figure 4.2: Shopping list manager

The user can have several shopping lists, he can load the one he wants to access to on the main page of the application.

Building a shopping list

The first step of building a shopping list is to create a new list or to load a list, if there is already an existing one. A new list is a shopping list where every boolean is false and every note is null. If the user chooses to load a list, a file explorer will be displayed to allow him to choose the shopping list he wants to modify, this is handled by the class FileManager.

The main class of the program is MainPage. This class contains all the data of the shopping lists that will be modified by the different actions of the user. This is also the class that creates the Home Page screen. Every instance of class is created from this class.

As you can see in the image below (cf. Figure 4.1.), which is the main screen of the application, after having created or loaded a list, the list is displayed on the left part of the screen. The buttons allowing the user to interact with the application are on the right part of the screen. The user can directly modify the list by adding or deleting products or by modifying the notes. All the buttons present on this screen are connected to different classes and pressing one button creates a new object which will achieve the task needed.

The sequence diagram below (cf. Figure 4.2.) illustrates the first step of the application.

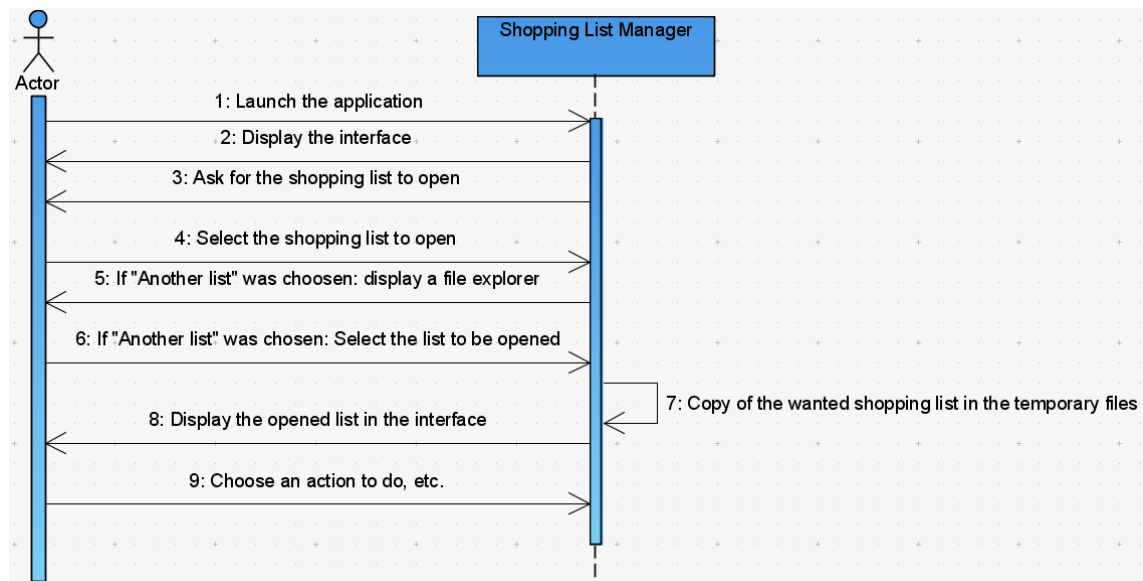


Figure 4.3: Sequence Diagram: Launch application and select the shopping list

For deleting a product or modifying notes, he needs previously to select the wanted product in the list and then to click on the button corresponding to the action he wants to realize. The class `RemovePdt` will be used for deleting product, the class `Notes` for editing the notes. For both objects, the name of the selected product will be given in parameters. If no product is selected, the user will be asked to select one. A confirmation will be needed for deleting a product with a popup window. To modify the notes, a popup window is displayed with the current note of the product, the user only has to input the new notes and to confirm that he wants to save it by pressing the button "Ok".

The five sequence diagrams below (cf. Figure 4.3., Figure 4.4., Figure 4.5., Figure 4.6. and Figure 4.7) illustrate the use of these options.

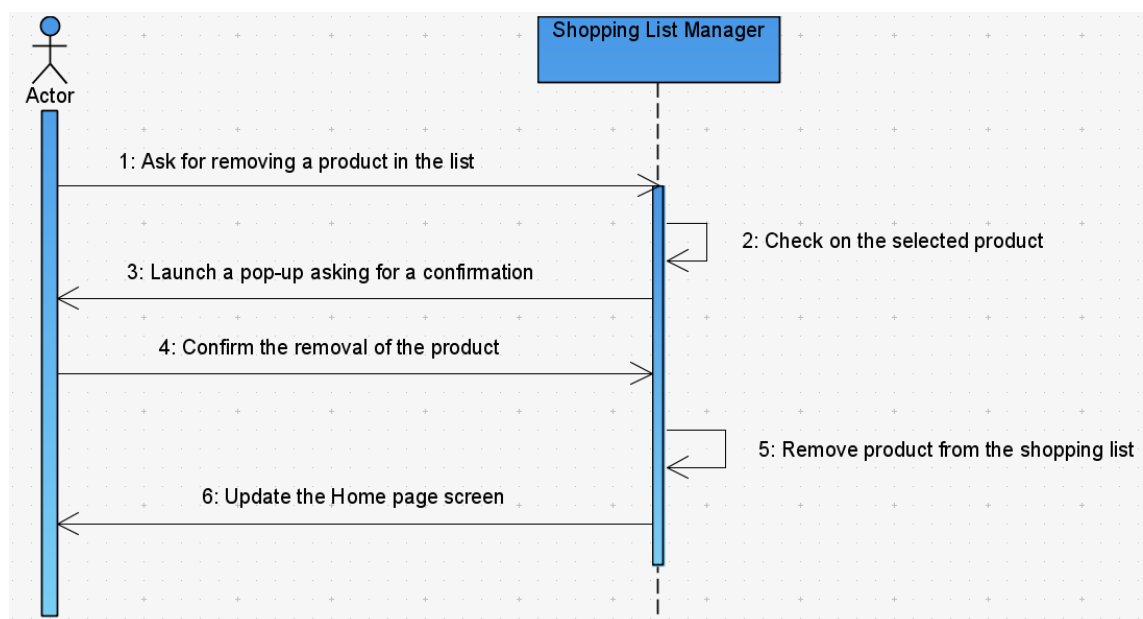


Figure 4.4: Sequence Diagram: Remove a product

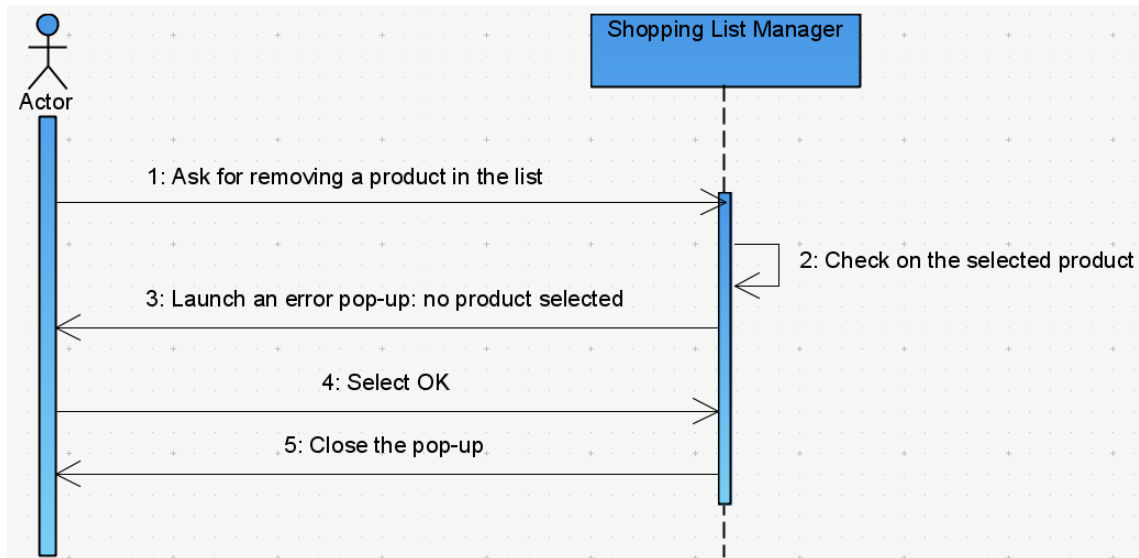


Figure 4.5: Sequence Diagram: Remove a product when no product has been selected

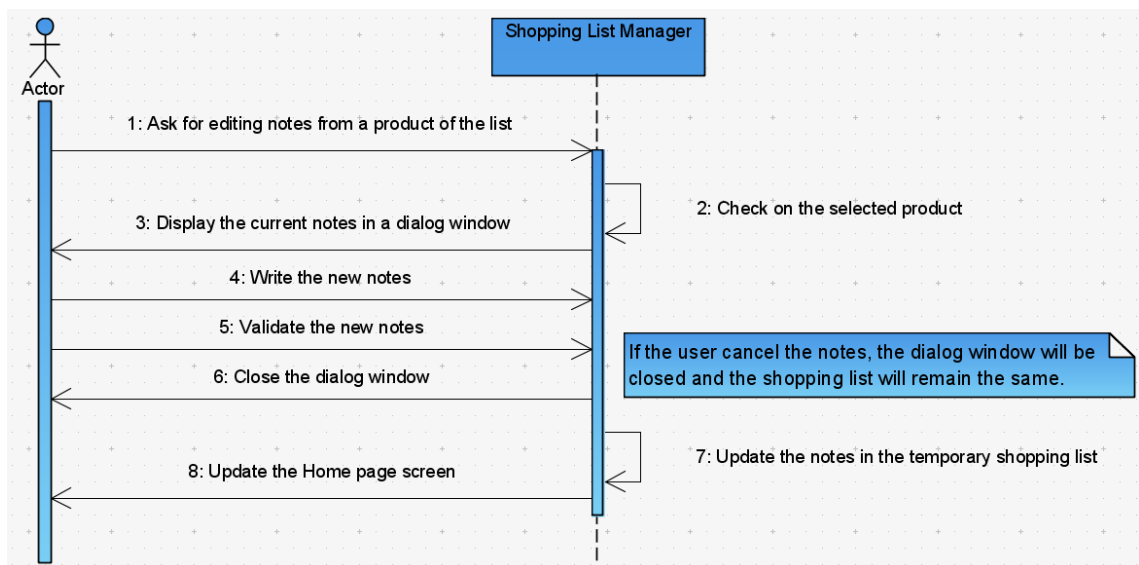


Figure 4.6: Sequence Diagram: Edit notes

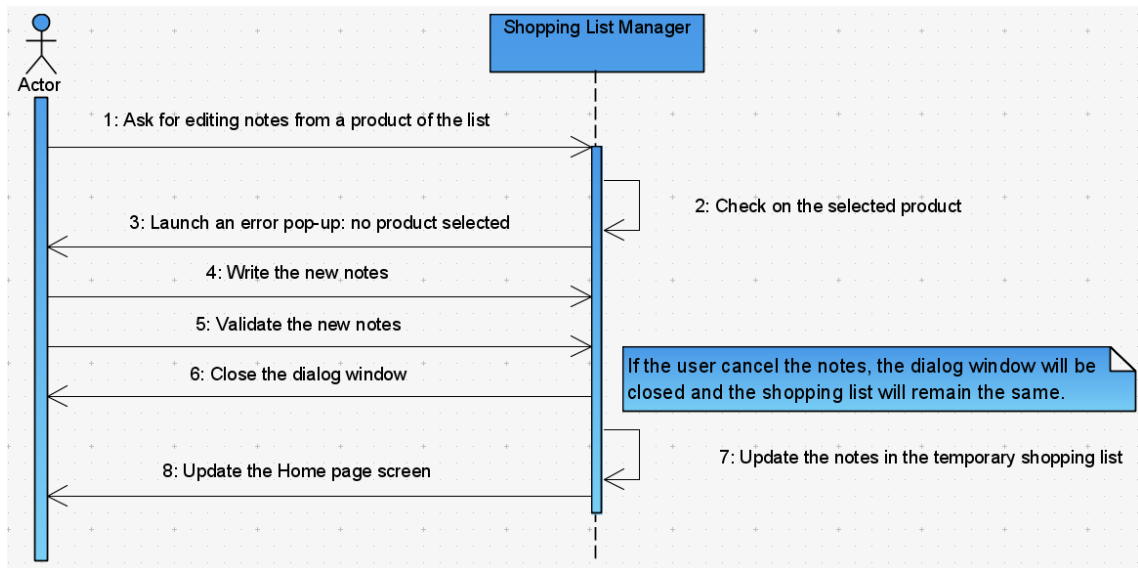


Figure 4.7: Sequence Diagram: Edit notes option when no product has been selected

For adding a product, a new screen will be displayed (cf. Figure 4.7.). The class `AddProduct` is involved, it receives in parameters the name of the shopping list to be able to display all the categories, products, notes and to be able to differentiate the products that are part of the shopping list from the others. The products are classified by category on the left of the screen, the notes associated to the selected product are on the right. The user has to select a category to access the list of products of this category via an expandable button. Then, he can select a product. After this selection, he can add the product to his list and write some notes. He also can remove it from the shopping list. All these modifications are saved in a temporary file as the user can still cancel all of them.

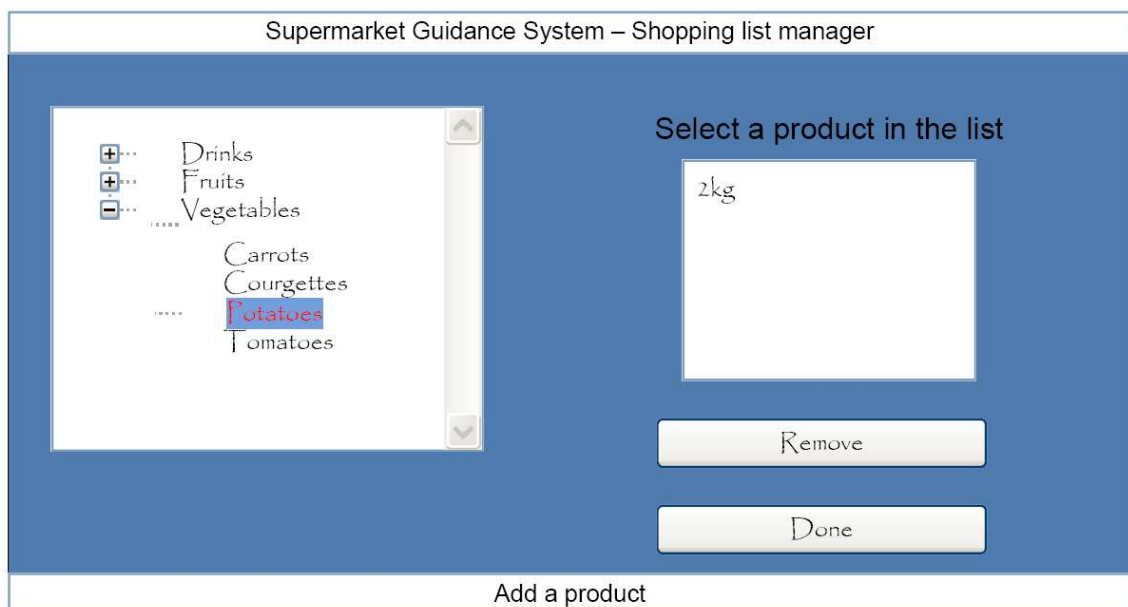


Figure 4.8: Add a product screen

When the user is done modifying the list, he goes back to the main screen, this screen is updated thanks to the class `UpdateWin`, the new list is now displayed on the left.

The sequence diagram below (cf. Figure 4.8.) illustrates that step.

Loading another shopping list

The user can whenever he wants load another list, create a new one or delete an existing shopping list. A file explorer is used in order to achieve that through the class FileManager.

The two sequence diagrams below (cf. Figure 4.9. and Figure 4.10.) illustrate the use of the Lists Manager.

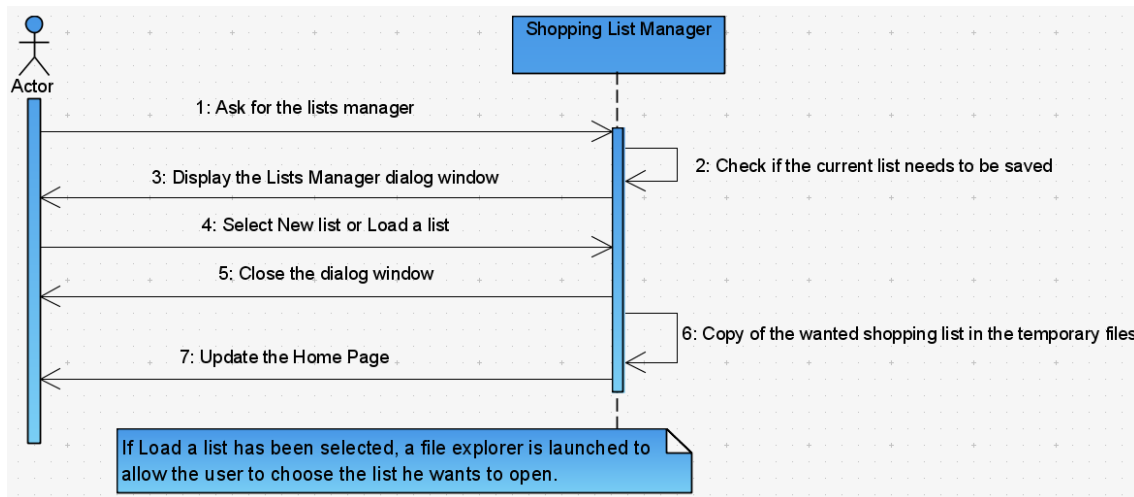


Figure 4.10: Sequence Diagram: Lists Manager

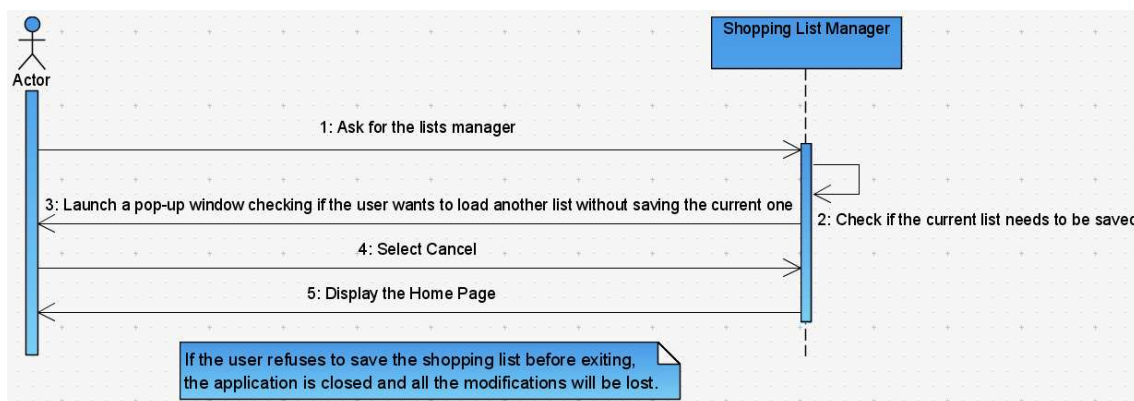


Figure 4.11: Sequence Diagram: Lists Manager when the current list has not been saved

Saving a shopping list

During the building of the shopping list, the modifications are saved in temporary files. It is only when the user saves the list that the modifications are saved in the real shopping list.

When the user is done with the modifications of his shopping list, he needs to save the list. He only has to select the Save list option. The class SavePopUp launches a file explorer and the user can choose the name of this list before saving it but not its location (all the shopping lists must be saved in the folder predefined by the application). The booleans and the notes corresponding to every product will be

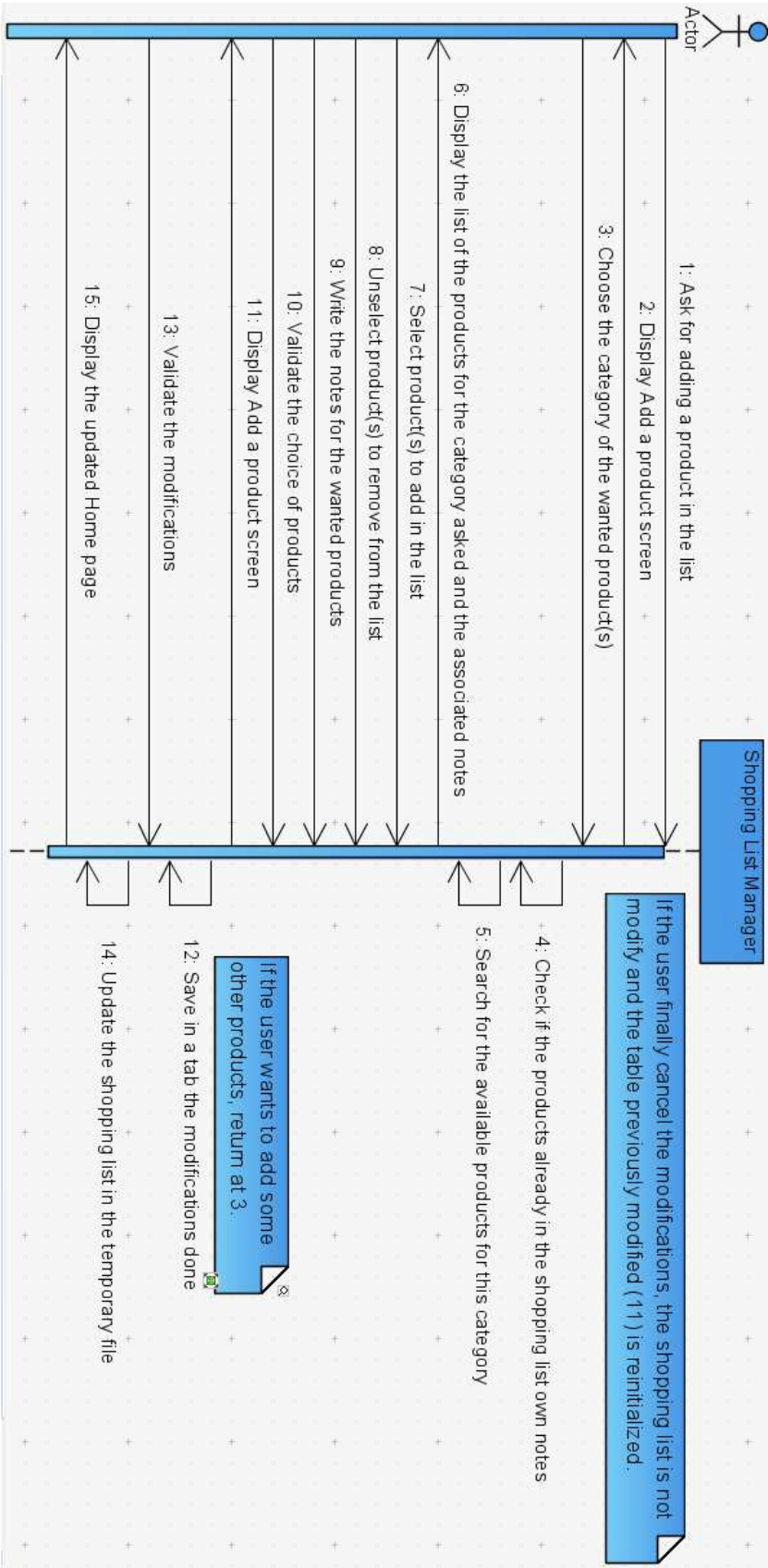


Figure 4.9: Sequence Diagram: Add a product in the shopping list

checked and modified if necessary.

The sequence diagram below (cf. Figure 4.11.) illustrates that step.

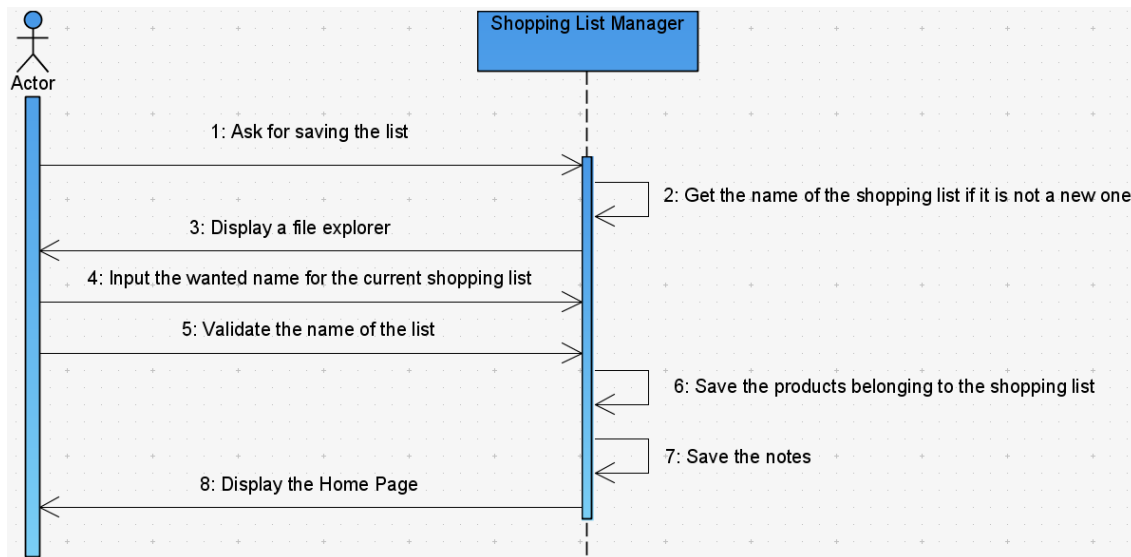


Figure 4.12: Sequence Diagram: Save the shopping list

Exiting the application

The user can exit the application whenever he wants. The class `CheckSaveExit` is used to check if some modifications of the list were done and not saved, in that case, the application suggests him via a popup window to save the shopping list before exiting, otherwise, the application is closed.

This step is illustrated by the two sequence diagrams below (cf. Figure 4.12. and Figure 4.13.).

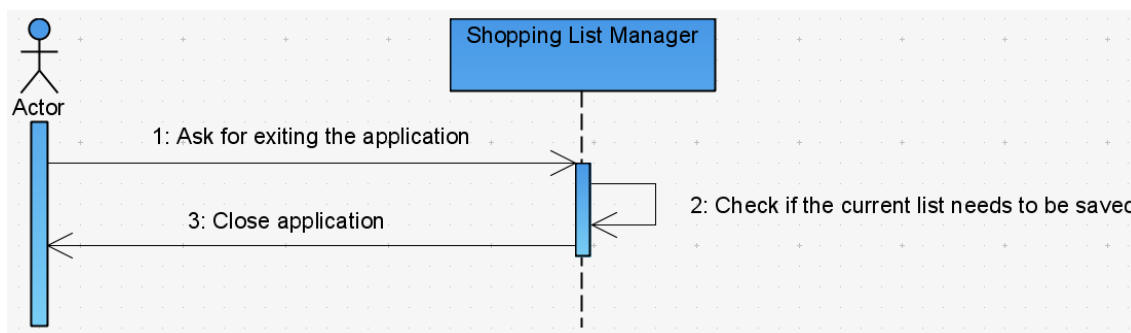


Figure 4.13: Sequence Diagram: Exit

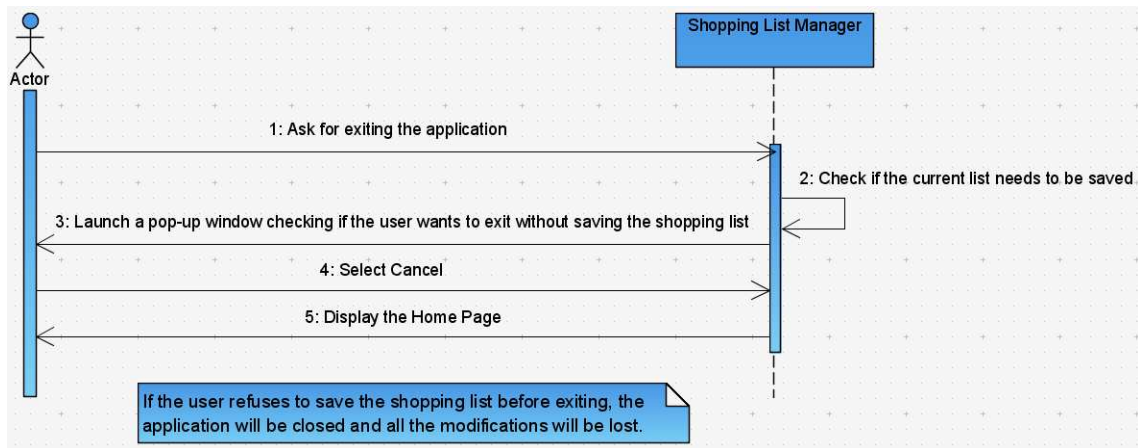


Figure 4.14: Sequence Diagram: Exit when the current list is not saved

4.2.2 Guidance in the supermarket

Connexion to the server

The first step of this part is to initialize the connexion between the mobile application and the server. We are planning to use xml-rpc in order to do that. The connection is initiated from the mobile application.

Then the mobile application needs to get two items: the name of the supermarket and the ID number corresponding to the connexion. The name of the supermarket is needed as our system can be implemented in many different supermarkets. The ID value allows to differentiate the different customers in the supermarket. In that way, the server will be allowed to send the information corresponding to the right customer.

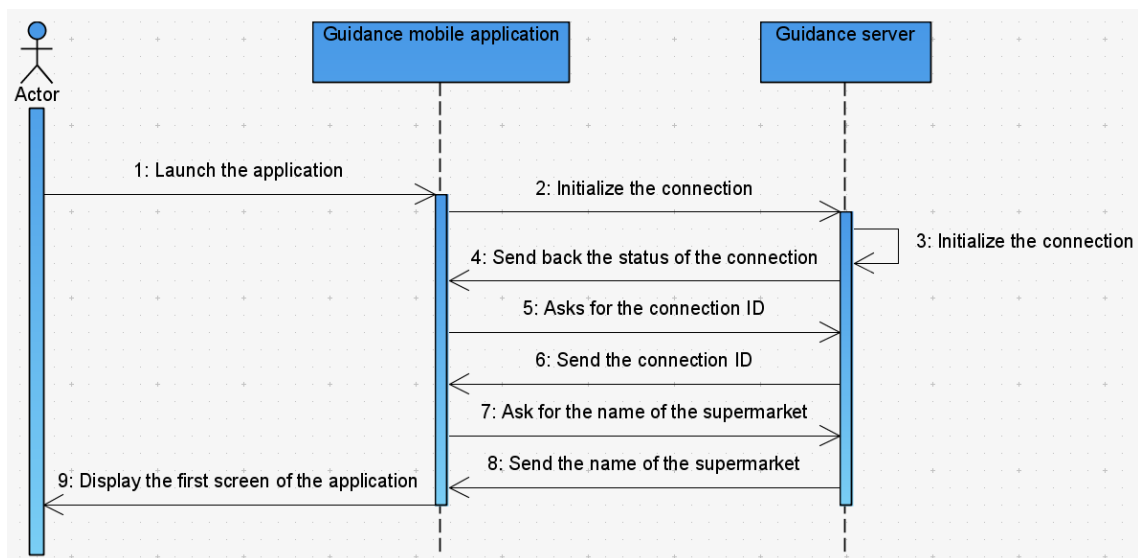


Figure 4.15: Sequence Diagram: Initialization of the connexion

Choice of the shopping list

After this connexion step is done, the main screen of the application is displayed. It asks the user to choose the shopping list he wants to use for shopping (as he can have several saved ones). Every available

list is displayed and the user has to choose one. The content of this list is sent to the server in order to compute the path the user will have to follow in the supermarket to pick up every product.

Go to a specific area

The guidance can now start.

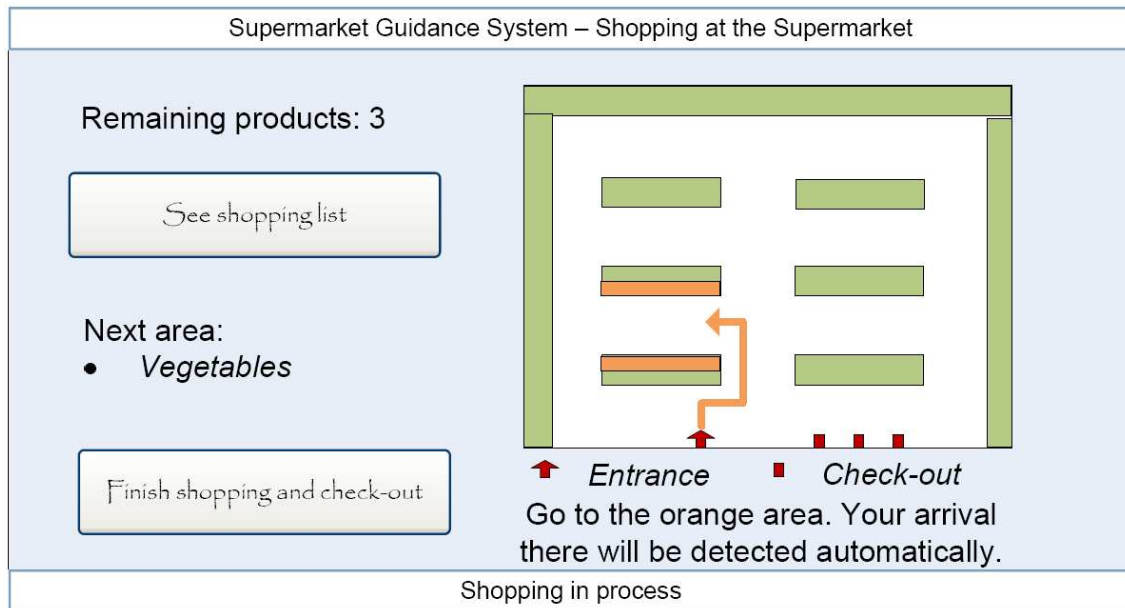


Figure 4.16: Next Area Screen

The mobile application gets from the shopping list the number of products that the customer is supposed to buy. Then it needs to get the name of the next area where the user should go, a request is done to the server to get it.

The important part of that moment is the display of the map to go to the next area. The server is asked to send it. To to this display, we had several possibilities: The first one is to generate an image (.jpg or others) in the server program regarding to the path the user has to follow and then to send it to the mobile application which will only have to display it. The second possibility is to use xpm files (X Pixmap files) which is a standard file format allowing to send image information (cf. A.4. Design of the map display via xpm data). And the third one we thought about is to send a table of integers in which each integer corresponds to the color to be displayed; when the mobile application receives it, it will draw a table on the screen which will be filled by the different colors described by the integers of the table.

As we do not have any knowledge about image creation, we were a bit anxious to generate a jpeg image. Besides, the map needed does not contain any text, we only need to display colors, so we decided that xpm data solution was the most appropriate, as it seems quite simple to implement but also xpm files are not too heavy to be sent to the user.

On this screen, the user can ask the complete shopping list to be displayed.

Supermarket Guidance System – Shopping list manager											
Shopping list											
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 5px;">Products</th> <th style="text-align: left; padding: 5px;">Notes</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">Carrots</td> <td style="padding: 5px;">1kg of fresh carrots</td> </tr> <tr> <td style="padding: 5px;">Coca-Cola</td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;">Potatoes</td> <td style="padding: 5px;">2kg</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> </tr> </tbody> </table>	Products	Notes	Carrots	1kg of fresh carrots	Coca-Cola		Potatoes	2kg			
Products	Notes										
Carrots	1kg of fresh carrots										
Coca-Cola											
Potatoes	2kg										
<div style="border: 1px solid black; padding: 10px; display: inline-block;">Close</div>											
Shopping list											

Figure 4.17: Shopping List Screen

Pick up the products

When the supermarket system detects that the customer arrived in the right area, the screen changes and the moment for the user to pick up the products has come.

Supermarket Guidance System – Shopping at the Supermarket											
<p>You are in the following area: Vegetables</p> <p>Products to pick up</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 5px;">Products</th> <th style="text-align: left; padding: 5px;">Notes</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">Potatoes</td> <td style="padding: 5px;">2kg</td> </tr> <tr> <td style="padding: 5px;">Carrots</td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> </tr> </tbody> </table> <p>When every product has been picked up, you can continue shopping</p> <div style="text-align: right; margin-top: 10px;"> <div style="border: 1px solid black; padding: 10px; display: inline-block;">Continue shopping</div> </div>		Products	Notes	Potatoes	2kg	Carrots					
Products	Notes										
Potatoes	2kg										
Carrots											
Picking up products											

Figure 4.18: Vegetables Area

On the screen are displayed the list of products to pick up corresponding to the area where the customer stands. They are associated with the notes for each product. This list is not requested to the server, but it is found directly in the shopping list.

For this screen, the mobile application asks the server to send the new map, the one corresponding to the area where the user arrived. On this map is specified the location of the products present in the shopping list. The map is displayed the same way than for the previous one.

When the customer has picked up every needed product, he asks for continuing shopping. If there is still some products left, the screen will display the map to the next area, otherwise, the application suggests the user to go to check-out.

Note: The user can stop shopping and go to check-out whenever he wants during his shopping.

4.2.3 Checkout

When the user asks for going to check-out, this information is sent to the server which sends back a waiting number. Then, the mobile application is regularly asking for getting the updated waiting number. At the same time, the user has to wait for this number to become zero, his number will decrease of one everytime a check-out assistant will have finished with a customer.

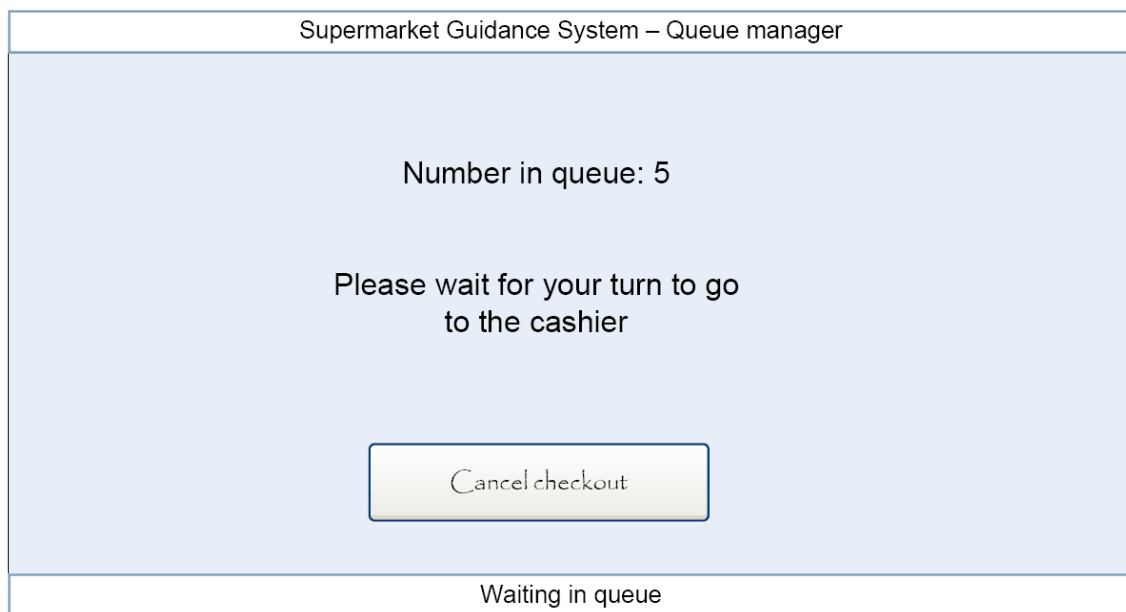


Figure 4.20: Check-out - Waiting ticket 5

When the waiting number is zero, the screen is updated and it asks the customer to go to the cashier whose number is specified. He can then exit the application.

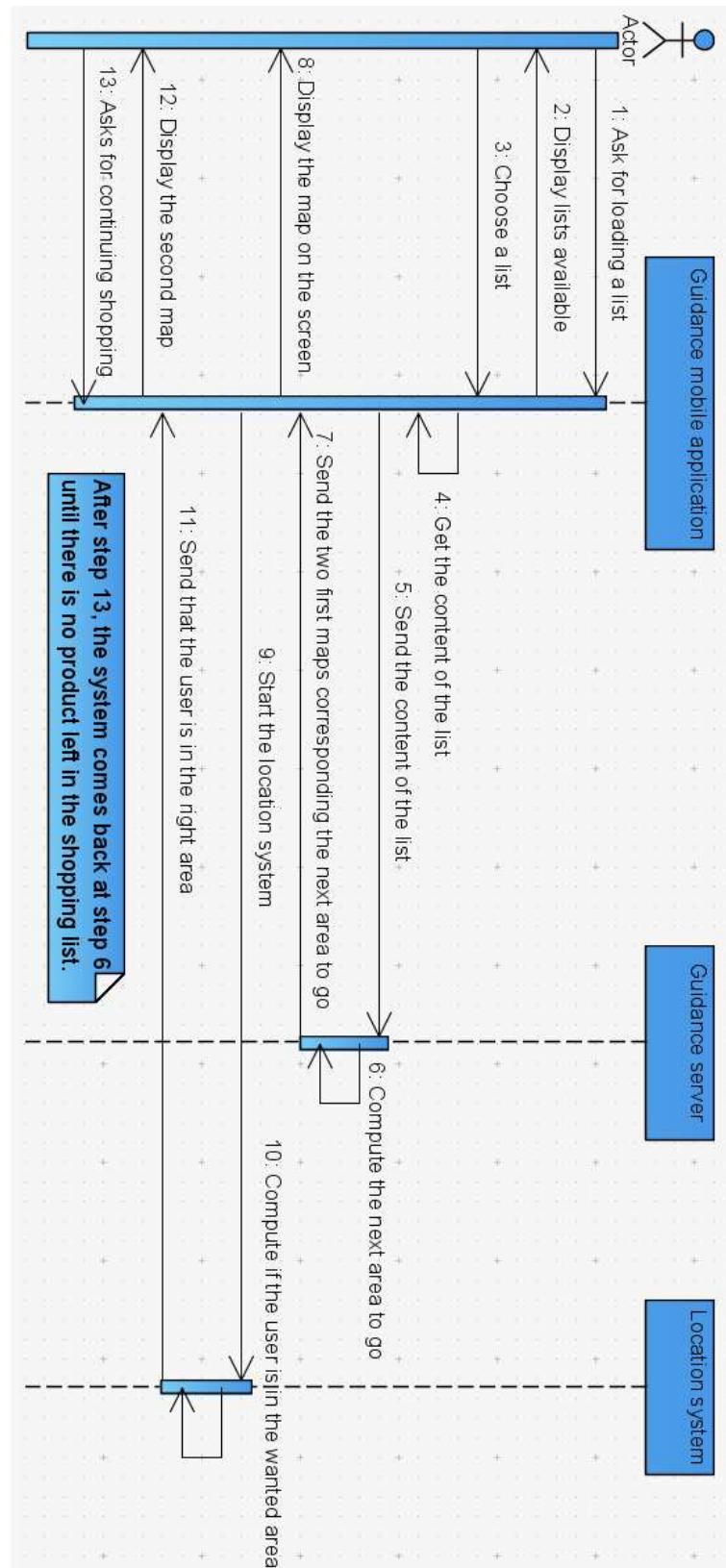


Figure 4.19: Sequence Diagram: Guidance in the supermarket

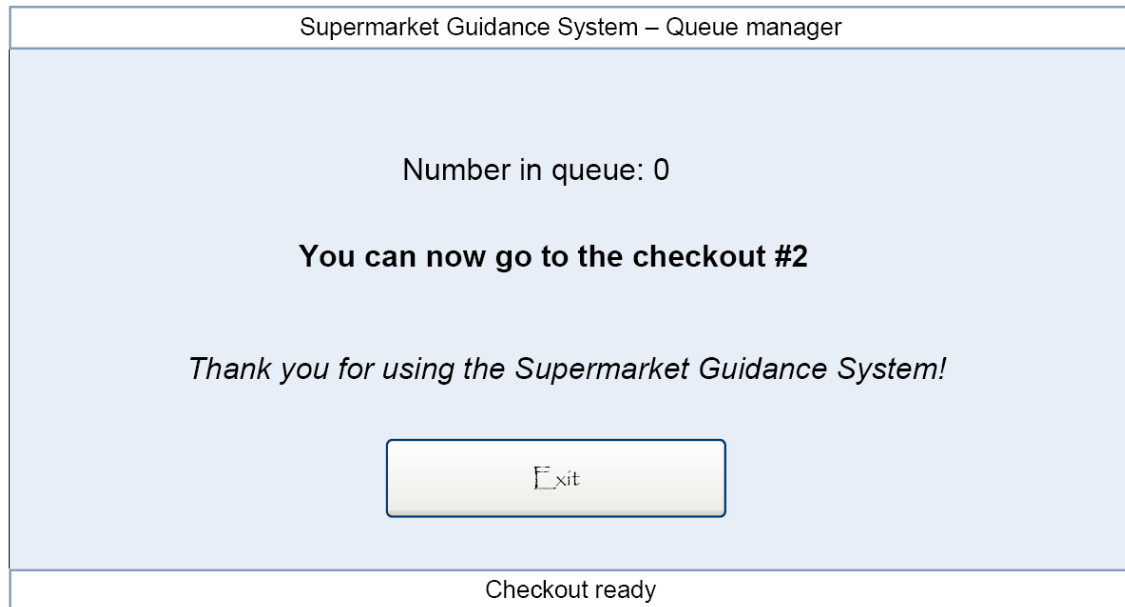


Figure 4.21: Check-out - Waiting ticket 0

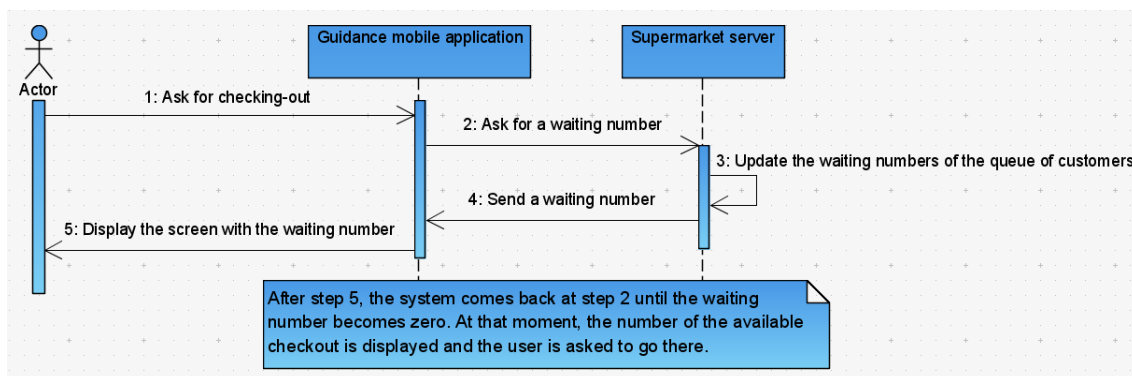


Figure 4.22: Sequence Diagram: User and checkout

4.3 Supermarket System - the server

4.3.1 Guidance in the supermarket

The supermarket owns a server which is running the Guidance application. The purpose of this application is to keep track on every client connected and to provide them directions through the shop. A client is a mobile device of a customer inside the shop and willing to use the Guidance System.

According to the concept of xml-rpc connections, the client can access methods from the server punctually, but the server cannot really keep an active connection with the client as it is the case with a socket connection for example. Also, the server **cannot** send anything to the client on its own. It is **always** the client that requests something from the server (by calling a method), and then the server can reply. This must be kept in mind while designing the server.

So the server offers a set of methods and the whole shopping process is about calling the right methods at the right time and with the proper parameters. The server is designed and the client has to adapt to

the way the server works.

The whole shopping process (excluding checking-out, which is discussed in the next section) includes these following steps:

- Initializing the connection with the client (many clients can be connected at the same time)
- Retrieving the whole shopping list
- Providing the directions (map) to the closest area the customer must be sent to
- Providing the instructions to pick up the products inside the area (zoomed map)
- Providing the next directions to the next area and so on, until every product was picked up.

We will go through each step one by one, considering what the server has to do for each one.

Initializing the connection with the client

As mentioned earlier, there is no such a thing as one connection that would be opened at the start and closed at the end, like a socket connection would work. In fact, a connection only lasts the time of the execution of one method of the server.

So when a client executes a method, we cannot know which methods this client executed before, which data he sent, etc. In order to know that, it has been decided to give a unique ID to the client, and the client would submit this ID as a parameter every time he executes a method: this way, we actually know who he is and allows the server to react accordingly.

So this is what initializing the connection is about: providing a unique ID to the client, which the client will always send in the future, so the server knows who he is.

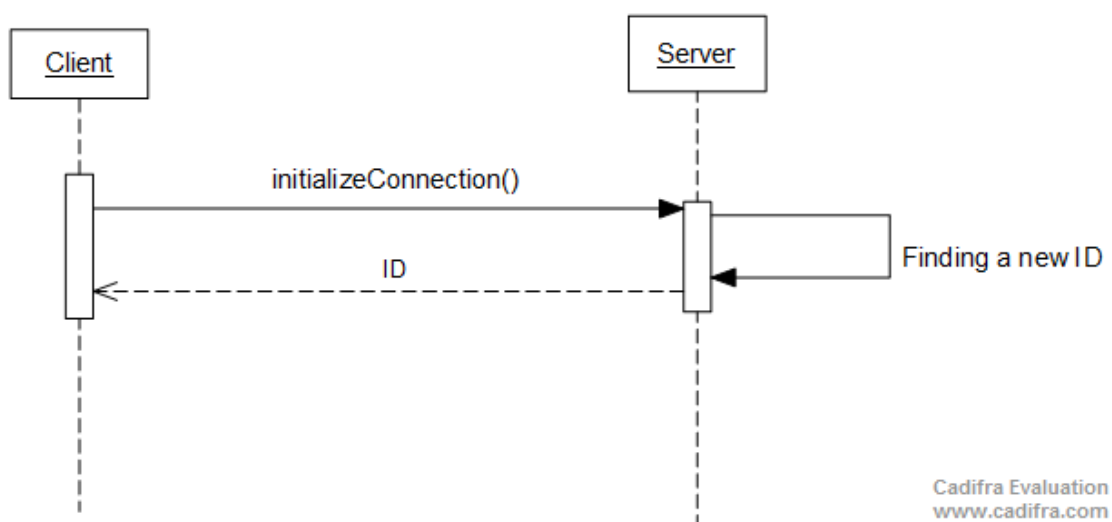


Figure 4.23: Sequence diagram: Initialization of the connection between client and server

Retrieving the whole shopping list

The shopping list of the client must be retrieved by the server, so the server knows where the customers has to go to and can determine a path.

Both the client and the server have a reference table of the products, where each product is associated with an ID. So in order to build the shopping list, the client will call a method to add a product to the list, and will provide the ID of this product. This method will be called as many times as there are products in the customer's shopping list.

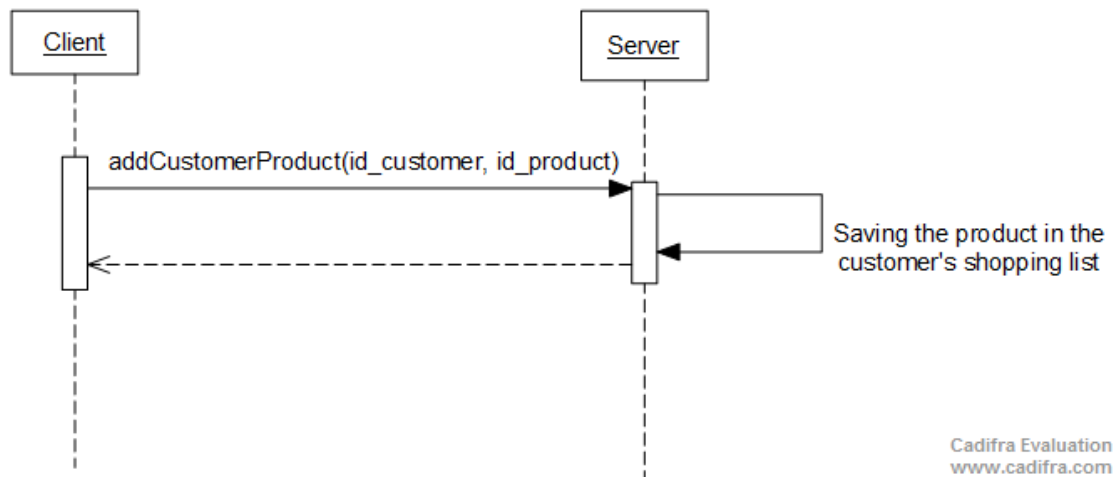


Figure 4.24: Sequence diagram: Retrieving of the products of the shopping list

The server does not know the size of the customer shopping list. When the client calls the next methods, it will assume that the shopping list is complete.

Providing directions to an area

This is a big part of the server's work. It will result in providing the client a map of the shop that includes the directions.

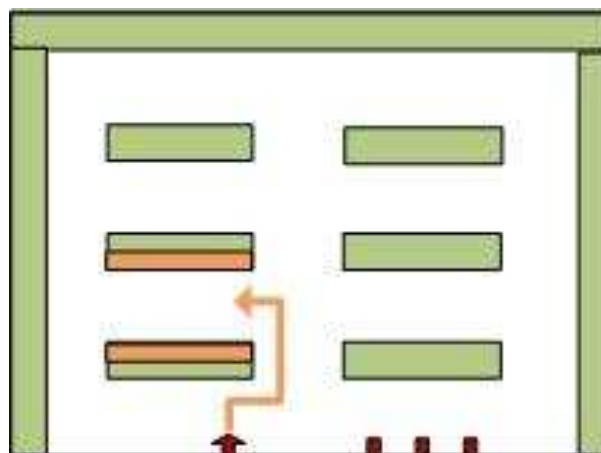


Figure 4.25: map with the directions to an area

In order to provide these directions, the server must go through several steps:

- Determine which areas the customer has to go to
- Determine which one among those areas is the closest
- Create the map
- Update the customer's data

Determine which areas the customer has to go to

A product is associated to a category (for example, "Wine" is associated to "Drinks"). The supermarket is divided in several areas, one area containing all the products of one category. When the client requests for directions, the server will establish the list of the areas the customer has to go to, considering that when a product is in the list, he has to go to the corresponding area.

Determine which one among those areas is the closest

Providing directions that will allow the user to walk as few as possible is one of the objectives of the system. Hence the process that will look for the closest area the customer has to go to, so he is not sent to a random area at the other side of the shop when he should pick up a product just a few meters away from his current position.

In order to perform this research of the closest area, a Dijkstra-like algorithm is used. It is a simplified version of the regular Dijkstra algorithm and this is how it works. The map of the supermarket is made of 900 pixels, some of them are walkable and some are not (you cannot walk through the shelves for example). Starting from the current position of the customer (which is initially the entrance of the shop), the algorithm will search for the next possible pixels, and will give them a cost of one, as it takes travelling through one pixel to reach it. This process is then repeated from every pixel which have a cost of one, to see which pixel can be reached with a cost of two; and so on.

The algorithm ends when a possible destination is found, possible destinations being "spots", a special type of walkable pixel that represent the places where the user can stand. For example, the Breakfast area will be given two spots, one at each side of the shelves. When the algorithm reaches one of those two spots, it will stop looking for a path as it found the closest area.

Note: Sending the customer to the closest area he must visit may not provide the overall shortest path: a better algorithm would consider the whole path at once, considering different possibilities of going through all required areas with a given starting point which is the entrance and a given ending point which is the checkout area. This is a Traveling Salesman Problem¹ which is actually an extremely tricky problem to solve. Writing shortest path algorithm is not the main goal of our project, so we decided to use a simpler algorithm.

Create the map

Once the shortest area has been found, the server will create the map that will include the directions. Basically, it consists in the regular map of the supermarket, with enhancements:

- The area the customer has to go to is enlightened in Orange
- The path he must follow is also enlightened in Orange

¹http://en.wikipedia.org/wiki/Traveling_salesman_problem

In practice, the map is a two-dimensional array of Integer, that represent the whole shop pixel by pixel. A pixel can have several values, as it can be an entrance, a checkout, a clear floor, an enlightened floor (if it is part of the path), a normal shelf, an enlightened shelf.

This array is sent back to the client, that will display it to the user as an image.

Update the customer's data

Finally, the server has to update the data about the customer. Indeed, as it just send the customer to an area corresponding to a specific category, Breakfast for example, it must mark the products as taken, as the customer will take them.

It can seem a bit early to do that as the customer has not picked the products yet. On the other hand, it can be done now and it is more clean to do it as soon as the server knows which products are going to be picked up.

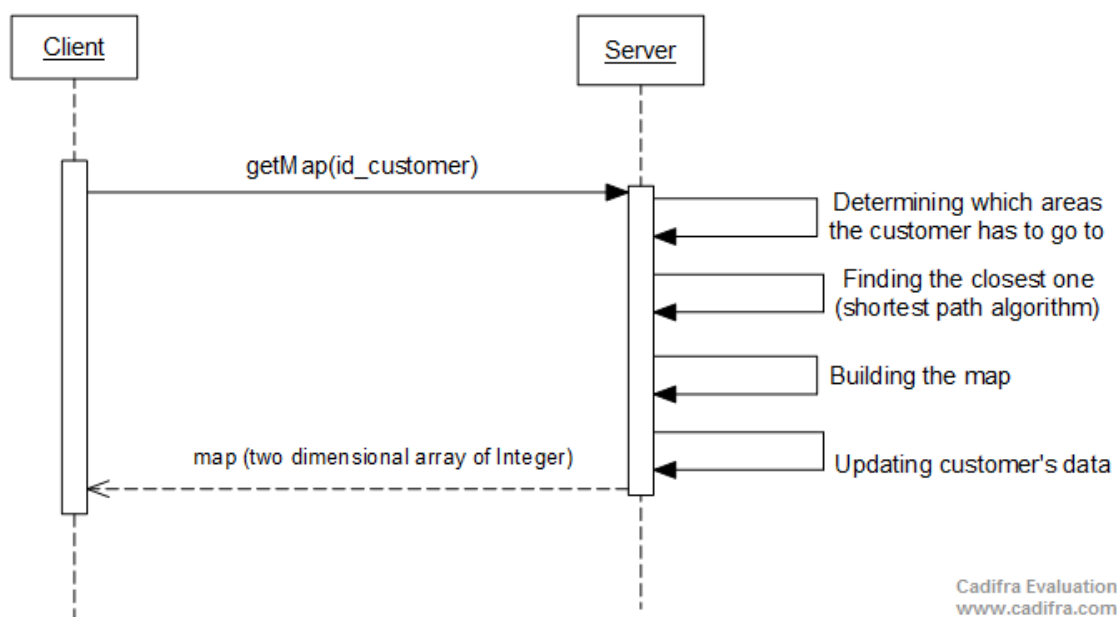


Figure 4.26: Sequence diagram: getting the map

Providing instructions inside an area

When the user arrives at the area he has been instructed to go to, his arrival will automatically be detected thanks to the location system of the university that we will use. Then, the client will ask the server for a detailed map of the area which includes the different products to be picked up in the area with their precise location.

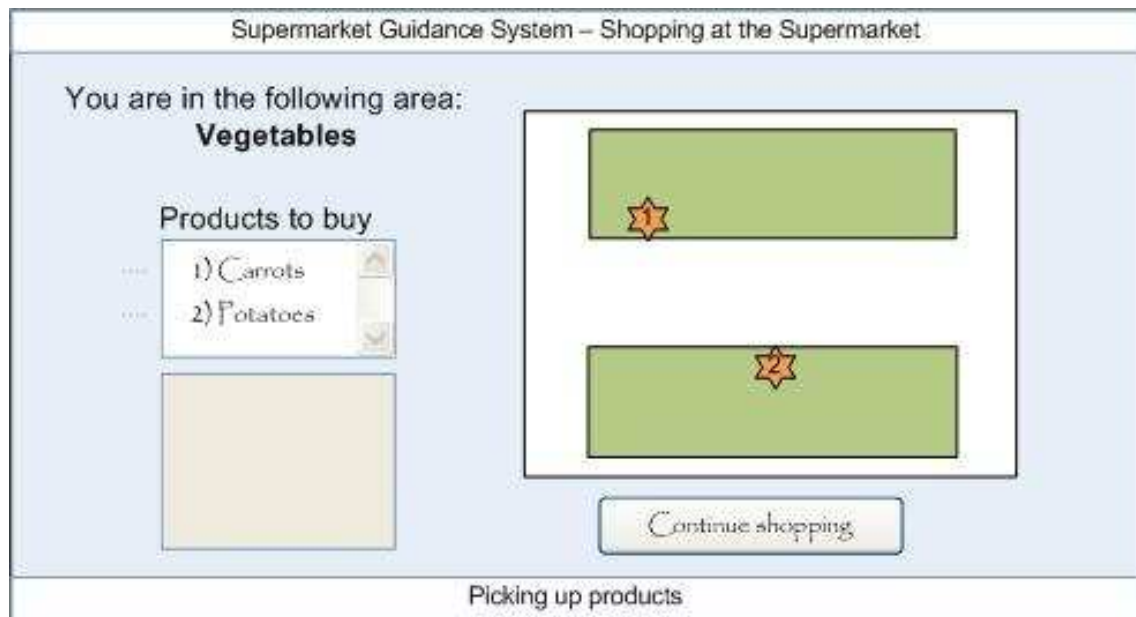


Figure 4.27: Zoomed map

The map creation process of the detailed map is similar to the creation process of the general map, except, of course, it is much smaller as it is zoomed in. Each pixel is given a value that represents either a clear floor, a shelf, or a product inside a shelf. Different values are given to different products so it is possible to differentiate them.

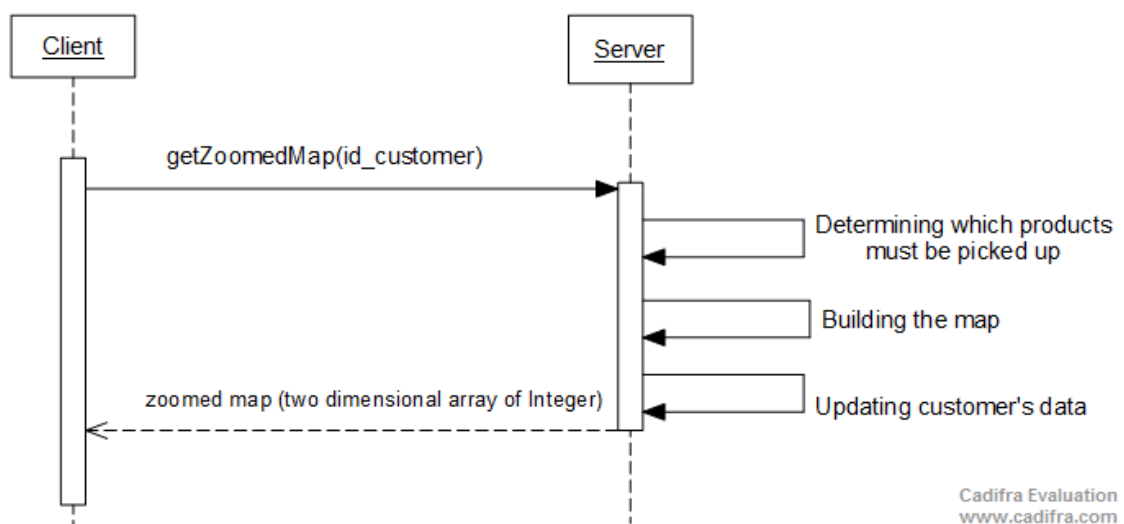


Figure 4.28: Sequence diagram: getting the zoomed map

As long as products remain to be picked up in other areas, the client will keep calling the method `getMap` and then the method `getZoomedMap`. When shopping is completed, it will call another function in order to check out.

Data storage in the database

Data must be saved in the server. A MySQL database is used for this purpose; it runs on the server.

We can differentiate two main types of data: the static data and the dynamic data. The static data is the one that is almost never supposed to change. It includes the map of the supermarket (which changes only when the supermarket is reorganized) and standardized reference table of the products and category, that associate each product to a category and an ID.

The dynamic data often changes; it consists in all the data that are relevant to keep track of the customers. It does include the customers' IDs, their current positions, their shopping list with the products that have already been taken and the ones that have not. It also includes data useful for the calculation of the path to the shortest spots: the list of the customers' spots, all his possible path in the supermarket, etc. Let's have a closer look to the tables of the database.

Static data

The map of the supermarket is saved in a table. Each entry of the table represents a pixel (identified by two coordinates).

Here is a simple view of the map of an imaginary supermarket:

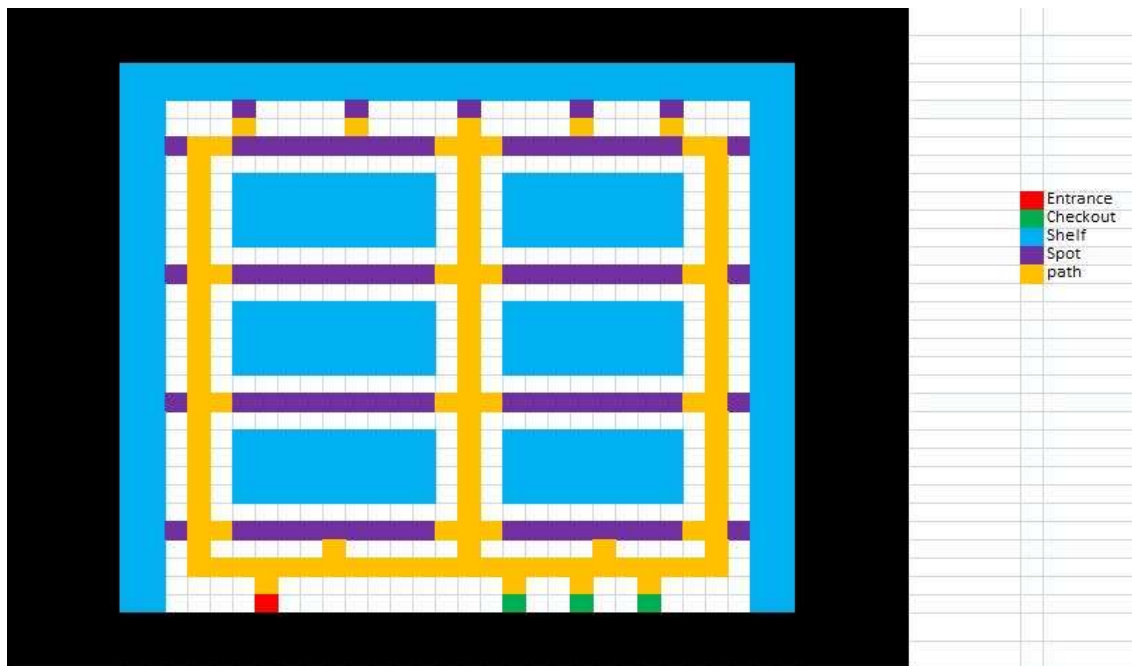


Figure 4.29: Map of an imaginary supermarket

The entrance is in red and the checkout are in green. Shelves are in blue.

The purple pixels are spots corresponding to the places where the customer can stand and is considered inside a specific area corresponding to a category. The orange pixels are walkable parts of the floor and the white pixels are non walkable parts of the floor.

Why cannot the customer walk anywhere inside the shop? Why would he have to stay on this imaginary orange path?

The customer can obviously walk on any part of the floor inside the shop. This concept of walking

floor is very useful during the execution of the shortest path algorithm to the closest spot. Indeed, the algorithm will only compute the orange pixels, not the white ones, which saves a lot of computation time.

Each pixel of this map is saved in an entry of the table of the database named "map". As the map size is 30*30, there are 900 entries in that table.

One entry includes four integer values: x, y, type, detail. x and y represent the coordinates of the pixel. type represent the type of the pixel, and can be:

- 0 (a floor)
- 1 (entrance)
- 2 (checkout)
- 3 (path)
- 4 (spot)*
- from 11 up to 98 maximum (37 in reality) (a product on a shelf, with an id of type - 10)
- 99 (a shelf)*

The detail field only matters for the entries with a type marked with a star *. For a spot, the detail has the value of the ID of the category the spots belongs to. For a shelf, detail has the value of the category of product it belongs to.

Product references and category references are also stored in the database, in a very classical way.





























←T→			id	id_category	label
<input type="checkbox"/>			1	1	Water
<input type="checkbox"/>			2	1	Wine
<input type="checkbox"/>			3	1	Soda
<input type="checkbox"/>			4	2	Apples
<input type="checkbox"/>			5	2	Bananas
<input type="checkbox"/>			6	2	Grapes
<input type="checkbox"/>			7	2	Pears
<input type="checkbox"/>			8	2	Oranges
<input type="checkbox"/>			9	3	Potatoes
<input type="checkbox"/>			10	3	Carrots
<input type="checkbox"/>			11	3	Courgettes
<input type="checkbox"/>			12	3	Concombre
<input type="checkbox"/>			13	4	Pork
<input type="checkbox"/>			14	4	Beef

Figure 4.30: References of products in the database









←T→			id	label
<input type="checkbox"/>			1	Drinks
<input type="checkbox"/>			2	Fruits
<input type="checkbox"/>			3	Vegetables
<input type="checkbox"/>			4	Meat

Figure 4.31: References of category in the database

The table on the left lists every product. Each product has a unique id, and is associated with a category.

The table on the right is the category list, with their specific id.

Dynamic data

The information about the customers currently inside the shop are dynamic data. Customers' shopping lists are one of those. It is saved in the customers_shopping_list table, as displayed on the next figure.





















←T→			id	id_product	taken
<input type="checkbox"/>			1	3	1
<input type="checkbox"/>			1	5	1
<input type="checkbox"/>			1	9	1
<input type="checkbox"/>			1	11	1
<input type="checkbox"/>			2	3	0
<input type="checkbox"/>			2	5	1
<input type="checkbox"/>			2	9	1
<input type="checkbox"/>			2	11	1
<input type="checkbox"/>			3	20	1
<input type="checkbox"/>			4	1	1

Figure 4.32: Shopping list in the database

The id is the unique id of the customer. The next field is the id of the product, the name of the product can be obtained by checking the reference table on the previous figure. The last field is a boolean which keeps track on whether the product has already been picked up or not.

This is the most interesting dynamic data stored on the server. There is much more, all included in the appendix. Some of it was added, or modified, during the implementation part of the project; this is discussed in the implementation section.

4.3.2 Checkout

Besides guiding customers through the supermarket, another task of the server is to handle the virtual waiting lines at the checkouts. Basically, when a client wants to check out, the server is notified and includes the client at the end of the virtual waiting queue, which is made of every client who need to check out.

When a checkout is free, the server gives it the next customer. Then it has to notify the client that he can check out and at which check out he should go.

Three methods are available to the client: one that request to enter the queue, one that requests for the place in line and one that requests for the number (id) of the checkout to go to.

The client first requests to join the queue. Then, it requests its place in line in order to display it to the customer, so he has an idea of how long it's going to take before checking out. When the place in line is zero, the client calls the last method, that will return the id of the checkout the customer has to go to.

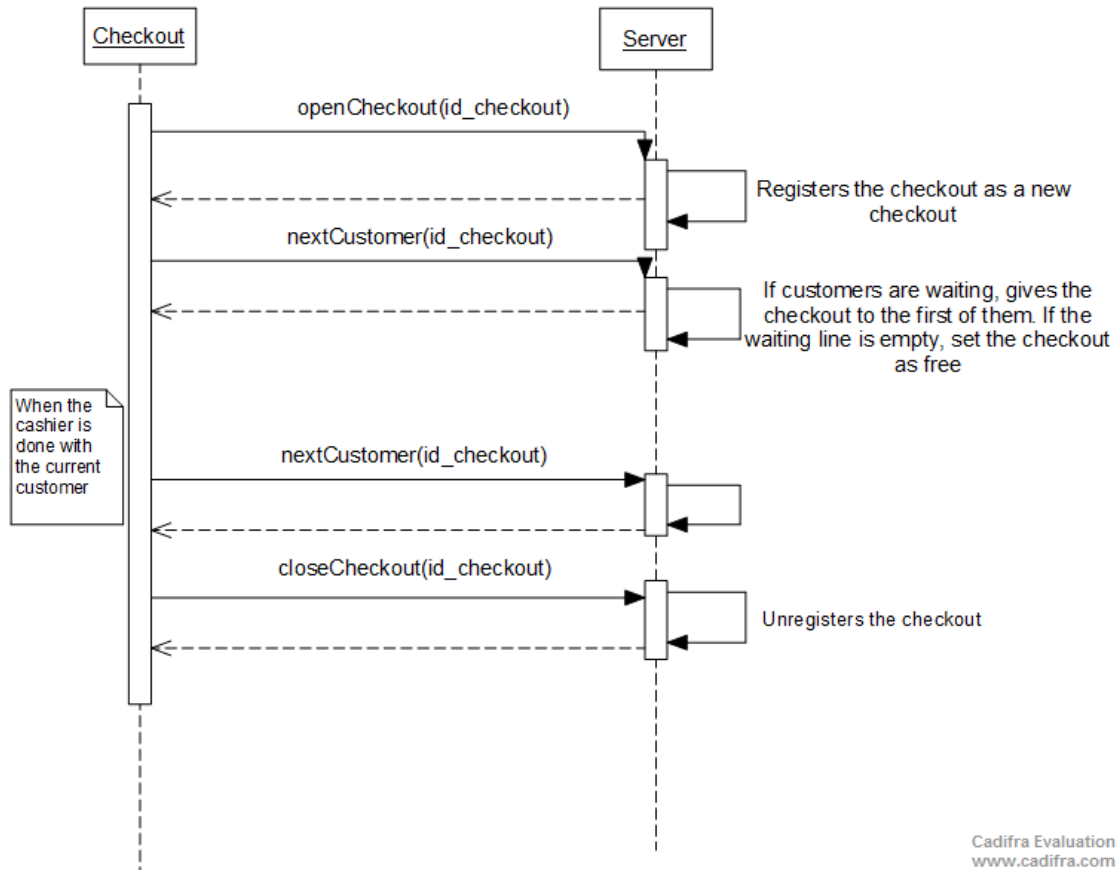


Figure 4.33: Sequence diagram: check-out

In order to keep track on which checkout are available, three other methods are also available to the checkout.

At the initial stage, there are no checkout registered in the system, as at the beginning of the day, every checkout is closed. When a checkout opens, it calls the `openCheckout` method and provide its number (usually, in a supermarket, each checkout is given a number that never changes). Then, it notifies the server that it is free and up to receive a customer. The server will either send a customer to the checkout immediately or, if nobody is currently waiting to check out, will set the checkout as free and will send it the next customer that enters the virtual waiting line.

When the cashier is done with a customer, it will notify the server again (by calling the same method) in order to ask for the next customer. At the end of the cashier's shift, the checkout will call a method to unregister from the system.

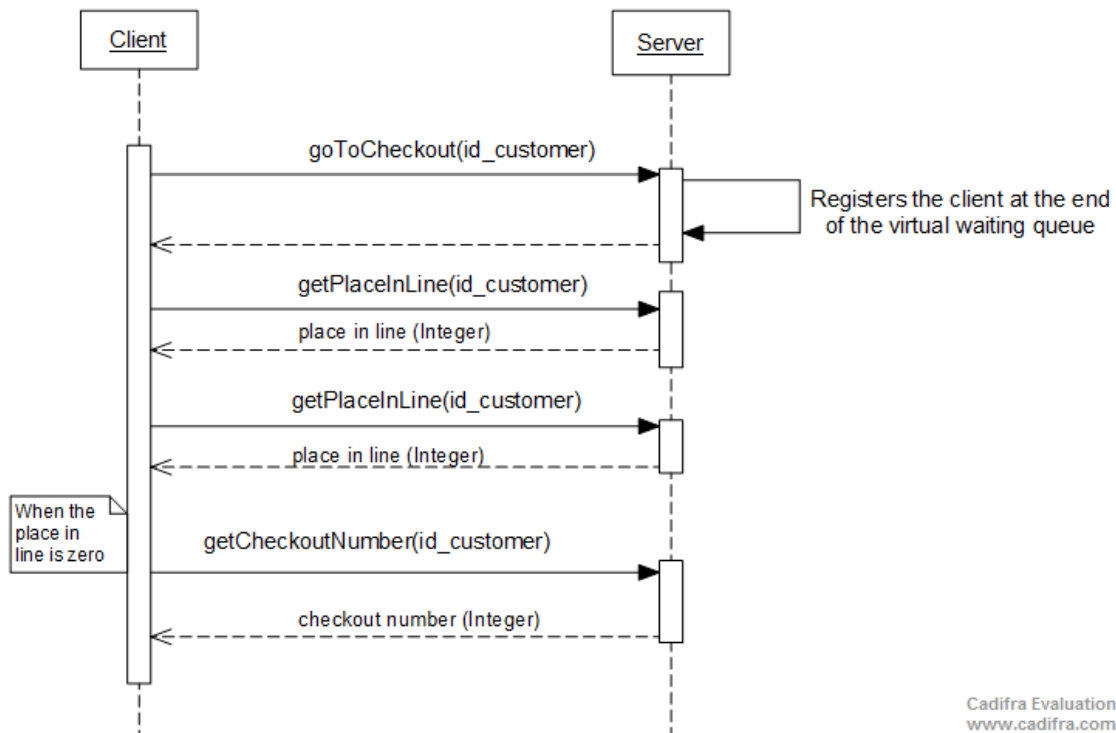


Figure 4.34: Sequence diagram: check-out and client

Data storage in the database

As for the guidance part of the server, the checkout data are stored in the database. Two tables are enough in order to save the data: `customers_queue` and `checkouts`.

The table `customers_queue` saves the virtual waiting queue, with the id of the customers waiting, and their place in line.

The table `checkouts` keeps track on which checkouts are open, and if they are free.

←T→	place ▲	id_customer	←T→	id free
<input type="checkbox"/>			<input type="checkbox"/>	
	1	12		1 0
<input type="checkbox"/>			<input type="checkbox"/>	
	2	9		2 0
<input type="checkbox"/>				
	3	11		

Figure 4.35: Check-out in the database

4.4 Supermarket System - the check-out application

The check-out assistant has a defined role in our system. He is the one asking for the opening and closing of a check-out and he also needs to interact with the system everytime he has finished with a customer. This last action is responsible for the updating of every waiting numbers of the customers and it allows

the next customers to know that it is his turn to go to the cashier.

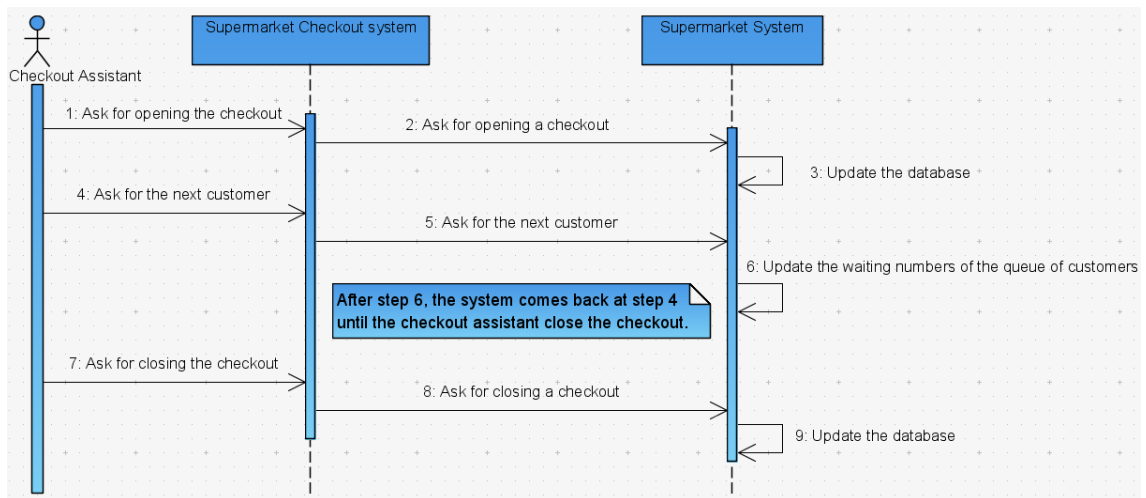


Figure 4.36: Sequence Diagram: Checkout Assistant

We will build a special interface for the cashier to allow him to interact with the system.

5

Implementation

The implementation chapter describes how we finally implemented the system. The aim is to show the differences between the system we actually realized and the one we described in the Design part. The problems we encountered will be explained as well as the different solutions we thought about to resolve them. It is divided into three parts, the choice of development language, the mobile application and the supermarket system.

5.1 Choice of development language

The project includes development on two different platforms: the Nokia tablet, and a server. Hence we should choose wisely the programming language that suits the best for each platform. It is totally possible to use different languages.

5.1.1 Mobile programming

The Nokia N800 is based on Maemo platform¹ which allows the developer to use the following languages: C, Python and Java.

Let's consider the pros and the cons of each language.

C language

C language is a popular, widely used language. It offers great liberty to the programmer, allows to do almost everything and is very fast to execute. The main drawback is that it is not the most programmer-friendly language and using C usually requires more time and energy than other languages do. The C programmer has to take care about some details that another programmer does not need to consider. This especially happens when dealing with Strings or when a garbage collector is necessary, for example.

C language is the main language for developing on Nokia Internet tablets. Indeed, it is the official language of Maemo platform.

Pros:

- Official Maemo language

¹<http://maemo.org/>, may 2008

- Everything can certainly be done
- Graphics easy to do with GTK libraries
- The project team is skilled in programming in C language.

Cons:

- Complex, time-consuming language for programming

Python language

It is now possible to develop applications on Nokia Internet tablets using Python. This was made possible by PyMaemo². Even though Python is not totally an official language of Maemo, Maemo website still provides documentation and tutorials for this language, and it seems to be totally supported by Maemo platform.

Almost everything that can be done in C on a Maemo platform, can be done in Python.

Pros:

- Probably everything can be done
- Native language for Maemo
- Graphics easy to do with Tkinter or PyGTK libraries
- Easy language in general

Cons:

- The project team is just a little familiar with Python.

Java

Java language is one of the most popular language among the world, mainly for its capability of running on many devices. Unfortunately, the N800 tablet is not really one of those, as it does not implement Java by default.

However, the purpose of the Jalimo project³ is to give developers the possibility of programming in Java on linux-based devices, including the maemo-based devices such as the N800 Tablet. Unfortunately, according to reviews found on Internet, Jalimo doesn't seem to be fully working (yet), which could lead to unexpected programming issues that might require time-wasting workarounds.

Pros:

- Highly portable language
- The project team is skilled in Java.

Cons:

- Uncertainty about how Java language is implemented on the device.

²<http://pymaemo.garage.maemo.org/>, may 2008

³<https://wiki.evolvis.org/jalimo>, may 2008

Other languages?

It is possible that other languages are available on the platform. However, their implementation is obscure and likely to be in an early development step. We did not dig too much into considering to use other languages than the ones mentioned previously.

Our choice

Considering the background of the project team, the first idea was to develop the project in Java. However, after some researches, we became aware of the likelihood of running into unexpected issues that may be hard to avoid because Jalimo might not be as robust and bug-free as we would like it to be. Python became a natural choice as it could give us guarantees that Java couldn't. Also, it is always exciting to learn a new language and Python is popular for being quick to learn.

We eventually decided to use Python in our project.

5.1.2 Server programming

The application that runs on the server does not have to be done in the same language as the one that runs on the mobile device.

The option of using Java has been considered immediately. Indeed, Java is perfectly suited for web-server-based application such as the one we need to develop. For this reason, it seems to be the best choice.

Also, the team members have already participated in developing a Java application on a server during the first semester project, and have a strong knowledge of the challenges that may arise and how to deal with them.

For these reasons, Java has been chosen as the server programming language.

5.1.3 Connexions between the server and the mobile application

This section will discuss how the different modules can interact together: on one hand the supermarket module in Java running on the server; on the other hand the client module in Python running on the Nokia.

XML-RPC is one possible solution. As a matter of facts, it is the technology used in Magnet projects at Aalborg University, and our project is one of those, so it totally makes sense to use it aswell. Research on the topic helped us understanding what it is.

XML-RPC is a communication protocol that allows a server to publish methods on a network, so clients can access them. A particularity of this protocol is that there are implementations in many languages... including Java and Python. Even better, the Python distribution we use on the Nokia device, PyGTK, natively implements an XML-RPC library so it is ready to use! This is not the case for the Java distribution we used but fortunately, it's just about downloading a few packages and including them to the project.

Conceptually, this protocol is a perfect match with our project. XML-RPC allows a server to publish methods on a network and what is a service made of if not a set of methods? We want the supermarket to offer a service on a local network, and this is just what this protocol allows us to do.

A specificity of this protocol we are aware of is the difficulty to send complex data types from the server to the client (and the other way around). However, the data we have to send are not very complex, except maybe the map of the supermarket that the server must return, and anyway, if issues were encountered, we probably can find workarounds easily. We just kept this aspect of the problem in mind while planning the implementation and the integration of this part of the project.

5.1.4 Database

The last component to form our system is the data storage on the server. In order to keep track of the customers inside the shop in an efficient and flexible way, we considered two options: using files, or a database.

Indeed, the server needs to keep track of the customers inside the shop: so it must store data. How can we do that?

First of all, using XML-RPC does not allow us to just keep this data in RAM memory. But would that have been a good option anyway? If the server had to be restarted for any reason, then the supermarket would be considered... as totally empty. Customer data would have been lost. We do not really want that.

So we have two solutions to store data: using files, or a database. To make a wise choice, let's consider what data we have to store:

- The map of the supermarket
- The position of the customers in the shop
- The shopping lists of the customers (that is used for determining the path to the next product)
- The next zoomed map of the customers
- The products to be picked up in the next area
- ...

Not forgetting the data about the cashier:

- Which cashier are open, and if they are ready to receive a customer

A quick look at this list convinced us that using file could only be a bad solution. This issue just asks for the use of a database, which allows the developer to easily read and write data of various types, which is exactly what we need to do.

About the type of database, the MySQL database seemed a natural choice: it's free to use and the project team has already used it successfully in the past. We knew that we will be able to do what we need to do with this solution.

So we decided to use a MySQL database.

5.2 Application on the mobile device

The implementation caused some modifications between the actual application on the mobile device and the theoretical one (cf. 4.2. Application on the mobile device). The description of the mobile application will be separated into three parts, the shopping list, the guidance in the supermarket and the checkout part. Please note that the programming language remains the same as we decided in the Design chapter, Python2.5 which will be used in a object oriented way.

5.2.1 Shopping list Manager

How are the shopping lists saved?

This part is described in the appendix (cf. A.3. Implementation of the saving of shopping lists).

Load a list

Most of the time, the user is only using one single shopping list that he modifies everytime he wants to add some products. That is why we decided to add the possibility of loading directly the last saved list in addition of the general option "Load a list". The name of this list is saved in another .txt file. In this way, he will save some time and will not get lost if he does not remember the name of his current list.

We also tried to avoid the problem of loading a file which is not a shopping list by adding filters in the file explorer that is displayed.

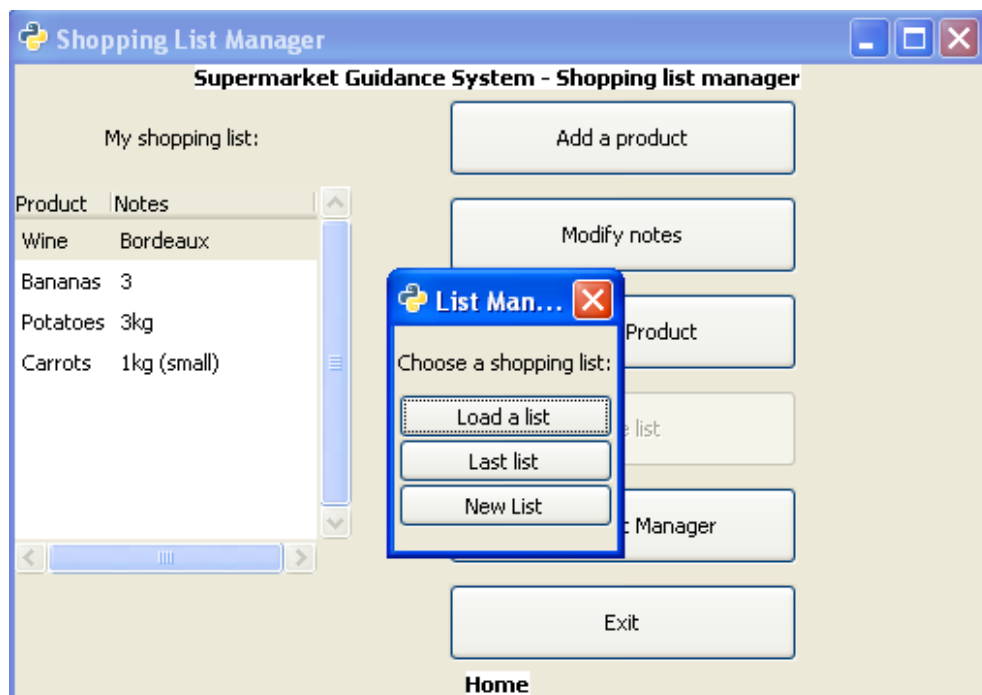


Figure 5.1: Shopping List - Load a list

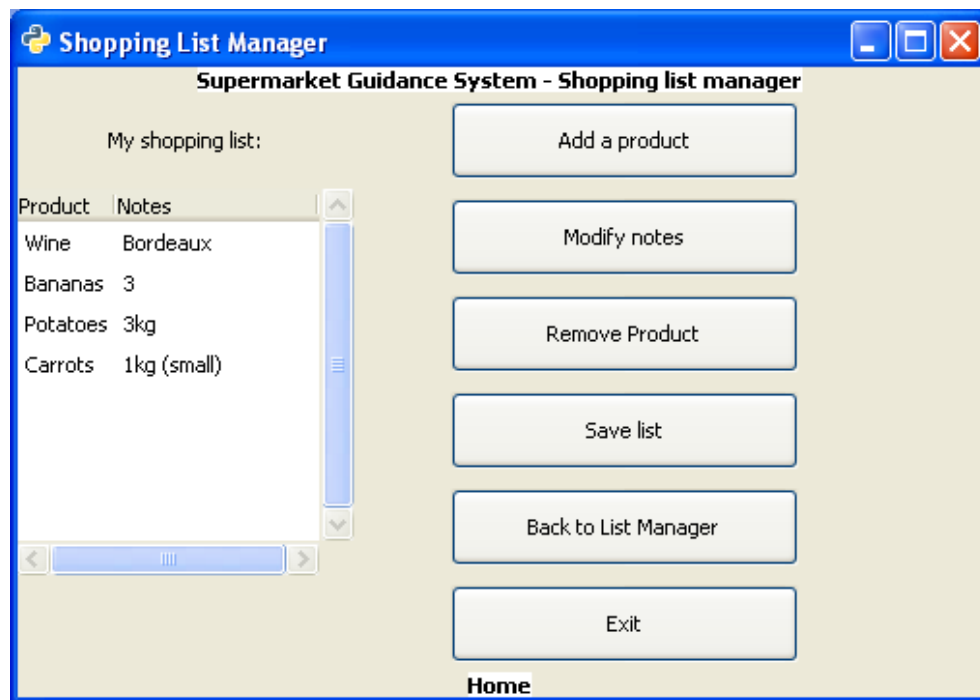


Figure 5.2: Shopping List - Home Page

Add a product

Initially, we wanted to have all the products in the same window. To display the ones the user wants, he was supposed to click on the expanding button of the category. But after many considerations and considering to the results of the survey for the usability tests (cf. 6.1.6. Results of the survey), we decided instead to replace the expanding buttons by buttons whose labels are the name of categories. When the user selects one category, a dialog window appears, it contains the list of check buttons with every product from this category and some text fields for the notes of each item.

To implement it, we added another class, `NeedAPdt`, handling all these differences.

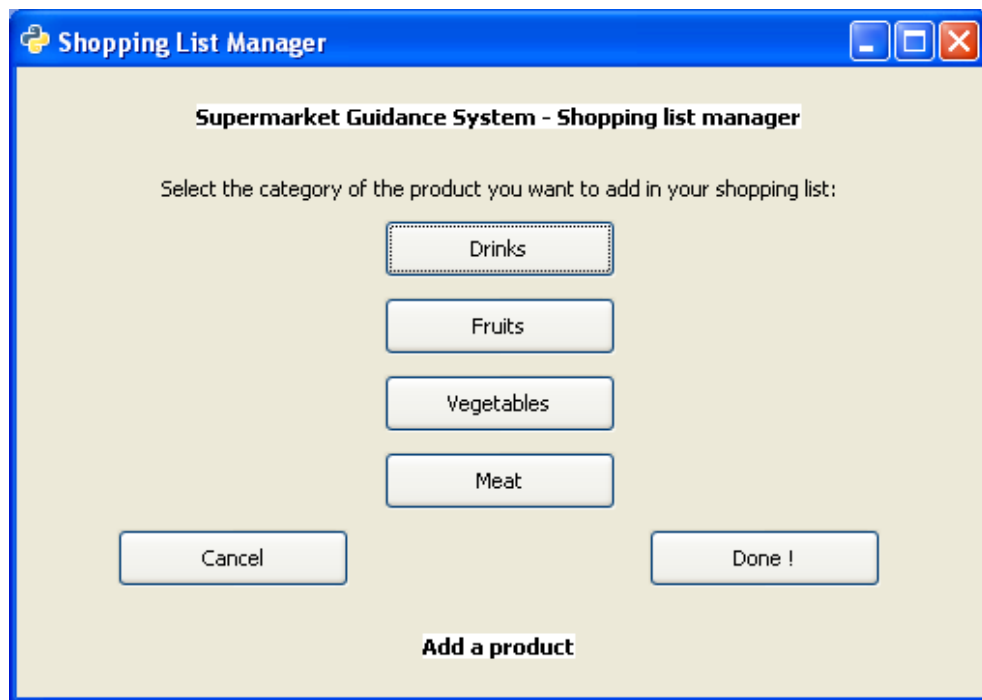


Figure 5.3: Shopping List - Choice of category Screen



Figure 5.4: Shopping List - Choice of products Screen

Save the shopping list

We decided to add some help for the saving of the list as the user could feel lost about giving a name everytime he saves his list. So in case it is a loaded list (not a new one), the current name of the list will

be suggested in the file explorer for the saving of the list (but the user can also save it with another name).

To be able to difference the two files we are using to save one single list, we are differentiating it by adding some characters in the name the user gives. For the list of products, we are adding "sl_" at the beginning of the name whereas for the notes file, we are adding "n_" at the beginning. For example, if the name given for the shopping list is "shopping_list1", the names of the associated files will be "sl_shopping_list1.txt" and "n_shopping_list1.txt". That allows us not to get confused in the files.

We encountered a problem when we loaded the program on the tablet PC because we developed on a Windows operating system whereas the mobile device is running on Unix. The handling of the files are different, on Unix, the way to a file is divided with slash '/' whereas on Windows, it is done with a backslash '\'. That means that the program was not able to find the files with the Windows version of the program, so we changed the slashes with backslashes.

Exit the application

We ran into a problem for exiting the application: something is still running when we close the application. This is a very big problem because on the tablet pc, the shell is totally blocked... It seems that this problem occurs in our application since the AddPdt object is created (when the Add a Product screen is displayed); since that moment, the application cannot be totally stopped. We tried to find out why but could not find the reason why.

We thought for a moment that it was due to the fact that the Home Page is not destroyed but modified for displaying the content of the Add a Product screen, so we modified the code to destroy the Home Page, to create a new page for the Add a Product Page and then to create again the Home Page screen when the Add a Product screen is closed but it did not change anything, there still was something running at the exiting of the program.

We noticed that Python is handling the destruction of the instances of objects by itself, there is no need to write destruction methods, so there could be a problem in these destructions where one is not done. We are using the version 2.5 of Python which seems to work well on the destruction of objects. But as in previous versions of Python (version x.x) many developers had problems with objects destructions, we tried to force the destruction by destroying manually the objects but some objects were never destroyed. We never found out why.

5.2.2 Guidance in the supermarket

How is the application launched?

Two options are possible: either the user launches himself the application when he wants to use it, or the supermarket system detects if the user owns the guidance system on his mobile device and launches the application on the mobile device.

We preferred the first solution, firstly because it doesn't make much sense for the supermarket system to be in charge of this task as our system is user-centric and not supermarket-centric, and secondly because the user can feel frustrated that the supermarket system can access his device without explicit permission. Moreover, checking if the user has our application installed on his mobile device seems to be quite complicated to implement.

Therefore, the user has to launch the application by himself to be able to use the guidance service.

The shopping list

After the user has chosen the shopping list he wants to use for shopping, all the products of this list are sent to the server by calling the function addCustomerProduct. This function is called for every single

product and the parameters given are the ID of the customer and the ID of the product to be added.

The display of the maps

For the display of the maps, we had several solutions (cf. 4.2.2. Guidance in the supermarket); the one we preferred initially is the use of xpm data.

Unfortunately, we then realized that xpm data was only used for the display of small icons and we could only get very small images. Even by modifying the content of the data by multiplying every character in width and height could not help to give a bigger image.

So we actually implemented our second choice, the list of integers. This list is get from the server thanks to the call of the functions `getMap` for the general map and `getZoomedMap` for map of the area. The data of the maps are firstly saved into a list; these data are integers and each value corresponds to an element of the supermarket (shelf, path, arrow, etc.). The maps giving directions to a specific area have 30×30 squares whereas the ones which give the locations of products have 10×10 squares.

Then a graphic table is created. A Drawing Area is added to every square of this graphic table. To display the map, for every square of the table, we check in the list the value of the integer corresponding to this part of the map and regarding to its value, we modify the background color of the drawing area.

As you can see in the figure below (cf. Figure 5.5), the white color corresponds to the walking space, blue to the shelves, green to the way to follow, orange to the area where the user needs to go and red to the cashiers.

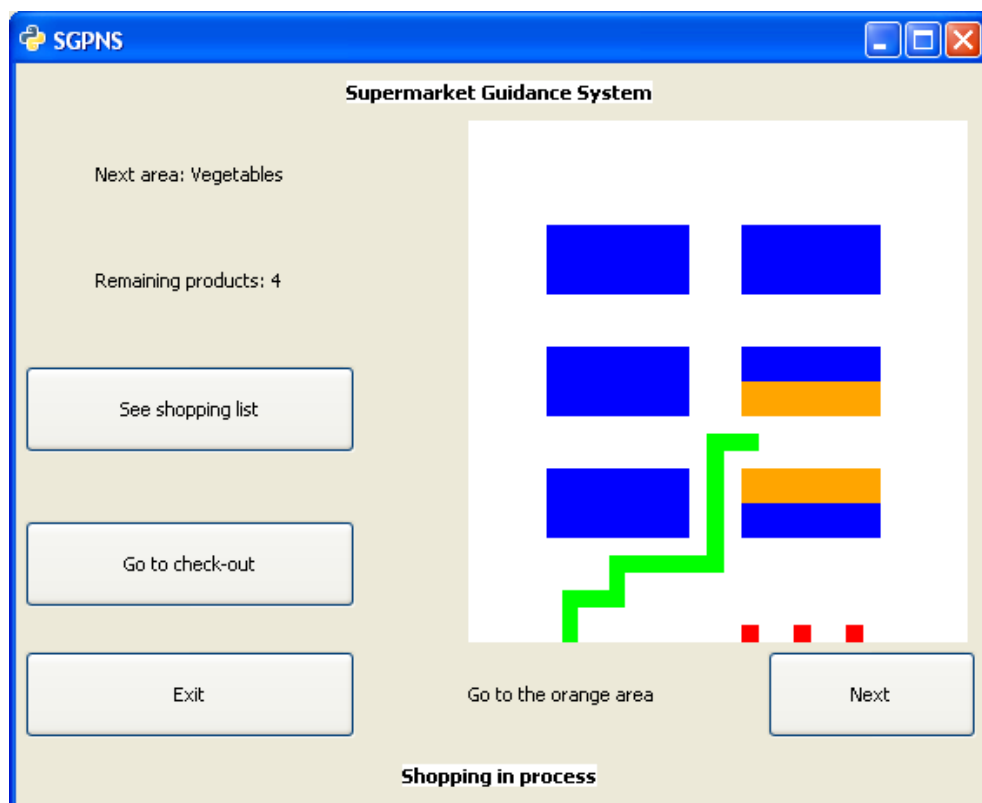


Figure 5.5: Guidance in the supermarket - Go to area Screen

About the maps giving the locations of the products, as you can see in the figure below (cf. Figure 5.6), every location is shown with a specific color. The list of products to be picked up displayed next to

the map gives the corresponding color to the foreground of the label of each product. (Notes: the color white corresponds to the walking space and the color blue corresponds to the shelves.)

To continue shopping and access the map showing the next area to go, the user has to press the button "Continue Shopping".

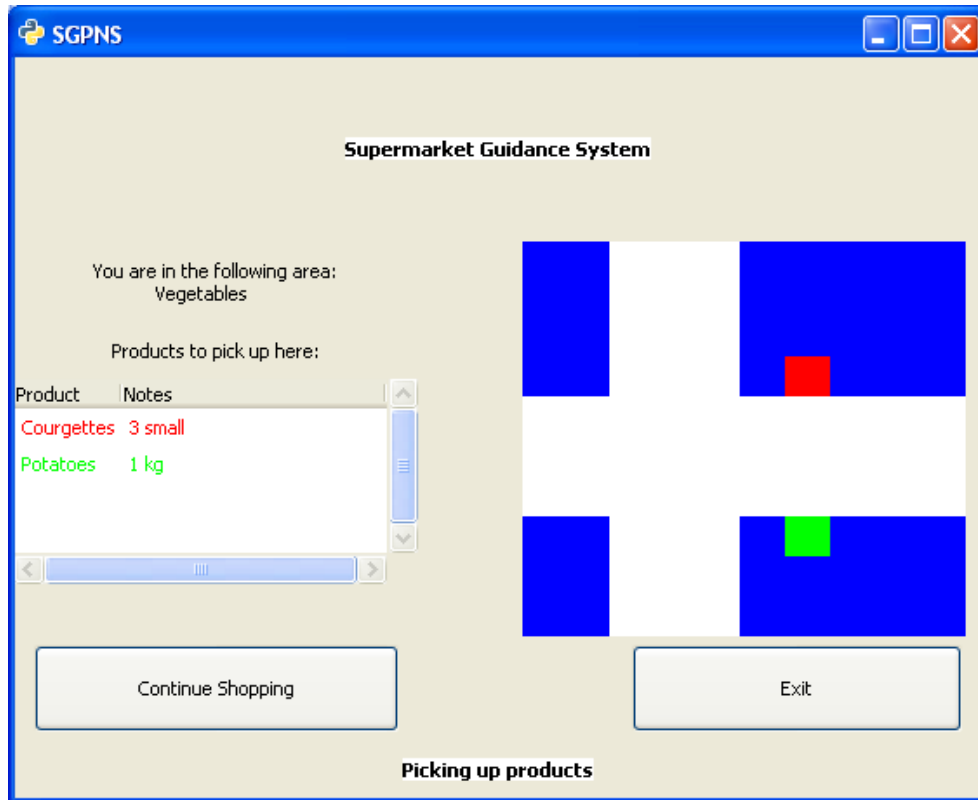


Figure 5.6: Guidance in the supermarket - Pick up products Screen

The display of the maps takes some times, especially for the ones which give directions to an area, as they are bigger but also because the server is calculating the best path during that moment. It takes around 30 seconds for the first maps and 20 second for the smallest ones. We will observe during the tests if these delays are acceptable by the users.

5.2.3 What about the localisation of the customer?

Unfortunately, we did not have enough time to integrate Magnet localisation system in our system. Therefore, we added another button on the screen "Next" which is simulating that the customer arrived at the right area. By pressing it, the user can access the map with the locations of the products for this area.

5.2.4 When is the shopping finished?

The mobile application computes by itself when there is no product in the list anymore. When the shopping is over, the map displays the way to go to the cashiers. At that point, the application only allows the user to go to check out or to exit the application.

5.2.5 Checkout

About the checkout part, the application needs to update the waiting number of the user. We are using the functions `goToCheckout` to initialize the request of going to check out and then `getPlaceInLine` to achieve the update of the waiting number. We decided to update it every 2 seconds. This task is done in a separate thread not to block the interface.

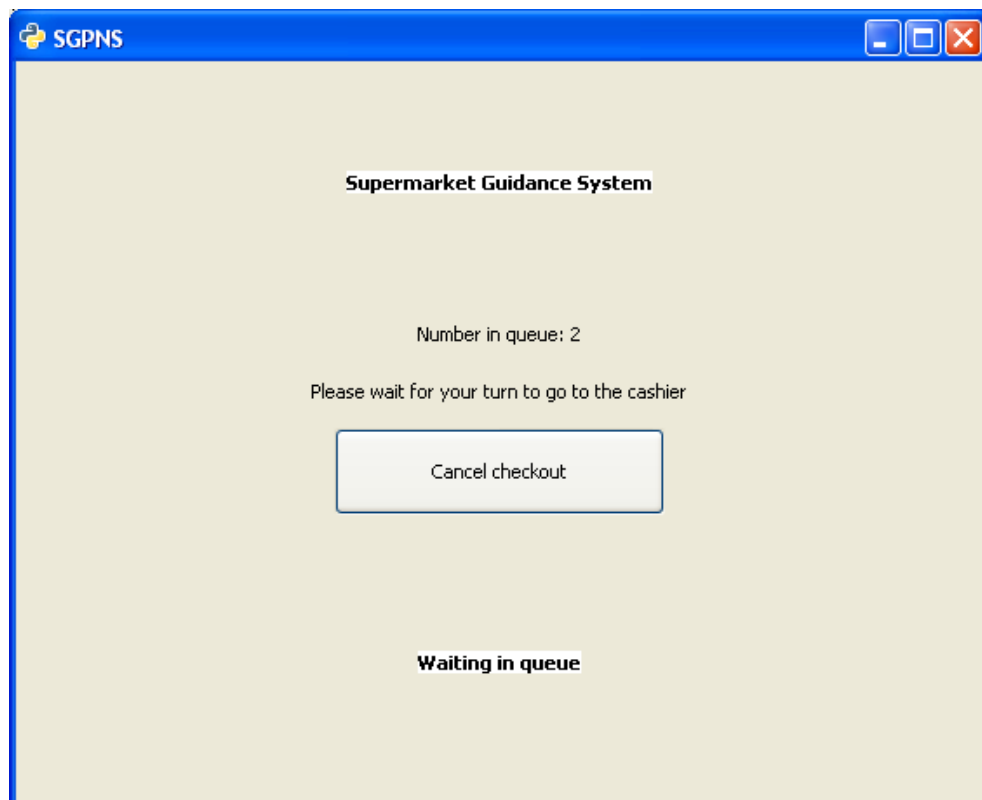


Figure 5.7: Guidance in the supermarket - Waiting in the queue

During that part, the user always can cancel his waiting number and going back shopping.

Finally, when the waiting number is zero, we call `getCheckoutNumber` function to get the number of the check-out where the user needs to go.

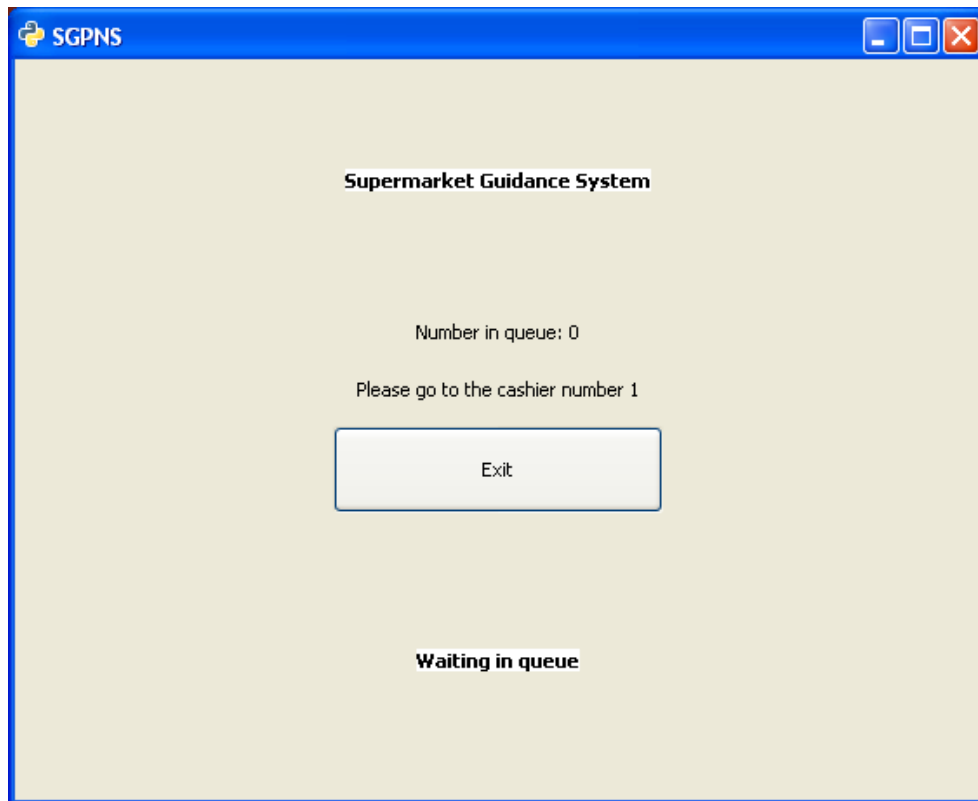


Figure 5.8: Guidance in the supermarket - End of the queue

5.3 Supermarket System

The supermarket system is divided in two main entities, the Guidance and the Checkout. However, it is implemented as two modules together in the same java program. This program is made of the following classes.

Main: Starting point of the program, starts the server

JavaServer: Runs the xml-rpc server. Loads the ConnectionHandler class as an accessible class from the outside.

ConnectionHandler: The main class of the program. It includes every method that the server publishes to the outside world, so the client can interact with it.

DatabaseManager: Used to interact with the database in order to execute instructions and queries, in order to read and write data.

Note: The Builder class is useful to create a new map from a .csv file and write it into the database. It is only used when the organization of the supermarket changes, and will not be described in details.

5.3.1 Common class: the Database Manager

As both the guidance system and the checkout system are, in practice, part of the same software, there is one general class that both of them are using: the DatabaseManager class. It is used to interact with the database in order to read and write data, something both the Guidance module and the Checkout module need to do. Let's have a look at this class.

When an instruction must be executed on the SQL server, a similar code is used:

```
String line = "INSERT INTO customers_shopping_list (id, id_product, taken) VALUES ('" +
idCustomer + "','" + idProduct + "','0')";
DatabaseManager.executeSqlInstruction(line);
```

The instruction is sent to the DatabaseManager using the executeSqlInstruction. Very simple. The DatabaseManager takes care of the rest, as it can be seen in the executeSqlInstruction method.

```
static void executeSqlInstruction(String strInstruction)
{
//Connecting to the database
Connection connection = null;
try {
Class.forName("com.mysql.jdbc.Driver");
connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1/supermarket","root","");
Statement instruction = connection.createStatement();
instruction.execute(strInstruction);
connection.close();
} catch (Exception e) {
System.out.println("SQL Instruction Error: " + strInstruction);
e.printStackTrace();
}
}
```

A connection to the database is initialized, then the instruction is executed. Finally, the connection is closed. If a problem occurs, an exception is raised and its summary is displayed in the server console, with the faulty instruction, for debugging purposes.

The first idea was to deal with queries in a similar way as for instruction. The difference between an instruction and a query is that no result is expected from an instruction (for example: "insert an entry into a table"), whereas a query execution returns a results (for example, "select all entries in the table where the id of the customer has a value of 5"). Firstly, we wrote a very similar method to execute queries; the only difference was, it was returning a ResultSet including the results. Unfortunately, we figured out that this ResultSet was not accessible, as the connection was closed in the method (by the connection.close; instruction), and a ResultSet cannot be accessed if its connection has been closed.

The question was: how important is it to close the connection? After all, the database is supposed to be able to handle unclosed connections, so we looked into this possibility for the purpose of simplifying the coding process.

We figured out that during a normal shopping process, many queries are executed; that is, a few **thousands**. We kind of knew that it was probably not going to be possible to let thousands of connections, whatever the robustness of MySQL servers regarding unclosed connections. Nevertheless, we ran a stress test with constant queries being executed on the database. The SQL server crashed after 5 minutes; the error message read: "Too many connections".

We modified the method of the DatabaseManager and the way we use it. This is how the method eventually looks like.

In this version, the method receives the connection as a parameter, so the function that needs to call executeSqlQuery needs to create a connection first. Then, it will use the returned results, and when it is done using the results, it will close the connection on its own. We ran the same stress test as previously with this new version and the server never crashed.

```

    static ResultSet executeSqlQuery(Connection connection, String strQuery)
    {
        ResultSet results = null;
        try {
            Statement instruction = connection.createStatement();
            ResultSet resultsTmp = instruction.executeQuery(strQuery);
            results = resultsTmp;
        } catch (Exception e) {
            System.out.println("SQL Query Error: " + strQuery);
            e.printStackTrace();
        }
        return results;
    }

```

5.3.2 Guidance

Most of the methods of the server are specific to the Guidance module. These methods are included in the ConnectionHandler class, which is the only class whose methods are accessible from the outside. The methods included are:

- boolean addCustomerProduct(int id_customer, int id_product)
- ArrayList<Integer> getMap(int id_customer)
- ArrayList<Integer> getZoomedMap(int id_customer)
- String getNextCategoryName(int id_customer)
- String getProductName(int id_customer, int number)

Some methods are more important than other and deserve more attention.

boolean addCustomerProduct(int id_customer, int id_product)

This method is used to build the shopping list. The server will add the product with the id id_product to the customer with the id id_customer.

ArrayList<Integer> getMap(int id_customer)

This is the most important method, in term of length and processing. This method builds the map that must be shown to the customer by following the process described in section 4.3.1, "Providing directions to an area". However, the implementation slightly differs from the initial plan established during the design phase.

First of all, writing this part of the software lead to adding new tables to the database, as we needed to temporary save much information regarding the search for the closest spot. Indeed, several information must be saved, such as the list of the possible destinations (the customer's spots, in the customers_spots table) or all the possible path of a user through the shop (only used while the algorithm is running).

The algorithm is as followed:

- Initial state: Starting from the customer's current position, with a cost of $c = 0$
- (Start of iteration) Checking each node (pixels on the map) where the user can go (ie that are walkable: see section 4.3.1, "Providing directions to an area") from all nodes with a cost of c . For each node

- Make sure the node has not been processed already
- Check if it is one of the possible destinations (if yes, exit the algorithm)
- Add it to the customers_path table with a cost of the parent's cost + 1, and the coordinates of the parent
- Increase c by 1
- (End of iteration)

You can find the corresponding code in the appendix (cf. A.5. Code of the server - guidance)

We can see that the loop implies many queries to the database: this is where the thousands of queries we experienced while running stress tests came from. We can also see that the algorithm itself is not that big, given that many lines of code are necessary to close the ResultSets and the connections after each SQL query.

Let's have a look at what is happening when the final destination is found. Actually, this is where the biggest part of the process is. As it is easier to understand through an example, let's consider that the closest area found is the Drinks area and that the user has two drinks in his shopping list: Wine and Water.

- Removal of the spots of the category reached from customers_spots. For example, if we found out that the closest area is the Drinks, then all the Drinks spots (usually there are two) are removed from the table customers_spots, as they are not potential destination anymore (the customer will not need to come back to this area in the future).
- Set products as taken. In the shopping list, the Wine and the Water are set as taken, as they soon will be.
- Build the map. Starting from the basic map of the supermarket, the path from the destination (the Drinks spot that was reached by the algorithm) to the starting point (the current position of the user) is enlightened. This path can be found as each pixel of the path was saved in the customers_path table of the database, and includes the coordinates of the parent pixel. The map is created in a two-dimensional array of Integers (int[30][30] map).
- Update the current position of the customer (the destination spot is now considered to be his current position).
- Create the zoomed map, an smaller (10*10) two dimensional array of Integers. This map will be saved in the customers_zoomed_map table and will be return when the client calls the getZoomedMap function. It is more convenient to build the zoomed map now than later, even if it may seem relevant to build it only when the client asks for it.
- Returning the map to the client*

*About that last task, we ran into some problems due to the fact that xmlrpc doesn't like complex datatypes. What we wanted to do in the first time was returning a one dimensional array of Integers, as this is what the client programmer would rather receive. But an xml-rpc was raised. This could later be fixed by setting the option enabledForExtensions of the xmlrpc server. However, even if the data seemed to be sent properly, the client would never receive it correctly. We eventually found out that returning an ArrayList instead of an array of Integers would solve the problem. This discovery was much appreciated as we had already started to implement a workaround that implied sending each pixel one as Integers one by one (and the map is made of 900 pixels), which would surely have caused some latency issues.

```
ArrayList<Integer> getZoomedMap(int id_customer)
```

As the zoomed map was created earlier, this is a very simple method that loads the zoomed map from the database, then sends it back to the client in the proper data type, an ArrayList.

String getNextCategoryName(int id_customer)

This methods sends back the name of the next category that will be visited by the customer.

String getProductName(int id_customer, int number)

A zoomed map can include from one to five products. For example, it can include Wine and Water if both products are in the customer's shopping list. However, the client doesn't know which product is the wine and which one is the water; it has to call this method to be returned the name of the product. The number parameter is the number read in the zoomed map: it identifies the product but only the server knows the reference.

5.3.3 Checkout

The other part of the server is the checkout, that deals with the virtual waiting queue. It actually includes six different methods: three for the client, and three for the cashier.

The implementation of this part of the system has been going particularly well, as no real problem was encountered. Probably the fact that it is quite a simple part helped a lot.

The three methods available to the client are:

- void goToCheckout(int id_customer), which will let the client enter the end of the virtual queue
- int getPlaceInLine(int id_customer), which will return the place in line of the client
- int getCheckoutNumber(int id_customer), which will return the ID of the checkout the customer has to go to

The three methods available to the cashier are:

- void openCheckout(int id_checkout), which will let the server know that the checkout is open
- void nextCustomer(int id_checkout), which will let the server know that the cashier is ready to receive a new customer
- void closeCheckout(int id_checkout), which will let the server know that the checkout is now closed

5.4 Supermarket System - the check-out application

The check-out application allows the communication with the server in order for it to get every information about the opened check-outs. The open a check-out is done thanks to the call of the server function openCheckout() with the number of the opened check-out in parameter. The cashier can also ask for a next customer, function nextCustomer(), every time he has finished with a customer. Finally, he closes the check-out, the function closeCheckout() is called.

The interface we built for this part is quite simple as it owns only four buttons, one for each function and one to exit the application.

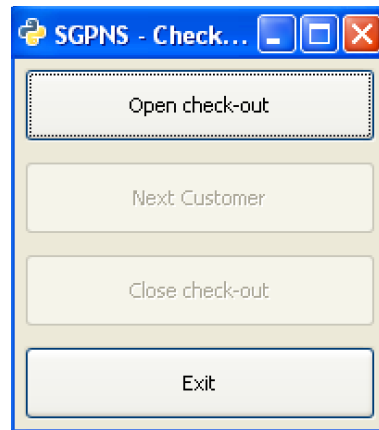


Figure 5.9: GUI for the check-out assistant - check-out closed

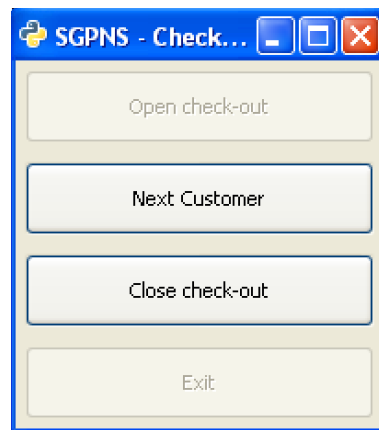


Figure 5.10: GUI for the check-out assistant - check-out open

6

Tests

6.1 Usability tests

While we were thinking about how to build the GUI of the application on the mobile device, we thought that the best way for the user to get the most ergonomic interface would be to allow him to test it and to comment it.

Therefore, we decided to build a low-Fidelity prototype (or low-Fi prototype) and to organize some usability tests.

6.1.1 Usability tests and user centred design

One particularity of our project is that it is a user-centric application. In comparison with the other pre-existing Intelligent Supermarket Systems, we are changing the viewpoint, it is not only helping in the supermarket but it is helping in the everyday life from the building of the shopping list to the payment of the products.

In concordance with this concept, we really wanted to adapt it perfectly with the users. That is why we decided to build our application with a user centred approach. It means that we firstly need to think about who is the final user is and to find his characteristics. Some characteristics can have an impact on his needs regarding our system but also on the design of our application. For example, if our user-target is elderly people, a larger font size should be used; if they are children, we will add images instead of labels on the buttons and try to relate the application to a game.

In our case, the user-target is very wide (cf. 3.1. User Definition) so it is more complicated to find some characteristics corresponding to all of them. Therefore, we decided to focus on the logic of the interface, to make it easily understandable by every kind of users. That is the reason why we organized these Usability tests.

6.1.2 Low-Fidelity Prototype

We used Microsoft Office Visio to build the screens of the Low-Fidelity prototype. The whole low-Fi prototype is shown in the Appendix part (cf. A.1.2. Low-Fi Prototype).

Here are two examples of this low-Fi prototype:

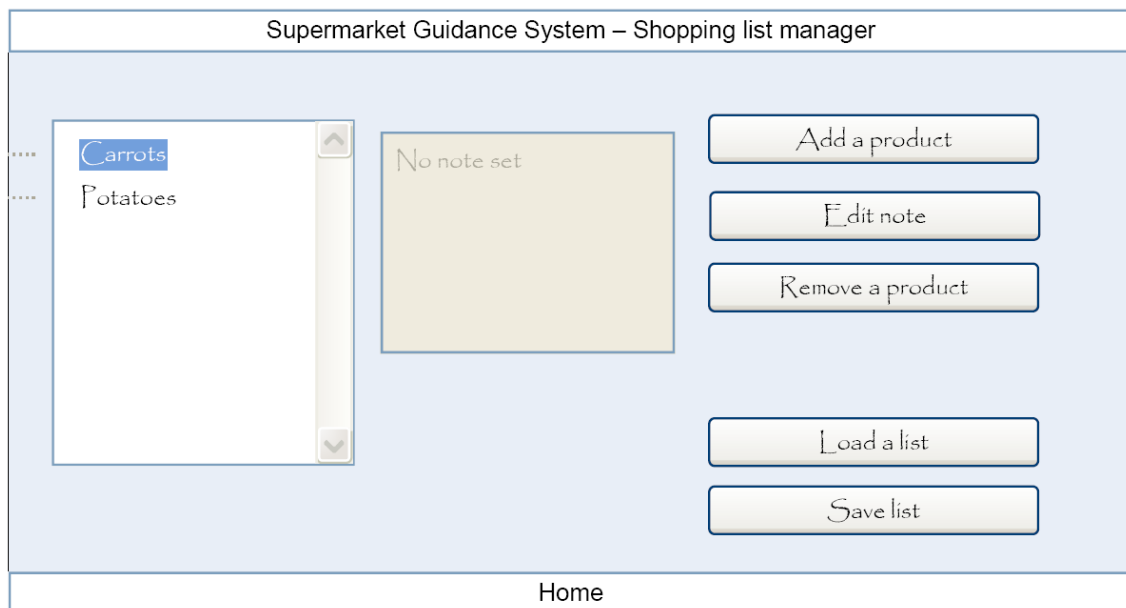


Figure 6.1: Shopping list manager

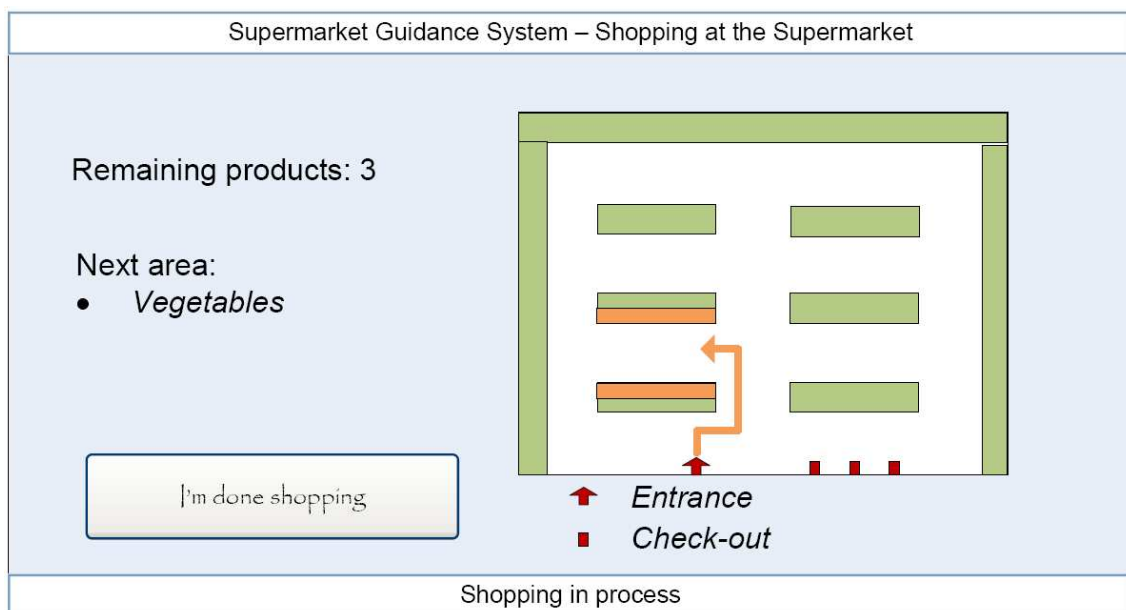


Figure 6.2: Guidance in the supermarket

6.1.3 Organization of the tests

The testers

Six testers took part in those usability tests, they are in the same age range: between 21 and 24 years old. At the beginning, we considered that our user panel could be divided into two parts. One group is composed of four people belonging to our study programme whereas the other one is composed of two people who are not studying a programme associated with computer science or mobile technologies. Both are very interesting for us because the first group is supposed to be capable to point out mistakes in a designer point of view but also a few mistakes in the logic of the GUI when they do not understand how

to do an action. The second one is more representative of the people who will use our application, they usually have more difficulties with the use of mobile systems so they are able to point up more problems in the logic of the interface.

To define more precisely the testers, we asked them some personal details in a global questionnaire. As you can see in the Figure 6.1., most of the testers are used to go shopping quite often, only one goes rarely to the supermarket. The Figure 6.2. shows that most of the testers feel quite comfortable with mobile devices but only one feels very comfortable and another one does not feel comfortable. So actually our testers group was more diversified than we initially thought, it is divided into three parts: the ones who are very comfortable with mobile devices, the ones who are quite comfortable and the ones who do not really feel comfortable.

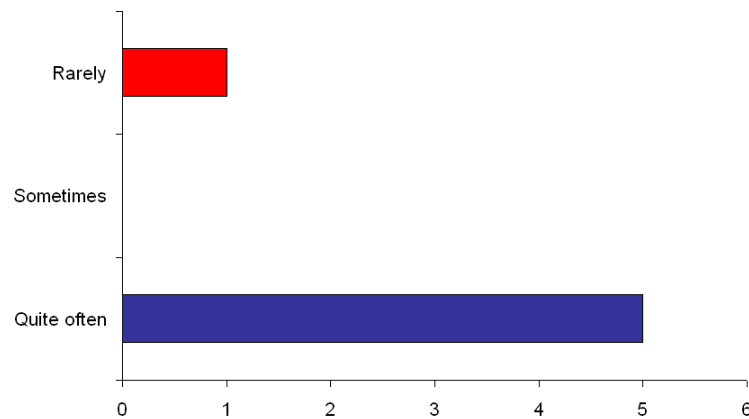


Figure 6.3: Do you often go shopping?

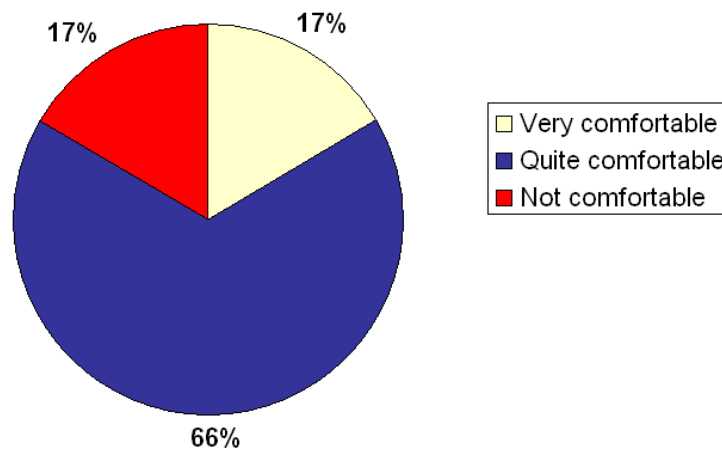


Figure 6.4: How comfortable are you with the use of mobile devices such as PDAs, Smartphones or Tablet PCs?

The tests room

The tests were organized in a laboratory specific to usability tests. The tester is comfortably seated on a sofa. Two people are taking care of the tests, one is talking with the tester (explaining the process and answering the questions), another one is in charge of changing the screens of the low-Fi prototype regarding to the actions done by the tester.

This room also allows to record the tests on a video thanks to video cameras installed in every corner of the test room and another room next to the test room where the recording system is monitorable. We preferred not to use this recording possibility because we would have needed a third person to take care of it but also because we thought that taking notes during the test would be enough to get the tests results.

The tests

These tests were proceeded on the 1st of May. For each tester, at the beginning, we read a text explaining him what is the goal of the application and how the tests are organized. Then, we provide the tester a tasks list to achieve and the first screen of the application. The user has to say what he is doing on each screen to do this task. While he is describing his actions, we are changing manually the screens displayed in front of the user. When the user is done with his tasks, he has to fill a survey composed of questions and check boxes answers which allows us to evaluate the opinion of the tester about the GUI. We decided not to suggest any neutral answer in this questionnaire to force the tester to take a clear position. The tester can also add some comments on this sheet. The fact that a tester succeeds in doing a task, does not mean that our design choices are the best ones; it only means that they were understandable. That is why we also use this survey to answer to some specific design questions. Some are about the way of building the shopping list (if it is easy and if it is the best way), some others about the way of providing directions in the supermarket (if it is understandable and if it is the best way) and finally, there is one question asking if the user would prefer having the whole list displayed on the screen while he is shopping (as there are only the products from the next department displayed).

You can find the introduction text, the tasks list and the survey in the Appendix (cf. A.1.1. Presentation of usability tests). The answers we get to the survey are also presented in the Appendix (cf. A.1.4. Answers to the survey)

6.1.4 The tasks list

In this part, we will explain the actions the user is supposed to do for every task part of the tasks list. After each action the user does, the image of the new screen displayed will be included in order to show the effects of this action in the program and to understand the next action the user has to do to complete the task he is asked to achieve.

Add Potatoes to the shopping list and specify you want to buy two kilos

This first screen is the home page of the application.

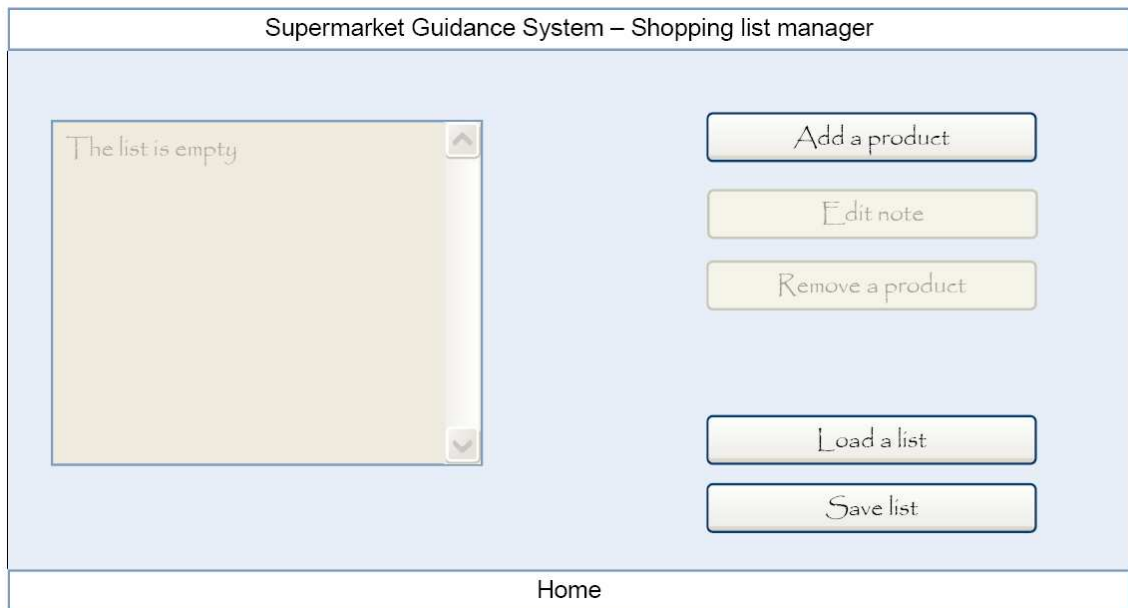


Figure 6.5: Home Page with an empty list

On this screen, the user has to click on the button "Add a product". This screen will be displayed:

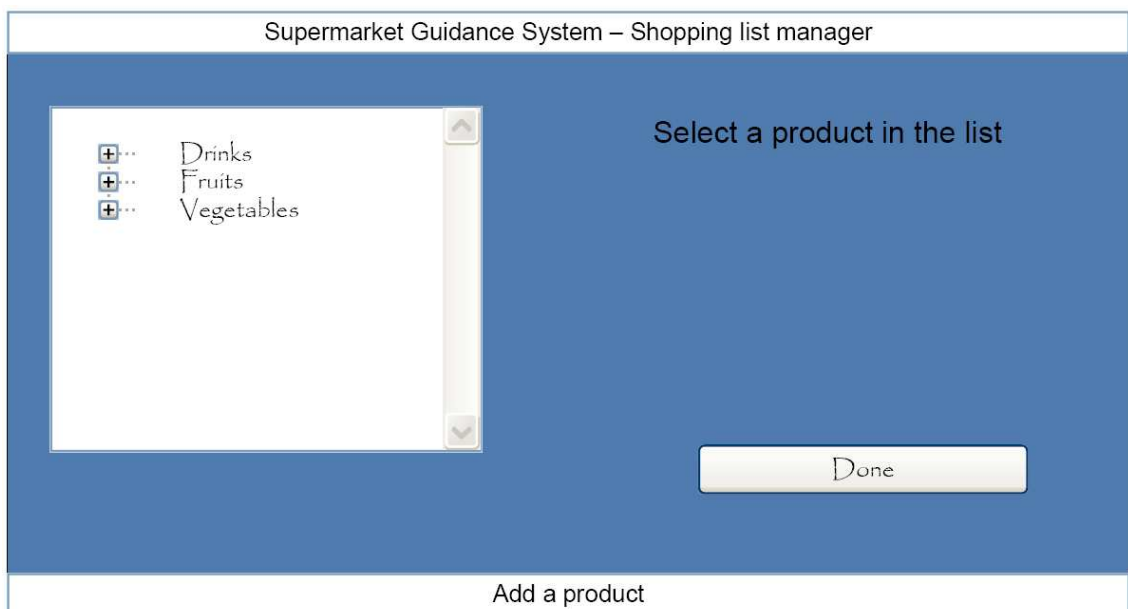


Figure 6.6: Add a product screen

He selects the category "Vegetables" as it is the category of the product "Potatoes".

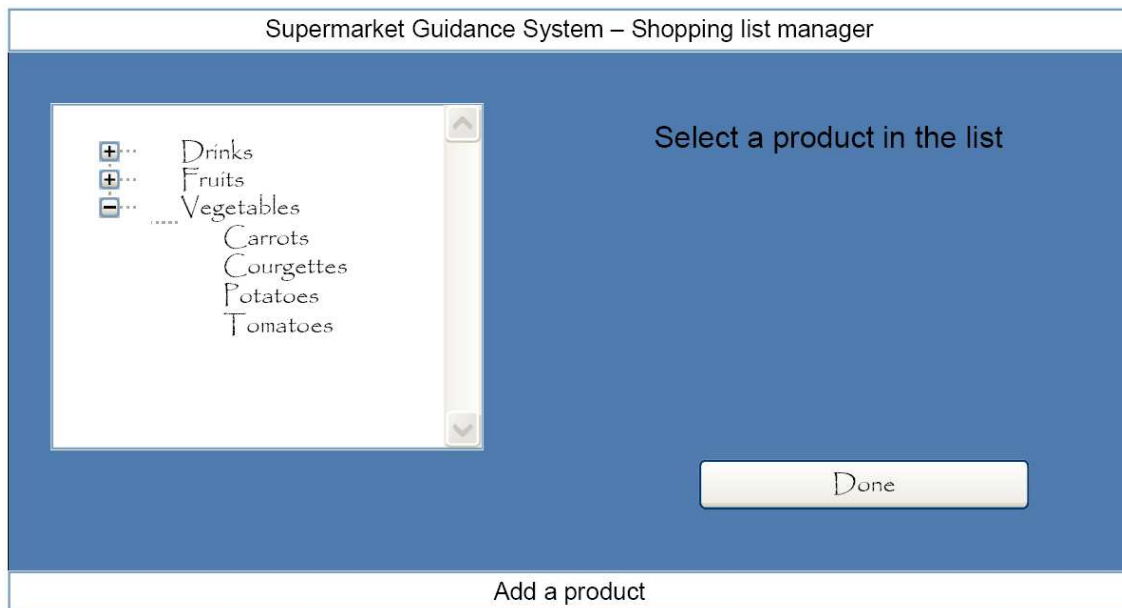


Figure 6.7: Add a product screen - Vegetable Selected

Then, he selects "Potatoes".

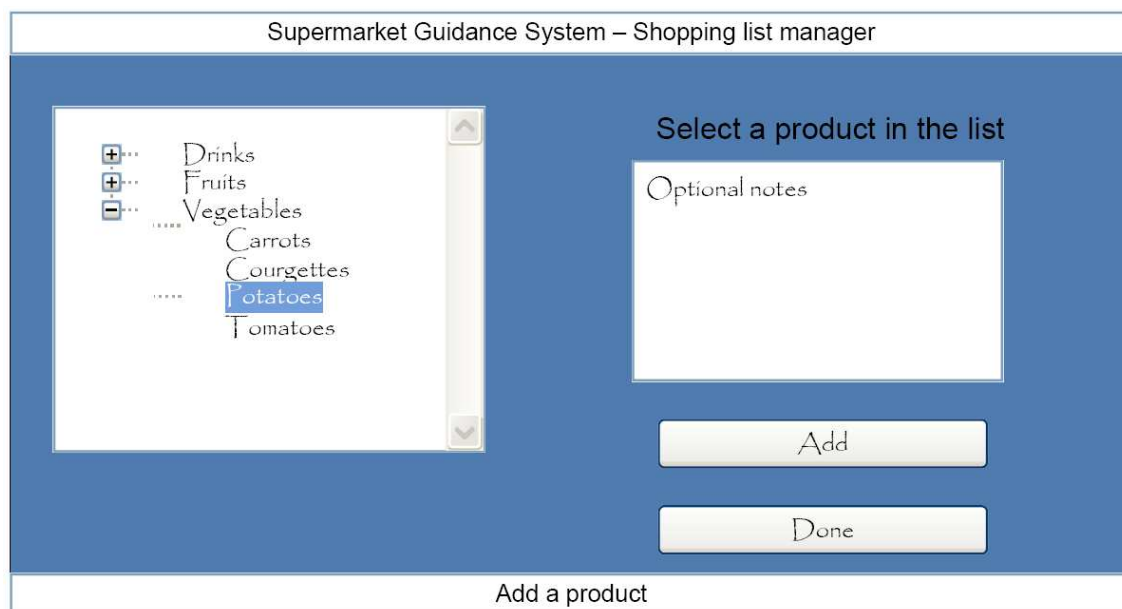


Figure 6.8: Add a product screen - Potatoes selected

He has to write in the Notes field 2kg and clicks on the button "Add".

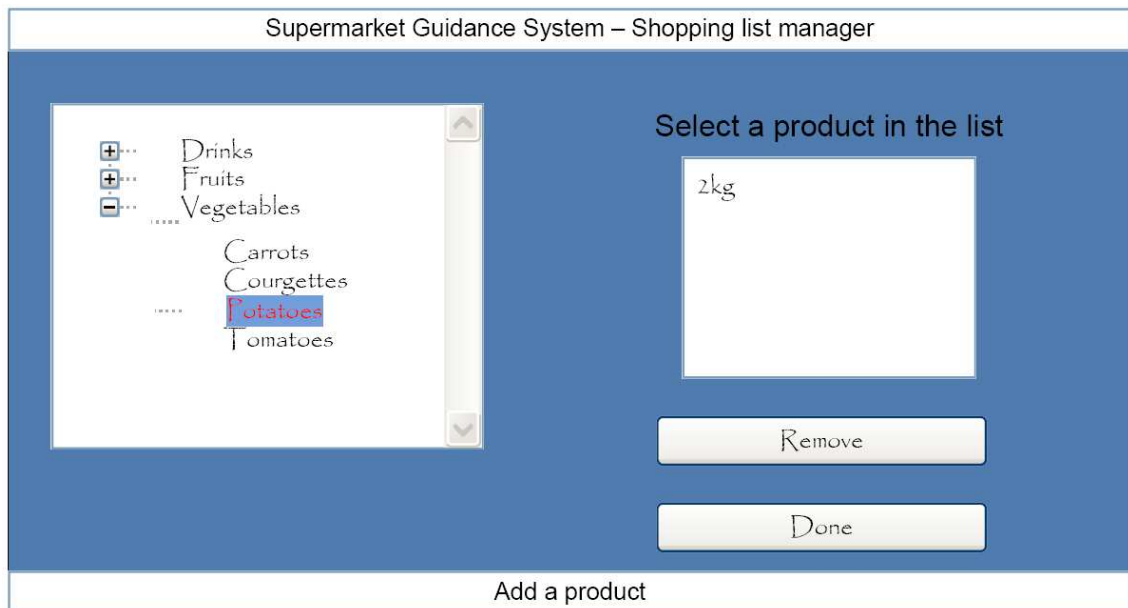


Figure 6.9: Add a product screen - Potatoes added

Add Carrots to the shopping list

The user selects "Carrots" in the list.

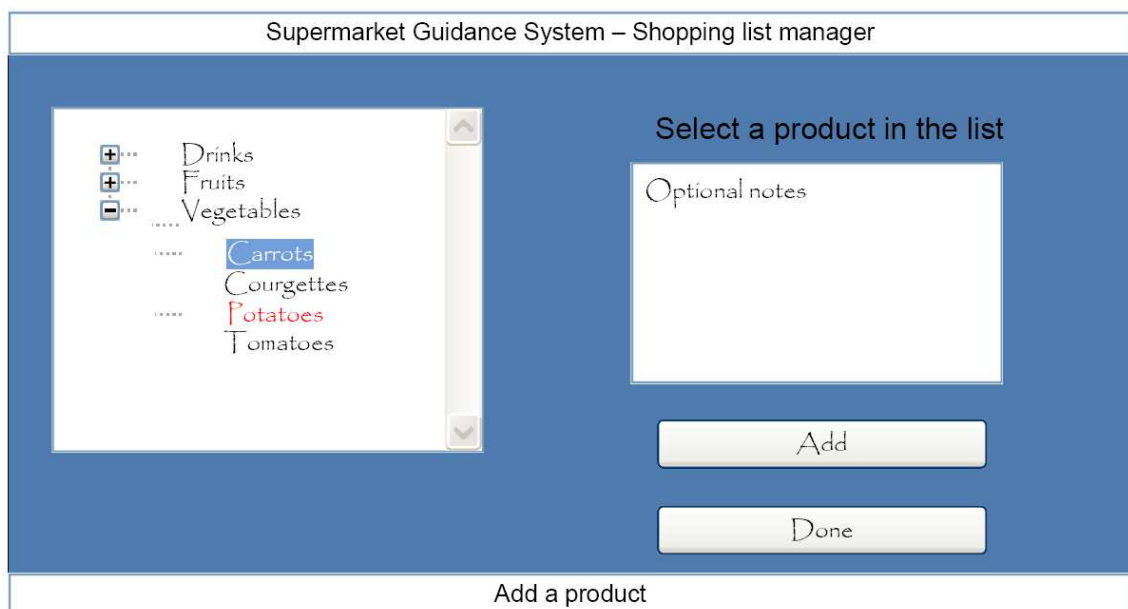


Figure 6.10: Add a product screen - Carrots selected

He clicks on the button "Add".

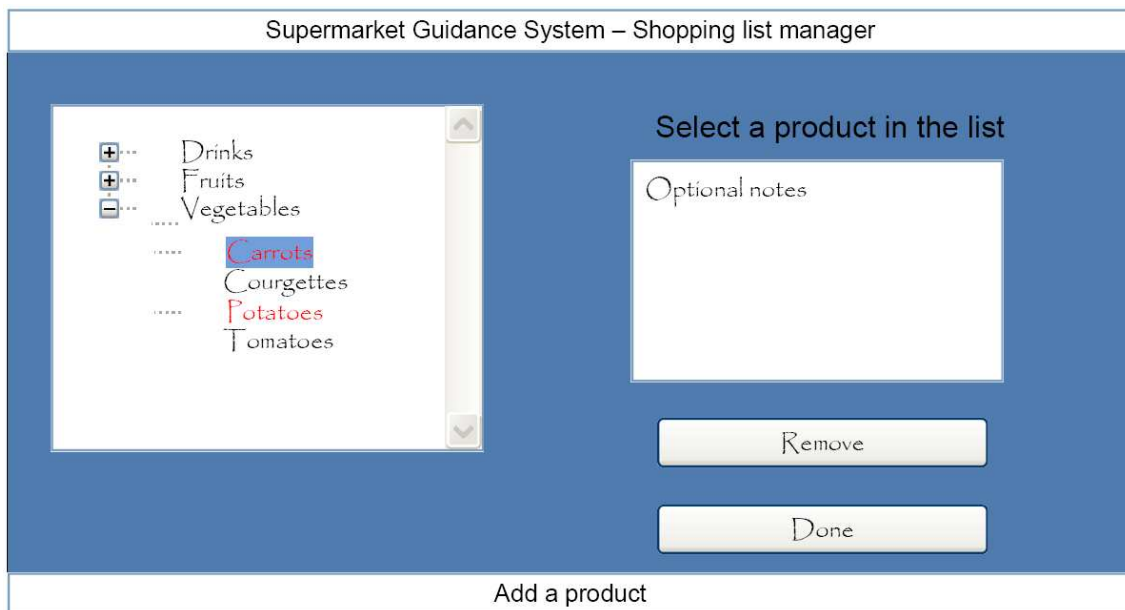


Figure 6.11: Add a product screen - Carrots added

Then he clicks on the button "Done" to come back to the main screen.

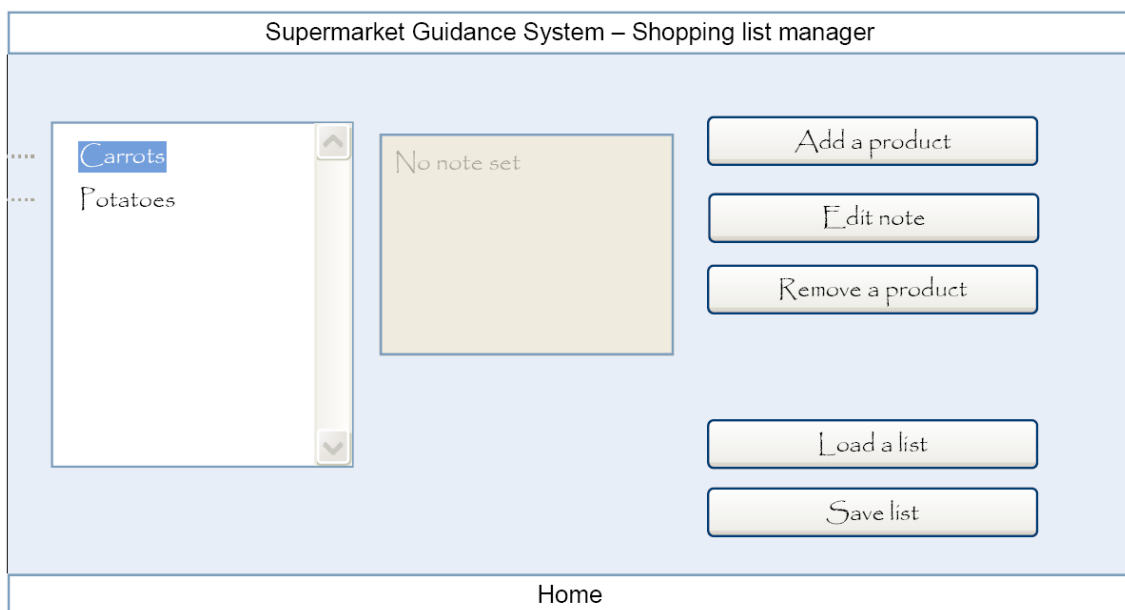


Figure 6.12: Home Page screen

Specify that you want 1 kilo of small carrots

The user firstly has to select "Carrots" in his shopping list.

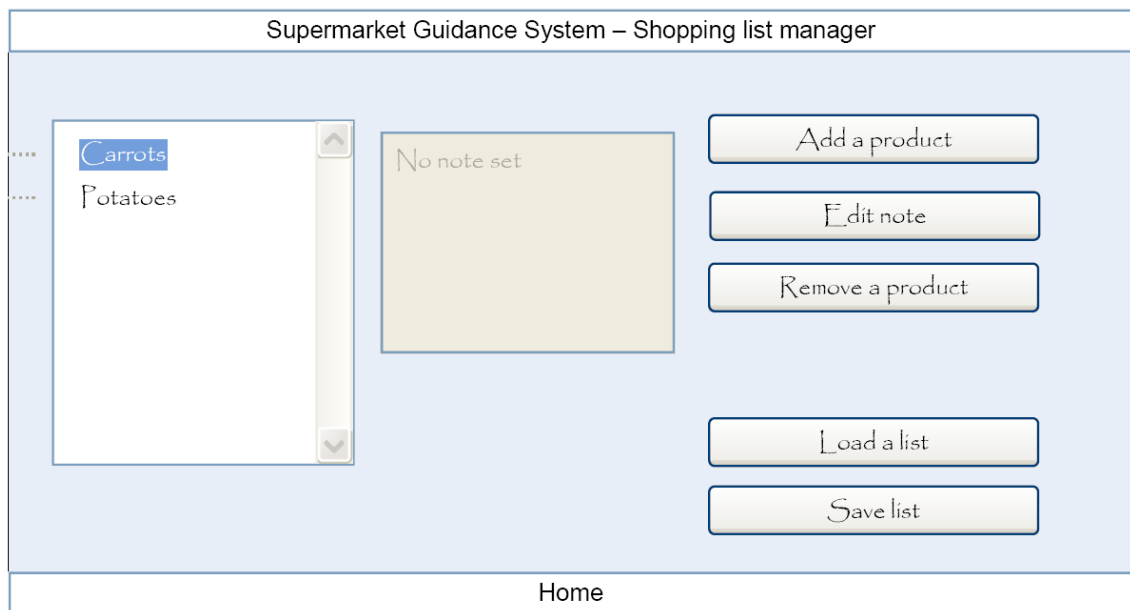


Figure 6.13: Home Page screen - Carrots selected

He presses the "Edit Notes" button.

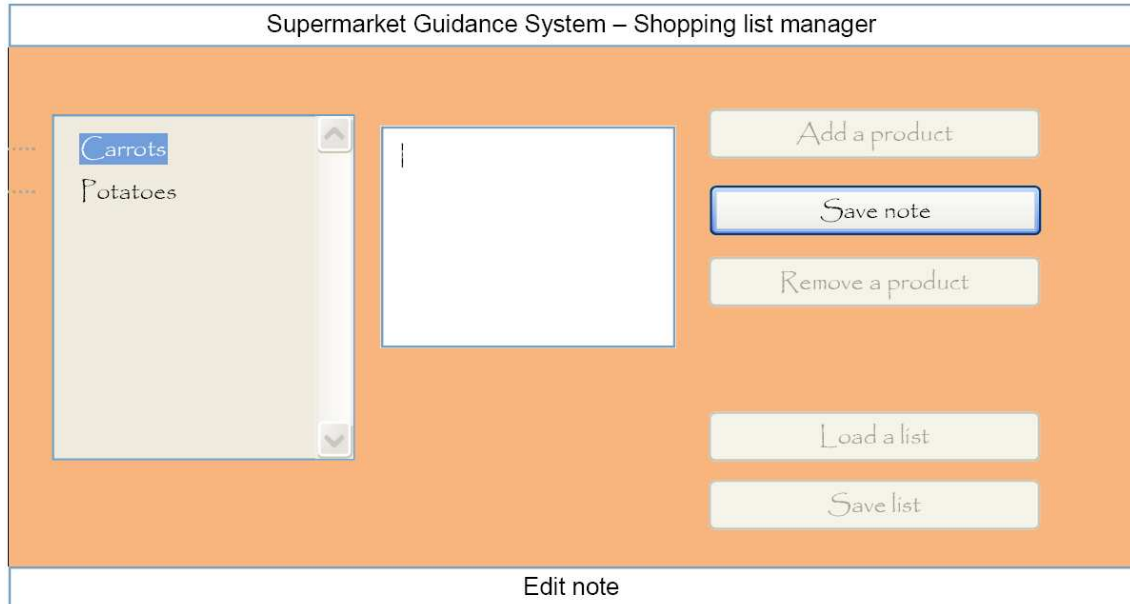


Figure 6.14: Edit Notes

Then, he writes in the Notes field that he wants 1 kilo of small carrots and presses the "Save Note" button.



Figure 6.15: Home Page screen - Carrots selected

Save the shopping list

The user has to click on the button "Save list". Then, he writes the name he wants for the list and presses the button "Save".

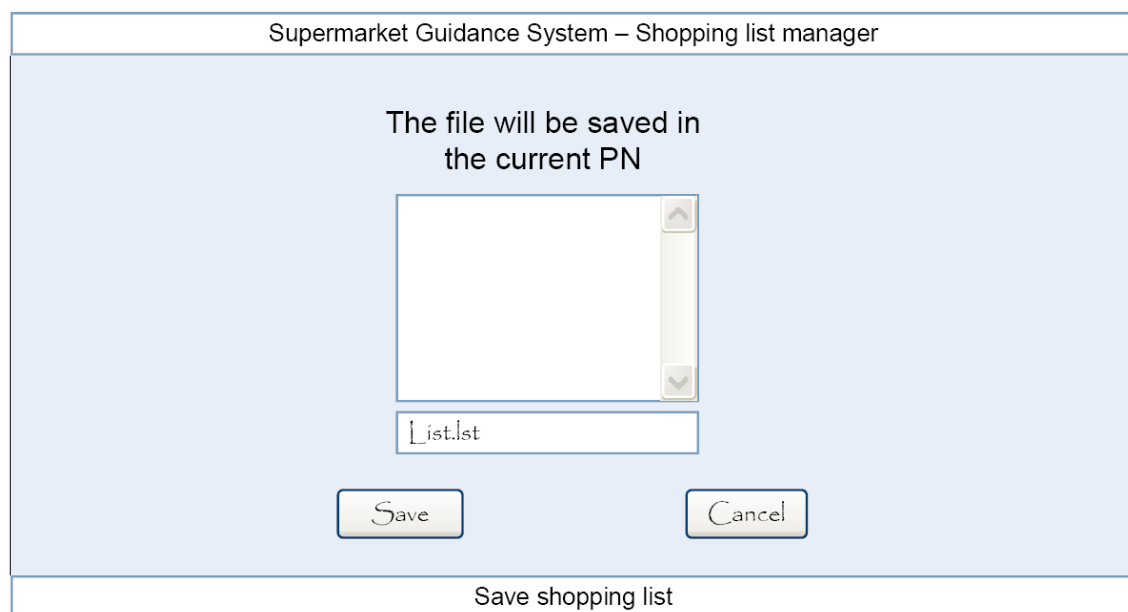


Figure 6.16: Save the list screen

Follow the instructions in order to pick up every item of the shopping list

The user is now in the supermarket. He has to click on the button "Yes" to launch the application.

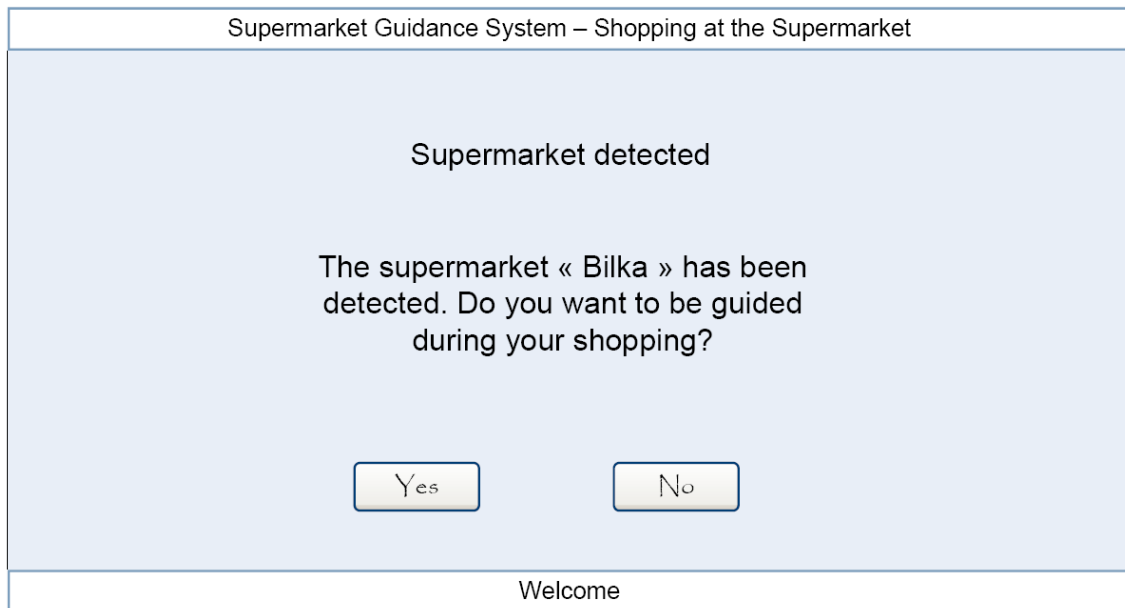


Figure 6.17: Detection of the system

Then, he has to follow the arrows in the supermarket to go the first area displayed (vegetables area).

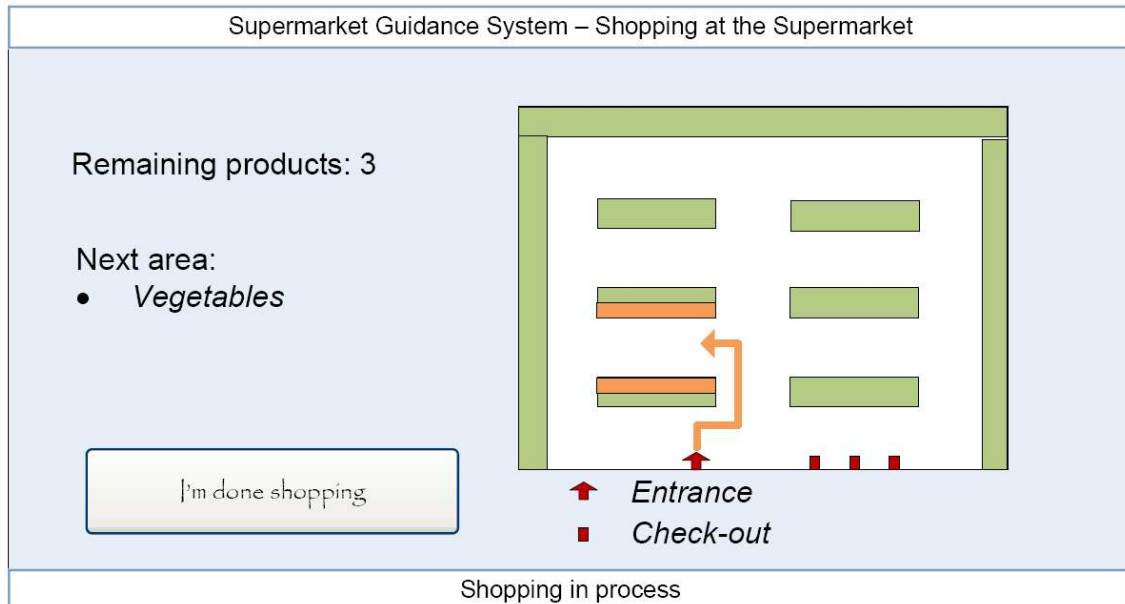


Figure 6.18: Go to a specific area

When he will arrive there, the screen will change automatically. He has to pick up the products whose name is written on the left part of the screen. They are located in the rows where the stars are drawn.

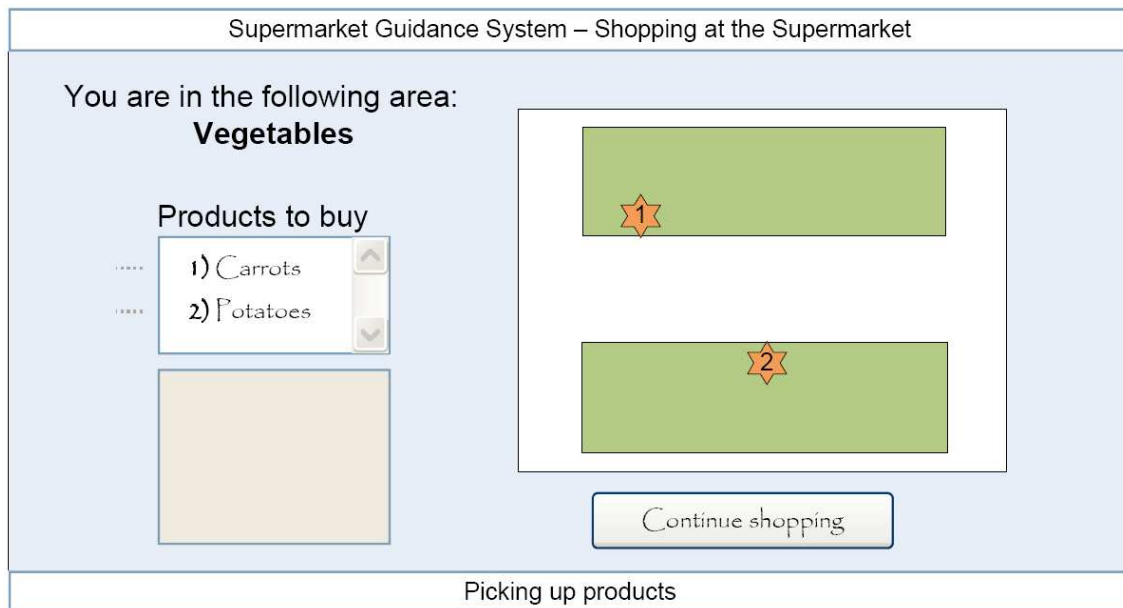


Figure 6.19: Pick up the products

To read the notes corresponding to a product, he needs to select the name of this product. The note will then be displayed in the field below.

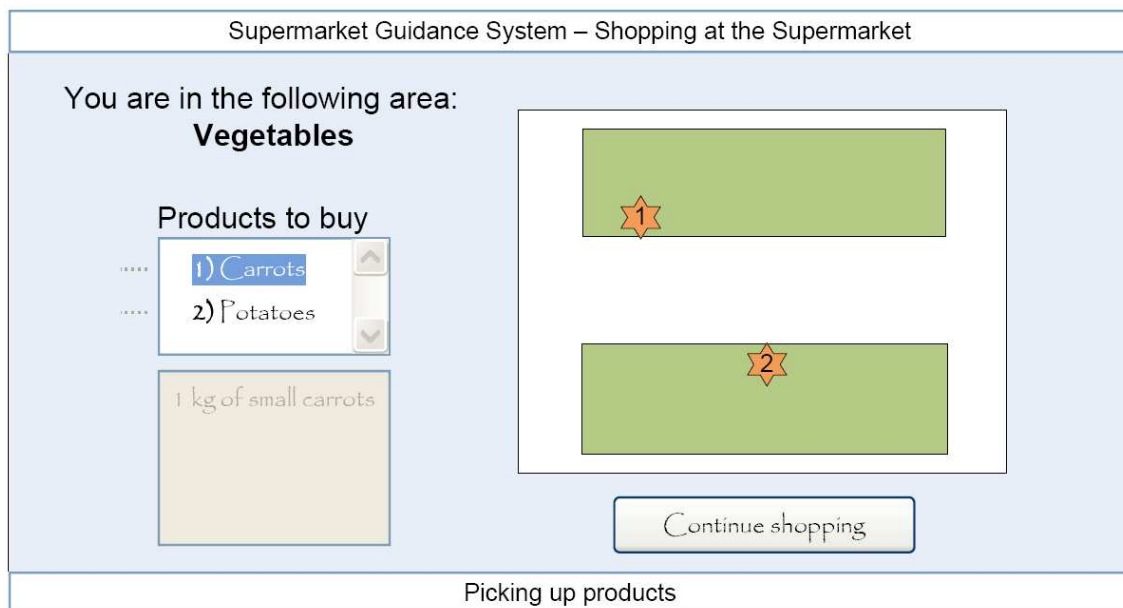


Figure 6.20: Check the notes of Carrots

After he picks up every product of the department, he presses the "Continue Shopping" button and he will be guided to the next department.

When there is not anymore remaining products in the list, he has to press the button "I'm done shopping".

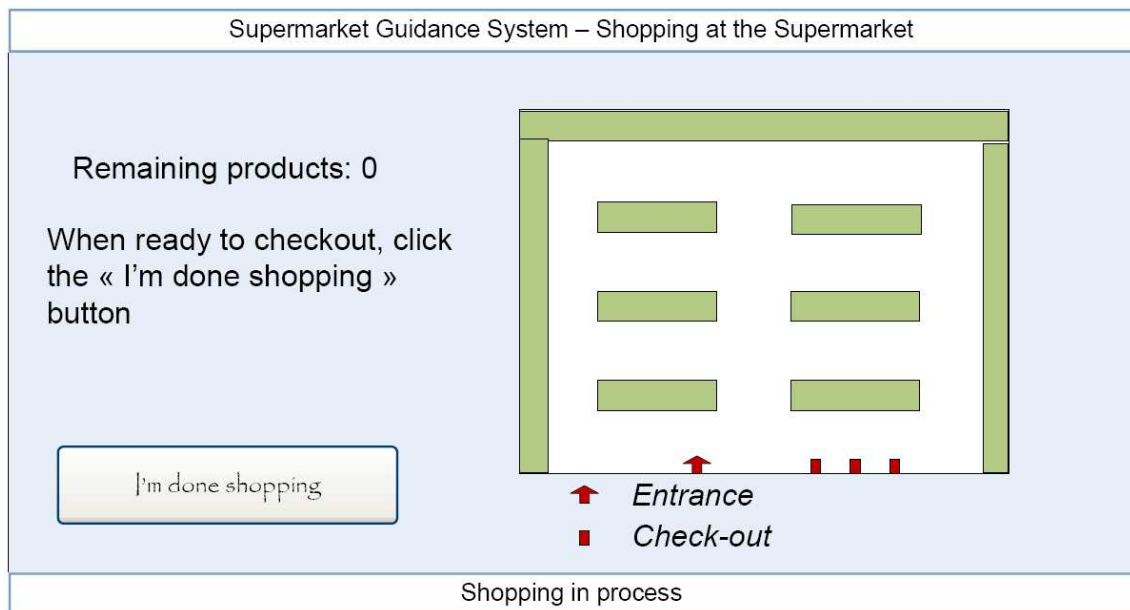


Figure 6.21: The list is empty

Check out

Now that he is going to checkout, he has to wait for his turn.

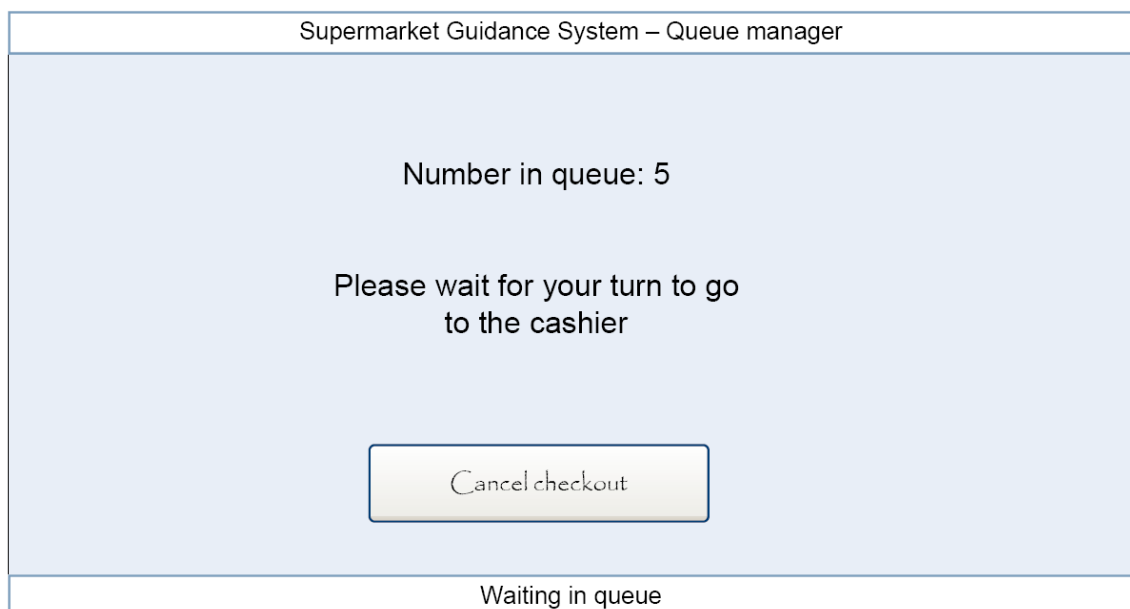


Figure 6.22: Number 5

When it is his turn, he has to go to the check out whose number is the one written on the final screen. He now can press the "Exit" button. He has completed the tests!

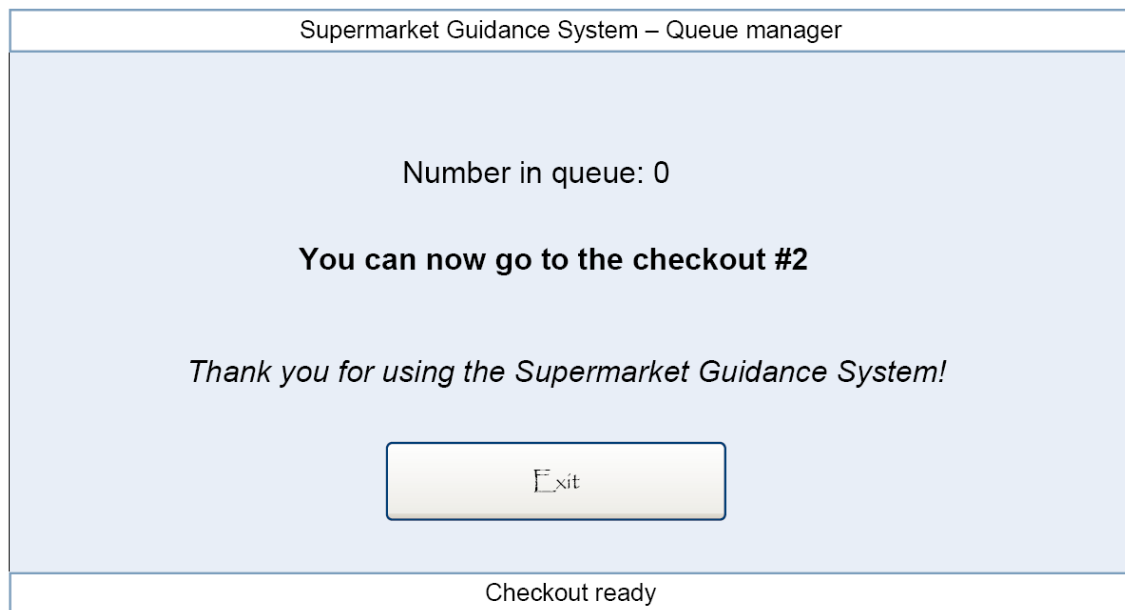


Figure 6.23: Check out

6.1.5 Results of the tests

Even if every tester was able to achieve their tasks list, several parts of the low-Fi prototype confused some testers.

Shopping List Manager

- Load a file

Firstly, the way of handling the loading of a list was not really understood by the testers. Some testers did not understand what to do with the button "Load a list" and a button "New list" was missing.

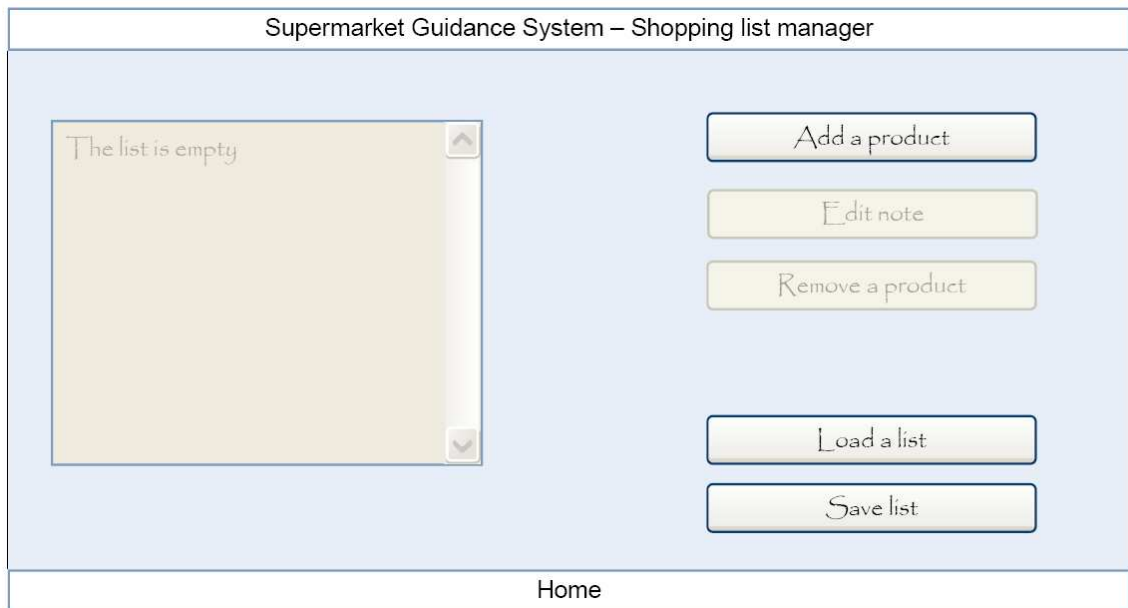


Figure 6.24: Home Page with an empty list

- Notes

The notes on the main page seemed to be specific for the whole list whereas it was only for the selected product.

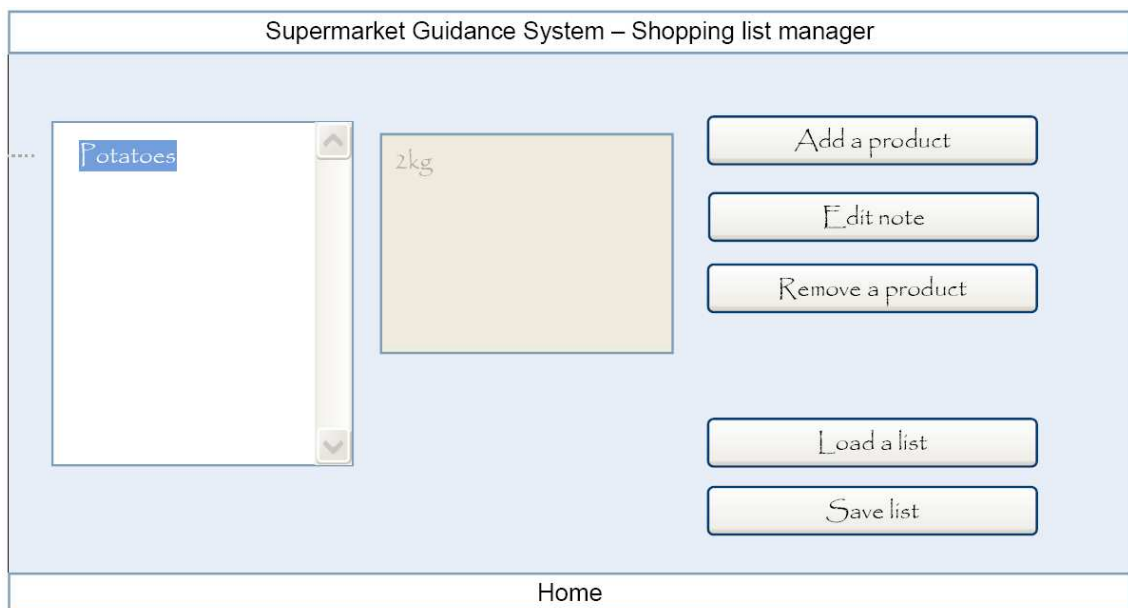


Figure 6.25: Home Page with the notes

- Background colors of the screens

One tester from the second group (the one uncomfortable with mobile application) was confused with the changes of background color regarding to the button he clicked.

Supermarket Guidance System

- "Continue shopping" button

This button seemed to be not clear. Some testers did not know whether they were supposed to take the products displayed on the current screen before clicking on this button.

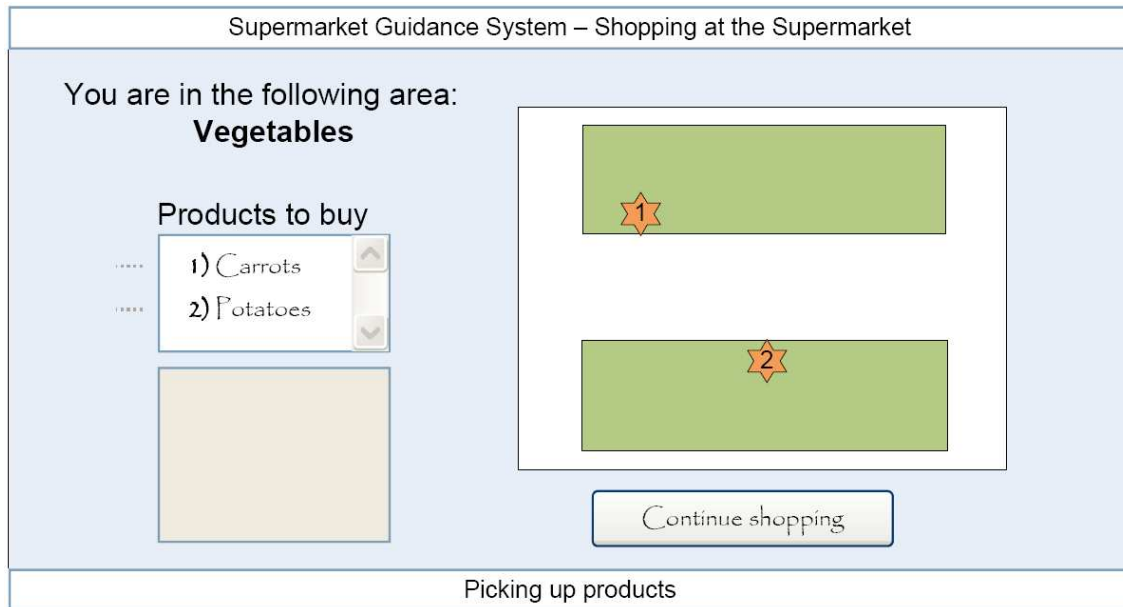


Figure 6.26: Guidance in the vegetables area

- Several items in the same department

Several testers did not understand that they were supposed to take all the items of the department at the same screen and pressed the "continue shopping" button before having taken everything.

- Notes

To access the notes, the testers were supposed to select an item on the screen. Only one found it. Some tried to click on the stars on the map to access them.

- "I'm done shopping" button

The label of this button is not clear because the testers did not know whether they still were able to stay in the supermarket to buy some other products or if they had to checkout after pressing it.

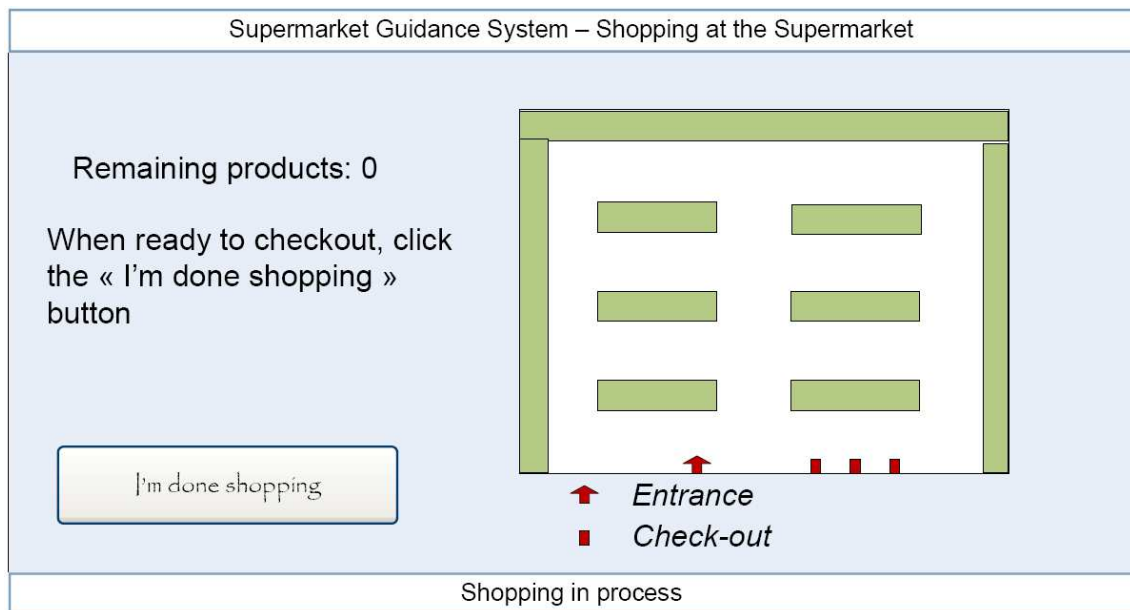


Figure 6.27: End of the guidance

6.1.6 Results of the survey

The statistics of the answers to the survey are available in the Appendix (cf. A.1.4. Answers of the survey).

These answers show that the application is generally perceived as easy to use and reliable. We noticed a problem with the question "The functioning of the program is logic and easy to remember".

About the shopping list, it seems that the testers could build it easily but 50% of them thinks that the way to build it is not the best one. Some testers made suggestions: an intuitive research and a list of products with checkboxes. Some others complained: "I have to click on many buttons to do my list... Is it convenient for a handheld device?"

About the guidance in the supermarket, the testers think that it is easily understandable and that it is the best way of providing directions. One tester suggested to add voice synthesis. Another would prefer an interactive map with rotating arrows regarding to his position in the supermarket. Moreover, one tester asked to have the products displayed with a bigger font: "The list should be displayed bigger: when you're stressed in the supermarket, it's even more annoying to have to focus on something which is supposed to help you."

Finally, some of the testers asked to have the whole shopping list displayed during shopping.

6.1.7 Decisions

These tests allowed us to point out several problems in the GUI we wanted initially to be implemented. Then, we took some decisions about the final interface.

Shopping List Manager

- Load a file

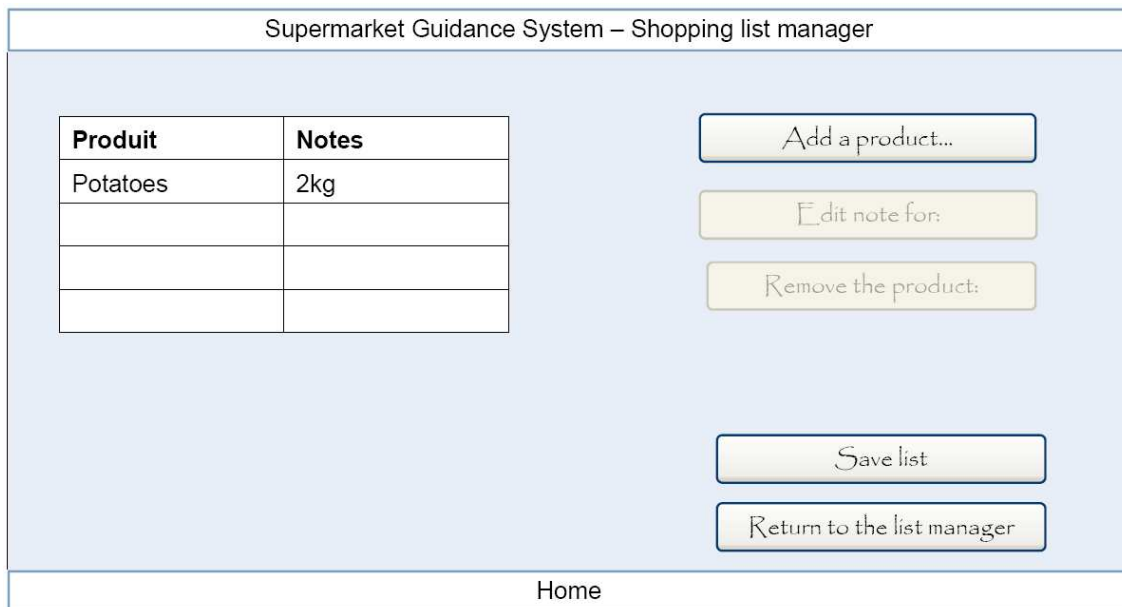
Firstly, about the loading of the shopping lists, we decided to add a first pop-up which will be displayed at the launching of the home page. This pop up will ask if the user wants to load a new list, the last list consulted or another list. Then, he will be able to launch this pop-up again by pressing a button "Return to the list manager" in the main screen.

- Shopping list

We decided to classify the products by category. The list of product for each category will be displayed by pressing the category button. This list will be composed of checkboxes buttons. In this way, it will be easier and quicker for the user to build his shopping list.

- Notes

We choose to display a table with two columns: products and notes. In this way, the user knows which notes concern which product.



Produit	Notes
Potatoes	2kg

Buttons:

- Add a product...
- Edit note for:
- Remove the product:
- Save list
- Return to the list manager

Home

Figure 6.28: Updated home Page with the table

- Background colors of the screens

Initially, we thought that changing the background color of the windows would help the user to know which screen is displayed. It seemed that it was not particularly helpful, but even confusing for one tester so we decided to keep the same color on every window.

Supermarket Guidance System

- "Continue shopping" button and Several items in the same department

To resolve these two problems, we decided to change the content of the screen with more explanation so that the user can understand better what he is supposed to do at that moment.

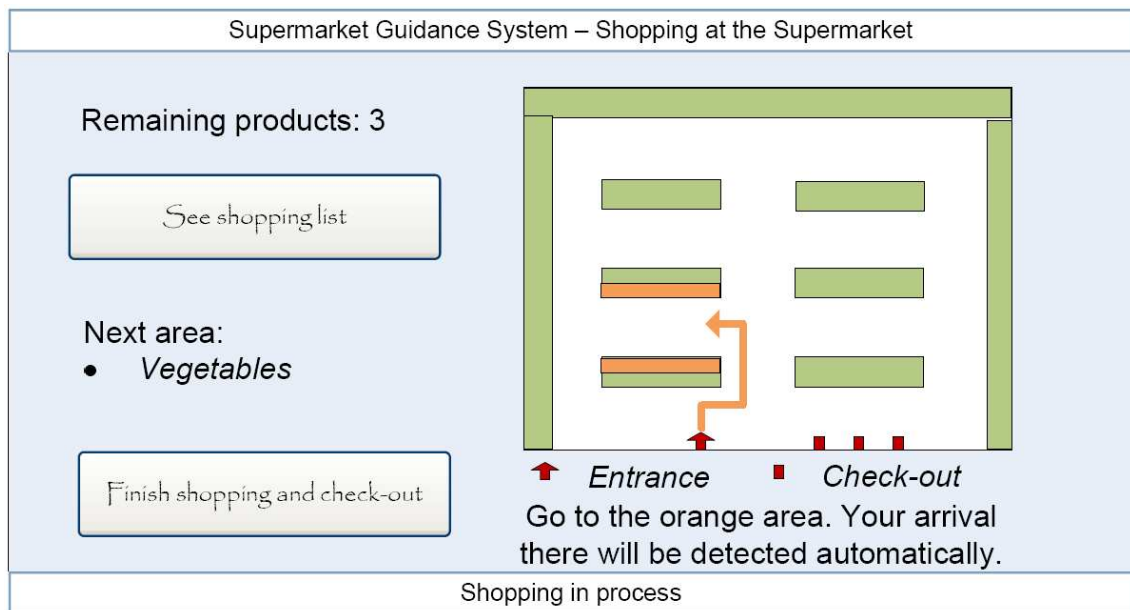


Figure 6.29: Updated Guidance Home Page

- Notes

The notes are now displayed directly on the screen. There is a table containing both products and notes for each department. It also allows us to display them in a bigger font. A button "See Shopping List" allows the user to display the whole shopping list.

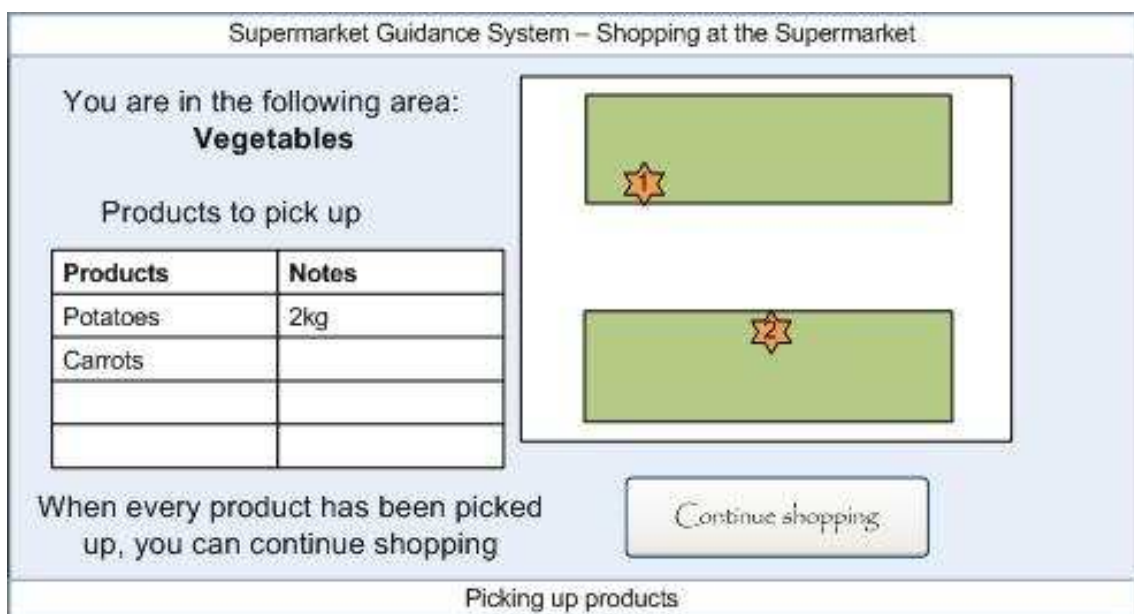


Figure 6.30: Updated Screen for a specific Area

Supermarket Guidance System – Shopping list manager	
Shopping list	
Products	Notes
Carrots	1kg of fresh carrots
Coca-Cola	
Potatoes	2kg
<div style="border: 1px solid black; padding: 10px; display: inline-block;">Close</div>	
Shopping list	

Figure 6.31: Updated Screen for a specific Area

- "I'm done shopping" button

We changed the label of this button with: Go to checkout.

6.1.8 New Low-Fidelity Prototype

The whole updated low-Fi prototype is shown in the Appendix part (cf. A.1.3. Updated Low-Fi Prototype).

6.1.9 Conclusion of the usability tests

These tests allowed us to get very interesting feedbacks from the testers. We modified several parts of our GUI and we got the answers we needed to know whether the way of building the shopping list and the way of guiding the customer in the supermarket were the best ones.

We also have some ideas about future improvement that would match with the users' needs such as vocal synthesis, completion of the list of products or displaying an interactive map.

6.2 Tests of the implemented system

After we finished to implement the whole system, we decided to run some tests in order to see if our system was easy to use, understandable by the users but also to point up some bugs if there are some.

You can find the pictures we took during these tests in the appendix (cf. A.2.3. Pictures of the final tests).

6.2.1 Organization of the tests

The testers

Seven testers took part in those final tests, they are in the same age range: between 21 and 24 years old. Three of them already participated in the usability tests we processed before (cf. Figure 6.32).

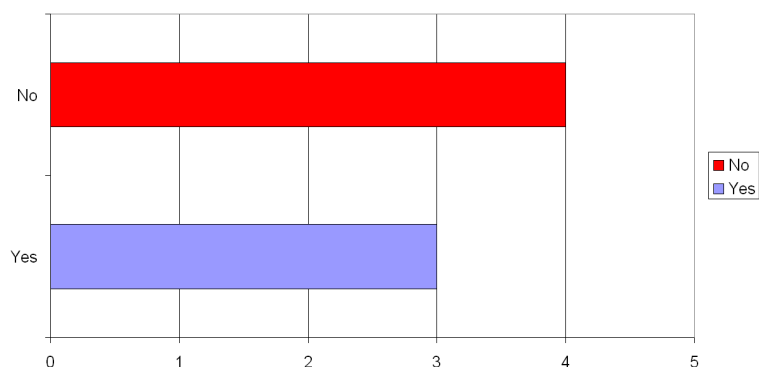


Figure 6.32: Did you participate in the usability tests of this system

So our user panel can be divided into two parts, the ones who already know the system thanks to the usability tests and the ones who do not know the system. Both are interesting because the first group can point up what was improved during the usability tests and in contrary, they can point up the modifications we implemented that do not fit with their expectations. We can also see with them if they are now able to understand some parts or to do some actions that remained to complicated during the first tests session. On the other hand, the second group is representative of people discovering the system and thanks to them, we will know how new users are reacting to our project.

To define more precisely the testers, we asked them some personal details in a global questionnaire. As you can see in the Figure 6.33, most of the testers are used to go shopping quite often or sometimes, only one goes rarely to the supermarket. The Figure 6.34 shows that most of the testers feel quite comfortable with mobile devices, only one does not feel comfortable with them.

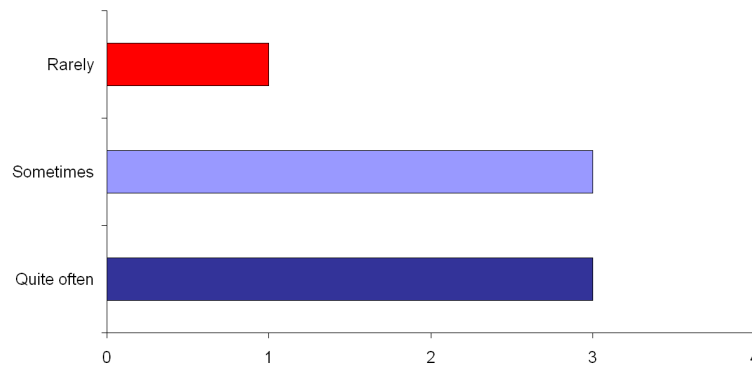


Figure 6.33: Do you often go shopping?

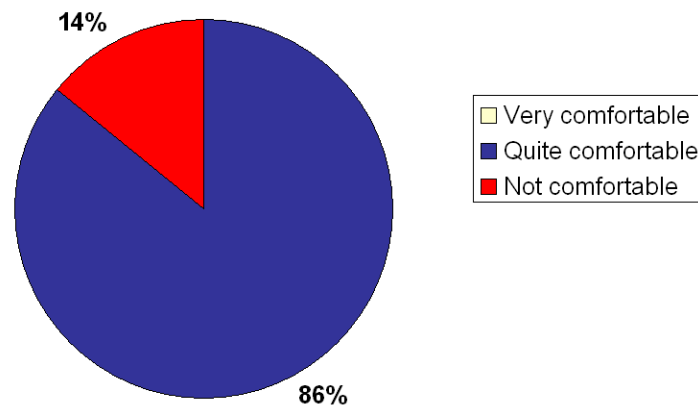


Figure 6.34: How comfortable are you with the use of mobile devices such as PDAs, Smartphones or Tablet PCs?

The tests rooms

For the tests of the first part of the system, the shopping list part, we will run these tests in a classical room. We do not need any specific installation or to be in a special room as the user can be anywhere to input his shopping list in his mobile device. So we only need the mobile devices with our system installed on.

For the tests of the part involving the supermarket by itself, the tests will be run in the library of our department. Of course, this is not a supermarket, but a library has almost the same characteristics as a supermarket with its organization of shelves.



Figure 6.35: The library

We changed some components to the library in order to turn it into a supermarket by adding pictures of the products the user needs to pick up at different places on the shelves. The Guidance System will guide the customers to these pictures to help him to find the right product.



Figure 6.36: The library transformed into a supermarket

The tests

At the very beginning of the tests, we will make a presentation for every tester (cf. A.2.1. Introduction text). The system will be explained but also the organization of the tests.

For this whole tests session, the two members of our project team will have predefined functions: one will be a facilitator, the person communicating with the tester and helping him when he has problems, the other one will be an observator, he will take notes during the whole tests.

The testers will all have a tasks list and they will get a mobile device with the application installed on.

For the shopping list input part, the testers will perform the tests one by one. In that way, the whole team can be totally focussed on this tester. Every tester cannot do the part at the same time as we need to be able to notice the problems and help the testers who run into problems. That means that we will do this task with one single tester every time so that the whole project team can totally be available for each tester.

About the part which will happen in the library, in reality, we should let the testers walk into the library with their mobile device and only be checking at the checkout if everything went well. But we would like to follow every tester to check how they are doing in the library, if sometimes, they have some hesitations or doubts and if they are running into some problems, we are supposed to guide them to succeed the test. Therefore, we will do the tests of the guidance one tester by one with always the two people of the project team with him, one as a facilitator and one as an observator. That also means that we have to divide this guidance part into two subparts: one during the actual guidance and one during the checking-out. In that way, for the queue manager part, there will be a real queue to handle (if we were doing the whole part in once with every single tester, the testers will be out of the queue before the next one arrives), we need that everybody arrives almost at the same time at the checkout.

When all testers have completed the guidance in the supermarket part, they will all ask one by one to go to the cashier. One person of the project team will be at the checkout, the other one will stay in the library to observe the behavior of the testers.

6.2.2 The tasks list

This is the tasks list we gave to our testers:

1st part of the test: Shopping List Manager

Context: You are in the main Kantine and want to build your shopping list.

Please execute the following task list.
Task list:

1. Add at least four products of at least three different categories in your shopping list
2. Save your shopping list

2nd part of the test: Shopping Guide

Context: You are going shopping. You enter the library.

This library represents a supermarket. The products are represented in the library with images printed on paper. To pick up a product, take one sheet of paper corresponding to the wanted product. In the final version, a location system is supposed to detect when you arrive at the right area and to update the screen by itself. This part is not working yet so instead, we added a temporary button "Next". When you arrive at the area you are asked to, click on this button to get another map of the area, more accurate.

Task list:

- Load the shopping list you previously created
- Follow the instructions in order to pick up every item of your shopping list.

- **ATTENTION:** When you are done shopping, you are asked if you want to go to the check-out. At this point, do NOT press the "go to checkout" button. Instead, please hold on and notify a member of the project team that you are done shopping. We want you to go to the checkout in the last part the test.

3rd part of the test: Queue handling

Context: You are now done shopping and you are going to the checkout. The project team has allowed you to do so.

- Check-out by following the instructions on the screen.
- Note: You do not actually pay for your products, they will just be checked by the cashier.

6.2.3 Results of the tests

In this section, we will describe the problems the testers encountered and the solutions we are planning to implement to resolve them.

Shopping List Manager

On the mobile application installed on the tablet PC, for every category, the first checkbox of the products list is always enlightened. It confused two testers who had the feeling that this first product was already selected. When the application runs on a computer, the checkbox are not enlightened. It seems that it is a specificity of the tablet PC, we do not really understand why there is this effect applying.

One tester was afraid to press the button "List Manager" because he thought that he would lose all his modifications for the current shopping list. Nevertheless, it is not really a problem because every time he selects an action in the List Manager which would imply a loss of the current list, a pop-up window is launched to ask for a confirmation, warning that the modifications of the current list will be lost.

The size of the button for one category, "Dairy products", is too small and it was not possible to read entirely the label. Indeed, to have a homogeneous interface, we decided to fix it at a predefined size. We will make it bigger to be able to see every label correctly.

Two testers complained because they were not able to see the shopping list just after having added some products of a category. Instead, the screen displayed is only the screen with all the categories available. To be able to see the shopping list, they need to press another button to come back to the main screen. Maybe we should divide the category screen into two parts, a left part with the shopping list displayed and a right part with the buttons of the category.

Finally, three testers complained about the handling of the shopping list. They did not like the file explorer, which is too complicated, too big with too many buttons on the screen. It was described by these testers as a not user-friendly screen. This file explorer is the one created by default by PyGtk. As anyway, the user is not allowed to change the directory, it seems not necessary to have such a complicated page. A single text field should be used to save a shopping list. And a more traditional list of available shopping list should be displayed for loading a file.

Guidance in the Supermarket

Every tester complained about the time needed for the maps to be displayed on the screen. Probably a solution to that problem could be to save directly on the tablet PC a map of the supermarket and only to send the modified elements when they are needed for the display. One of them suggested to add a

progress bar to allow to understand that something is actually going on.

The testers managed to interact with the mobile device while holding the papers representing the products but we noticed that it was sometimes a bit complicated for some of them (cf. Figure 6.37).



Figure 6.37: A user having a hard time holding the "products" and the mobile device at the same time

One tester did not understand what the detailed map (the map showing the locations of the products in a specific area) was representing. We think that the fact the user has to ask for this map by pressing the button "Next" is part of the problem because if it was displayed directly by the application when he arrives at the right area, it becomes obvious that it is concerning this area. Moreover the label of the button (Next) is maybe not the best one, we could have called it "Detailed map" instead for example.

Another tester asked for having the detailed map only displayed when he asks for it. It certainly is due to the time needed to display it. The user was already seeing the products on the shelves that he wanted to take but he had to wait for the display of the screen, which could be quite annoying when it takes too much time whereas it is just insignificant when it is quickly done. As the initial plans were to display it automatically when the user enters the area thanks to a location system, we anyway do not want to make this map optional.

Two testers did not know how to use the detailed map. They did not know in which directions they should read it. We probably should add some directions next to the map.

Another tester did not understand how the colors were used of the detailed map. There is one color per product. The square on the map representing a product is colored the same way the label of the product is colored in the list next to the map. But as it was directly understood by every other testers, we choose not to modify anything about that.

Again about this detailed map, we noticed some bugs during these tests as some products were not located correctly on the map. We will work on finding the reason why.

On the main screen, a label "Go to the orange area" explains to the user that he is supposed to follow the path displayed to arrive at the area on the map which is colored in orange. One tester was confused by this label because he previously added oranges in his shopping list and he thought the first step was to pick up the oranges. To avoid that kind of problem, we decided to use another color such as red not to have any possible confusion with any product.

Queue Manager

We did not encounter any problem during the part of the queue management. Each tester understood perfectly what he was supposed to do.

6.2.4 Results of the survey

Ratings

Some of the questions of the survey were the same than the ones from the survey of the usability tests. The ratings we obtained from these questions sometimes showed very important differences.

The answers for the questions "Is the functioning of the program logic and easy to remember?" and "Is it easy to build the shopping list?", the answers show clearly that the modifications we added since the usability tests correspond to the expectations of the users.

On the other hand, for the question "Is the application enjoyable to use?", the answers are not as good as they used to be for the usability tests. It is due to the time the testers had to wait during each display of the maps.

For the question "Is it better to shop with this system than to shop on my own as I usually do?", every single tester disagreed. There are two reasons responsible for these answers, firstly, because the testers think that they would be quicker without it, as the display of the maps is too slow, but also because the tests happened in a small area, which means that it is easy for them to find the products they wanted without any help. We think that probably in a bigger supermarket, the system would seem to be more useful.

The last point is about how the user feels about shopping and holding/interacting with the tablet PC at the same time. Most of them think that this is not easy. The problem is that they cannot picture themselves holding a shopping basket and using the tablet PC at the same time. This is a problem for the current state of the system, but when the scanning of barcodes will be implemented, the customer will have two free hands to use the tablet PC.

All the statistic results of the survey are available in the appendix (cf. A.2.2. Answers to the survey).

General remarks

Every tester complained about the time to load the maps. Some of them even said that they would prefer shopping without the application because of that delays problem. It means that we really should focus on improving it. This part is more than just an optimization, it is for the user a reason for using or not

using our system.

Two testers wrote remarks about the fact that it was complicated to interact with the tablet PC at the same time they are holding a basket. Maybe a tool to attach the tablet PC on a cart could be an interesting solution. But otherwise, as in a future implementation of the system should integrate the scan of barcodes, customers will not need a basket or a cart anymore, and so will not have this problem.

6.2.5 Conclusion of the tests of the implemented system

About the shopping list manager, it seems that the way of building the shopping list has improved and now fits with the users' expectations. Now, the tasks we need to complete are firstly to change the screen about shopping lists handling (save a list and load a list) and secondly to display the content of the shopping list on the category screen.

For the guidance part, we know that our priority is to find a solution to optimize the delays for getting and displaying the maps. We also need to work on the few bugs we encountered during the tests such as the location of the products which was sometimes not accurate enough. Otherwise, the instructions seems to be given properly as they were perfectly understood by the testers.

About the detailed map, with the results and comments we got, we could have the feeling that this map is really problematic and maybe useless. Nevertheless, we think that it is due to the test environment. The library room was quite small whereas our system is built to be used in big supermarkets, where you can get lost. That made this detailed map a bit useless in the library as the testers could see directly by themselves where the products they are looking for are. But for sure, in the real environment, these maps are definitely necessary.

These tests were very useful firstly because we will have some time between the deadline of the report and the project presentation to work on the technical part to resolve some problems but also for the next people who will work on our project to implement the parts that we did not have time to build. They will have a better idea of how the users feel about our system.

7

Conclusion and perspective

7.1 Conclusion

At an early stage of the project, we had decided of four main objectives: to build a system that will really help people to shop quicker and more easily, to implement an ergonomic user-centric interface to this system, to learn more about mobile development and to get skills in integrating Magnet beyond into a heterogeneous system.

According to the test we ran during the project, we can hardly say that our system makes the shopping experience quicker. It is due to a few details that we probably did not focus enough on as we considered they were secondary. One of these details is the long time it takes for the client to receive data for the server, an issue we only realized during the tests of the final prototype and so, that we did not have enough time to fix. Another one would be the lack of convenience of holding a mobile device and a stylus while shopping. So the whole system we developed in general would probably be strong and efficient enough to complete this objective, but because of such details, we cannot call it a full success.

On the other hand, the implementation of an ergonomic user-centric interface can be considered as a success. A lot of time and energy has been spent in that interface, its development has known milestones such as the feedback of the usability test, which was of great help.

Also, we definitely learned more about mobile application as we learned a whole new mobile language which was Python. We had to adapt to specific constraints of the mobile world, such as a limited screen size. We also had to learn how to interface a Python client with a Java server, successfully using protocols that we did not know before.

As for the integration of Magnet beyond into a heterogeneous system, this is unfortunately something we could not do. What we did not see at the beginning is that Magnet Beyond project is still at an early stage of development and there is only few documentation and no existing API yet, which makes it very hard to actually use it. We never succeeded. On the other hand, the Magnet spirit is still totally present in our project and is exactly the type of services that Magnet may offer in the future.

7.2 Possible Improvement

The next step that could be done would obviously be to implement the Magnet technology in this project. This would allow further improvements such as saving the shopping list within the PN so it can be accessed and modified by any user within the PN, possibly the PN-F if such is the wish of the main user.

Also, the list of products could also be saved within the PN-F, which would make updates very easy as there would be only one file to update and it would spread within the PN.

Besides the whole Magnet integration, many things could be done. First of all, holding a mobile device in one hand and a stylus in the other one is definitely not an option while shopping. We probably can get rid of the stylus by increasing the size of the button, so the user can use his fingers. But this should just be taken to the next step, which is integrating the device to the shopping cart, in a similar way to the Concierge System.

Another possibility would be to just get rid of the shopping cart by totally changing the way people go shopping. Instead of picking up products and filling up a cart or a basket, the customer would wander in the shop with his handheld device and scan the barcodes of the products he wants to buy. The reference of each product would be sent to the server, which would charge an employee to fill in a cart. The customer would pay, then get his cart back. This is probably the way we will be shopping in a few years, except maybe for certain categories of products such as Vegetables and Fruits, as the customer would probably want to see exactly what he buys.

Finally, as more and more home devices become "intelligent" and implement connectivity capabilities, we could imagine that in the long run, people would be equipped with intelligent fridge. Those fridges could be able to detect the lack of a certain product, and if it is part of the home PN, to suggest the addition of that product to the family's shopping list.

7.3 Perspective

This project must be considered as a proof-concept that allows to imagine what could happen in the near future, when Magnet technology would be part of the daily life.

Such a system, as a fully integrated Magnet service, will change the way customers do shopping in supermarkets: the task will become quicker and less exhausting, which will lead to a tiny improvement of people's quality of life!

Also, this system could have a massive side-effect. Indeed, a supermarket that implements this system will be likely to require less employees than one that does not, as the number of cashiers is lower for example. So this reduces the costs and on the long term, it should have an impact on the final price of the products, which will become slightly cheaper.

Finally, this project presents one idea of a Magnet service, but there are endless possibilities and it will be very exciting to see which direction the development of Personal Networking services will take in the next decade.

Part III

APPENDIX



Extra Documents

A.1 Usability tests

A.1.1 Presentation of Usability tests

Introduction text

You are part of a testing panel for the design of a mobile application. The goal of this application is to help the user shopping at a supermarket.

There are two different parts to the system and you will be asked to test both.

The first part is the Shopping List Manager, which is used to build a shopping list before going shopping.

The second part is the Shopping Guide, which is used to guide the user when he is in the actual supermarket.

You will be given a stylus that you will use through the interface. However, the second part of the test implies actions such as walking in the supermarket: you will have to voice these actions. For example, you will have to say "I walk to the first row and turn left", or "I walk there" and point an area of the map displayed on the screen.

You will be provided a tasks list for both parts of the system. You will have to follow the tasks list closely.

We will take notes during the test. Our goal is to see which difficulties you encounter, how long it takes you to achieve the different tasks, etc. However, feel free to express your feeling anytime: we want to get as much feedback as possible. For the same purpose, you will also be asked to answer a survey at the end of the test.

About this survey, we ask you to express your own feeling. You should try to use the whole scale instead of using only the two answers "I'm satisfied" and "I'm not satisfied" through the whole survey.

If you still have questions, you can ask us now. When we are done with the questions, the test will start.

Tasks list

- **1st part of the test: Shopping List Manager**

Context: You are at home and want to build your shopping list.

Please execute the following task list.

Task list:

- Add Potatoes to the shopping list and specify you want to buy two kilos
- Add Carrots to the shopping list
- Specify that you want 1 kilo of small carrots
- Save the shopping list

- **2nd part of the test: Shopping Guide**

Context: You are going shopping. You enter Bilka supermarket.

Task list:

- Follow the instructions in order to pick up every item of the shopping list
- Check out

Survey

1. Personal details

- Do you often go shopping?
 - Yes, quite often
 - Sometimes
 - Rarely
- How comfortable are you with the use of mobile devices such as PDAs, Smartphones or Tablet PCs
 - Very comfortable, I often use handheld devices and I'm often use their advanced features
 - Quite comfortable, I usually can find out how to do what i want to do without too much trouble
 - I'm not really comfortable with such devices and I often struggle to do what I want

2. Ratings

- The application is easy to use:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The general design of the program is good:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- It is easy to browse through the application without getting lost:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The functioning of the program is logic and easy to remember:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The application is enjoyable to use:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The application seems to be reliable:

- Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- It's easy to build the shopping list:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The way of building the shopping list is the best one:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree

Suggestions?:

- It's easy to understand the directions inside the supermarket:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- A map is a good way of providing directions:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree

Suggestions?:

- The way of providing directions in the supermarket is the best one:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree

Suggestions?:

- It is annoying not to be able to see the whole shopping list and to modify it while shopping:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree

3. Comments

Do you have any comments?

A.1.2 First Low-Fi Prototype

Shopping List Manager

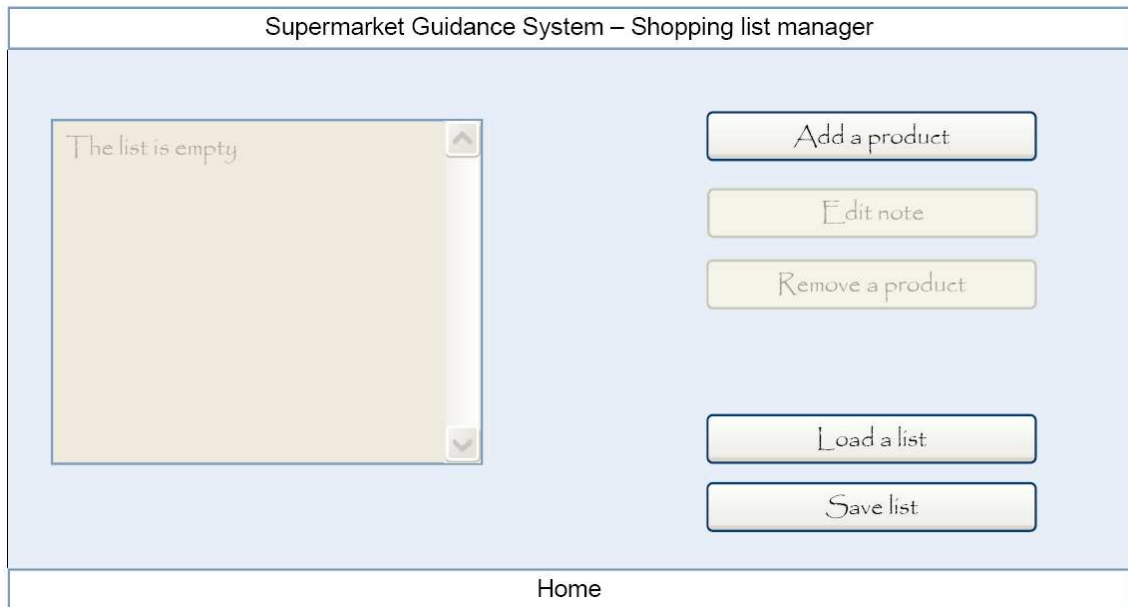


Figure A.1: Shopping List Manager - Home Page 1



Figure A.2: Shopping List Manager - Home Page 2

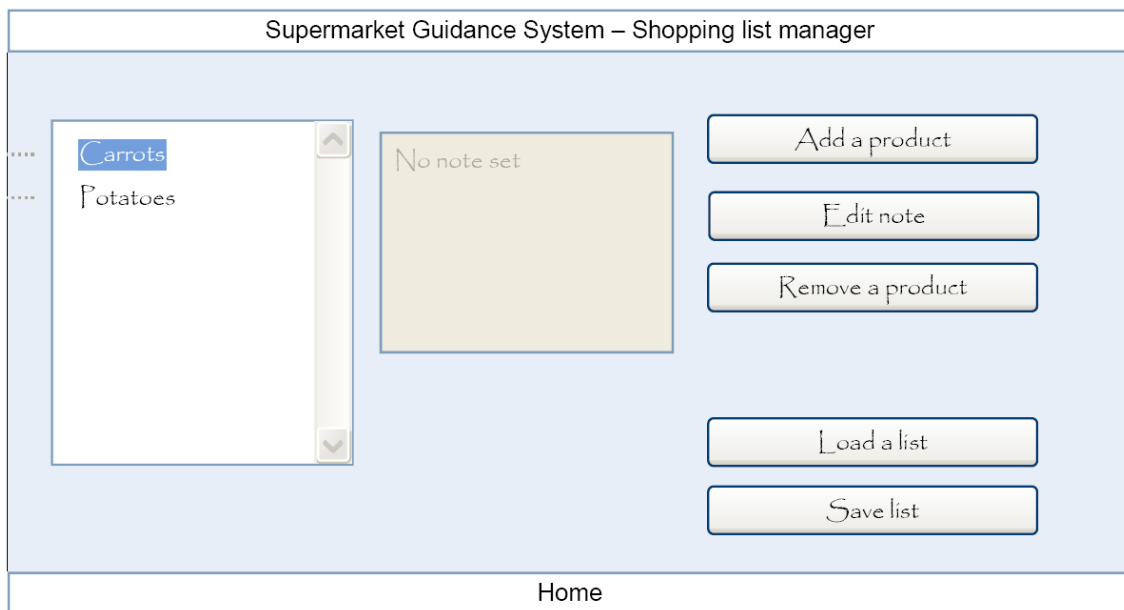


Figure A.3: Shopping List Manager - Home Page 3



Figure A.4: Shopping List Manager - Home Page 4

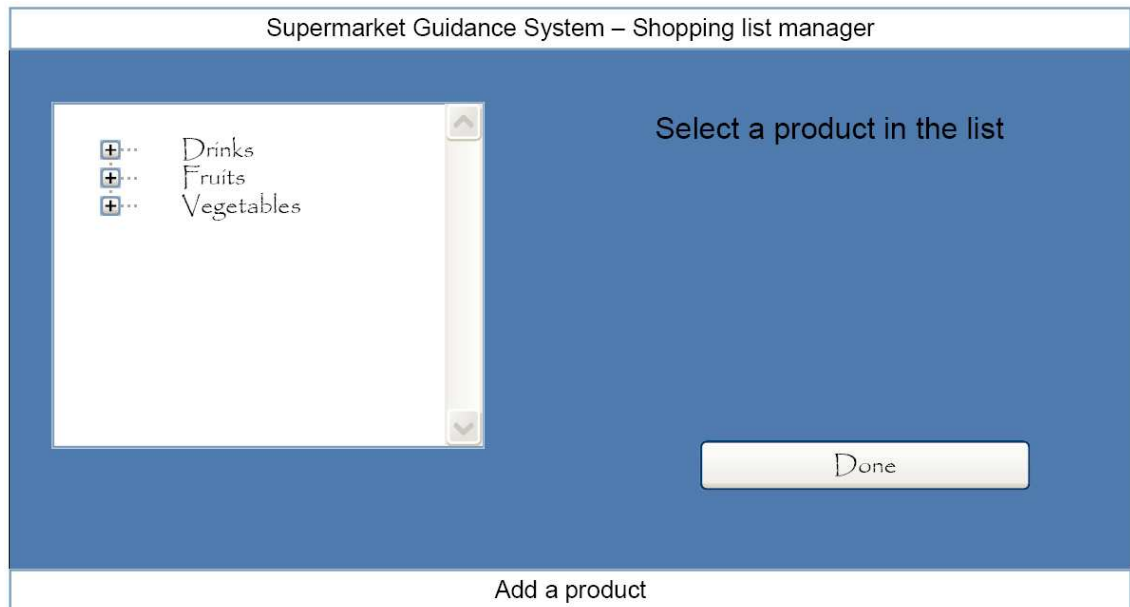


Figure A.5: Shopping List Manager - Add a Product 1

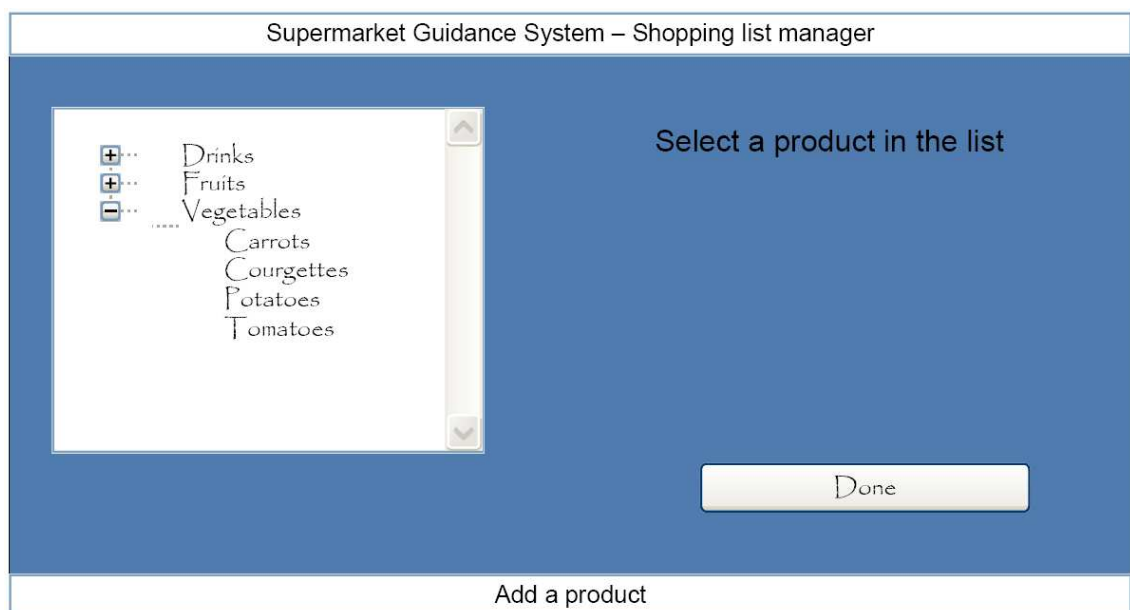


Figure A.6: Shopping List Manager - Add a Product 2

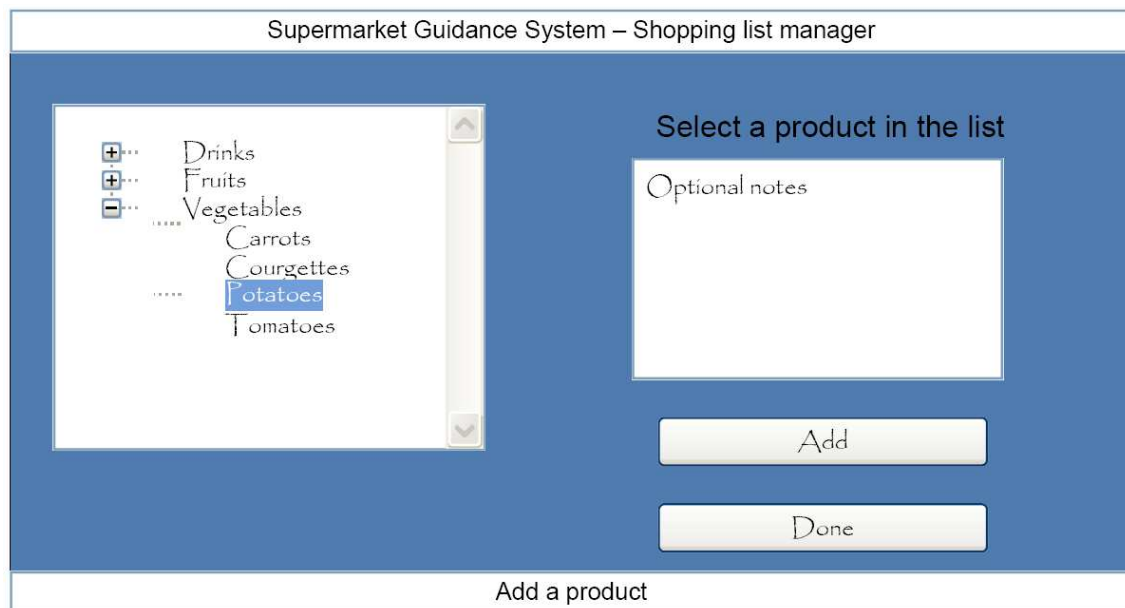


Figure A.7: Shopping List Manager - Add a Product 3

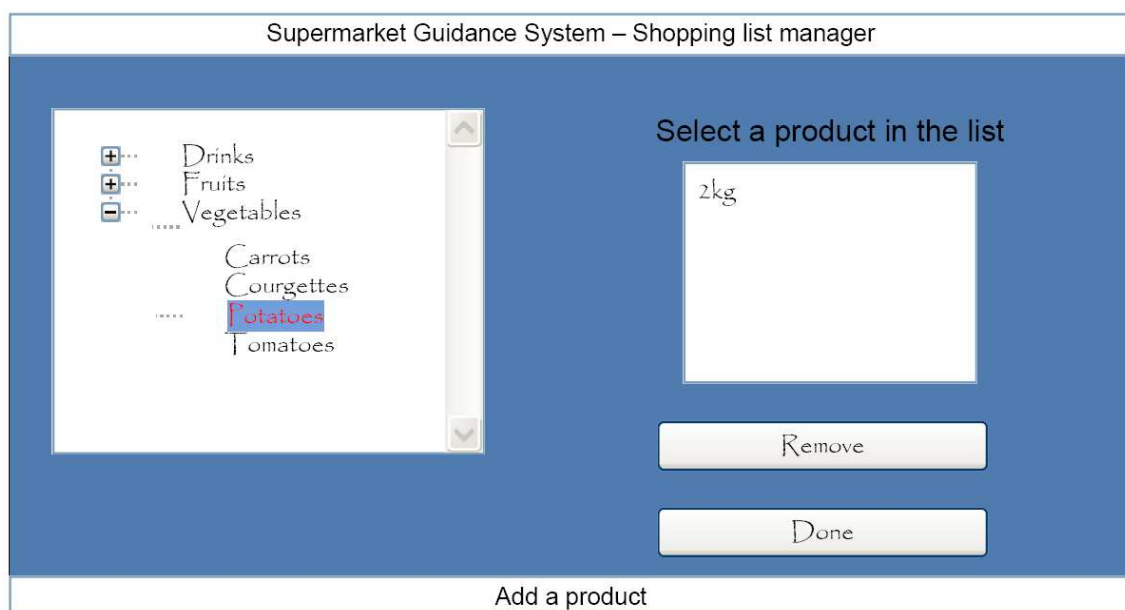


Figure A.8: Shopping List Manager - Add a Product 4

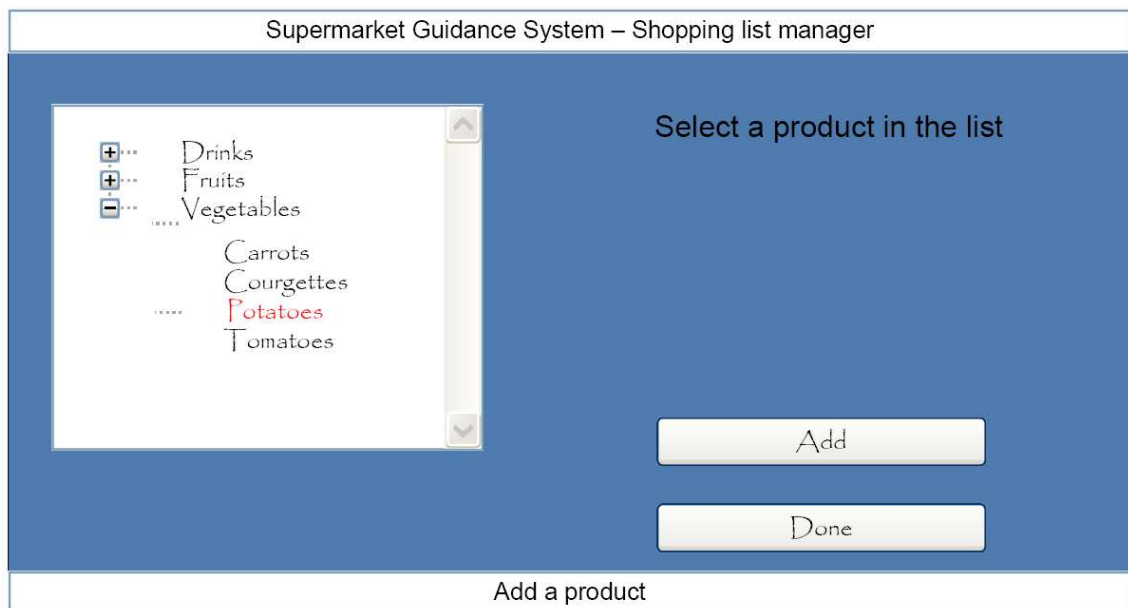


Figure A.9: Shopping List Manager - Add a Product 5

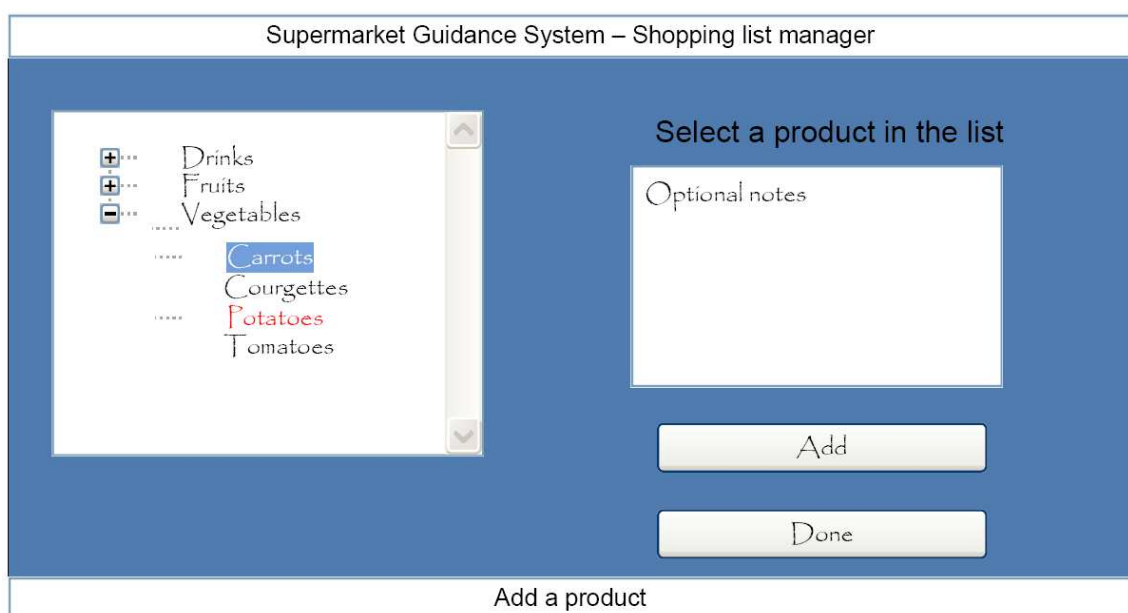


Figure A.10: Shopping List Manager - Add a Product 6

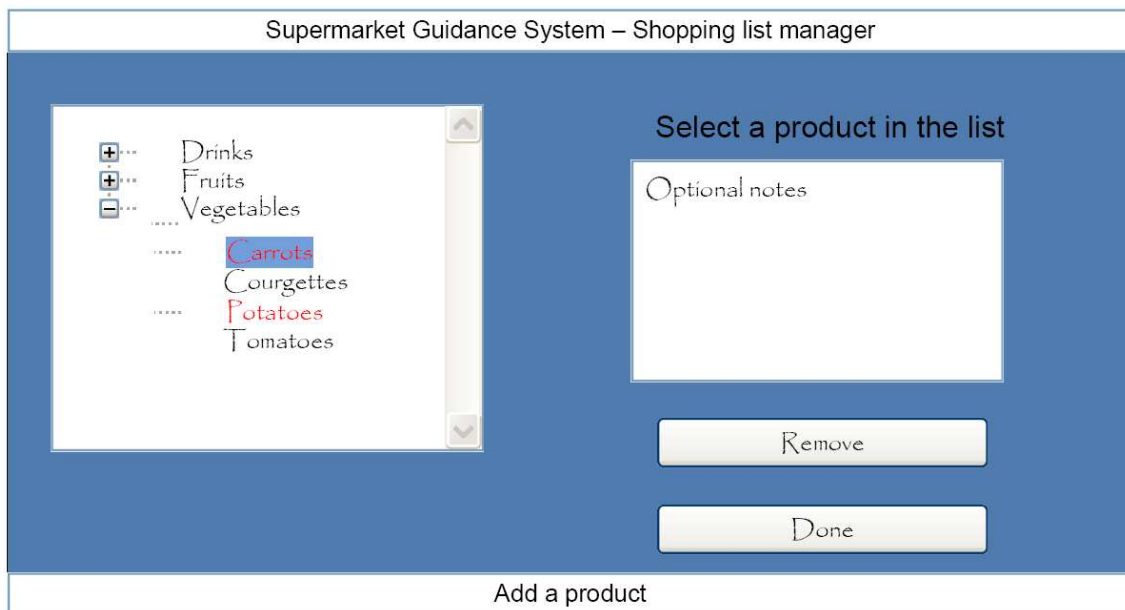


Figure A.11: Shopping List Manager - Add a Product 7

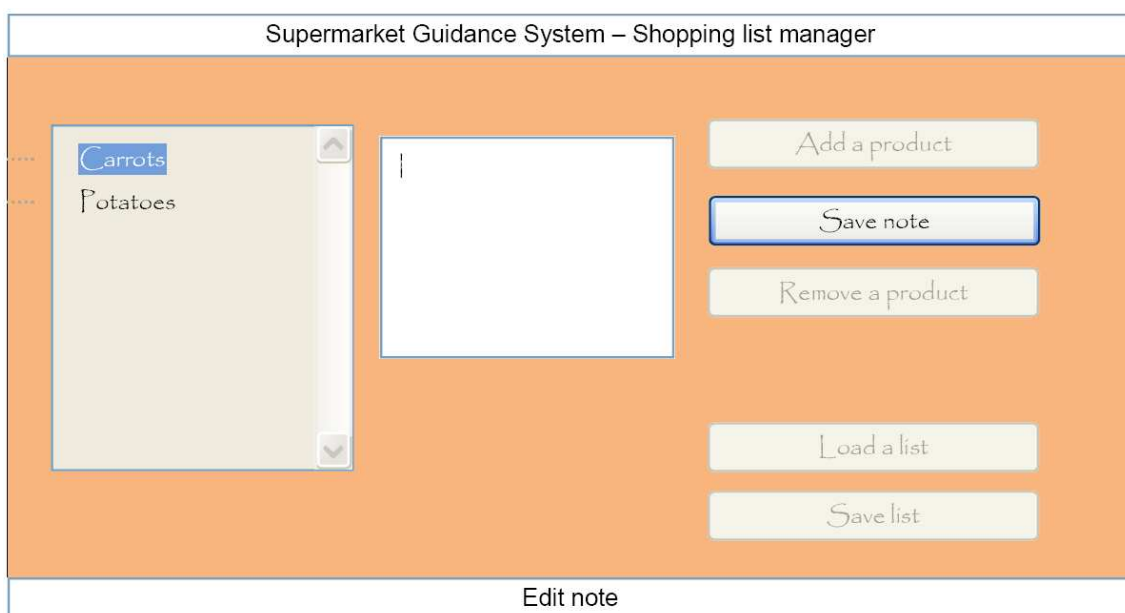
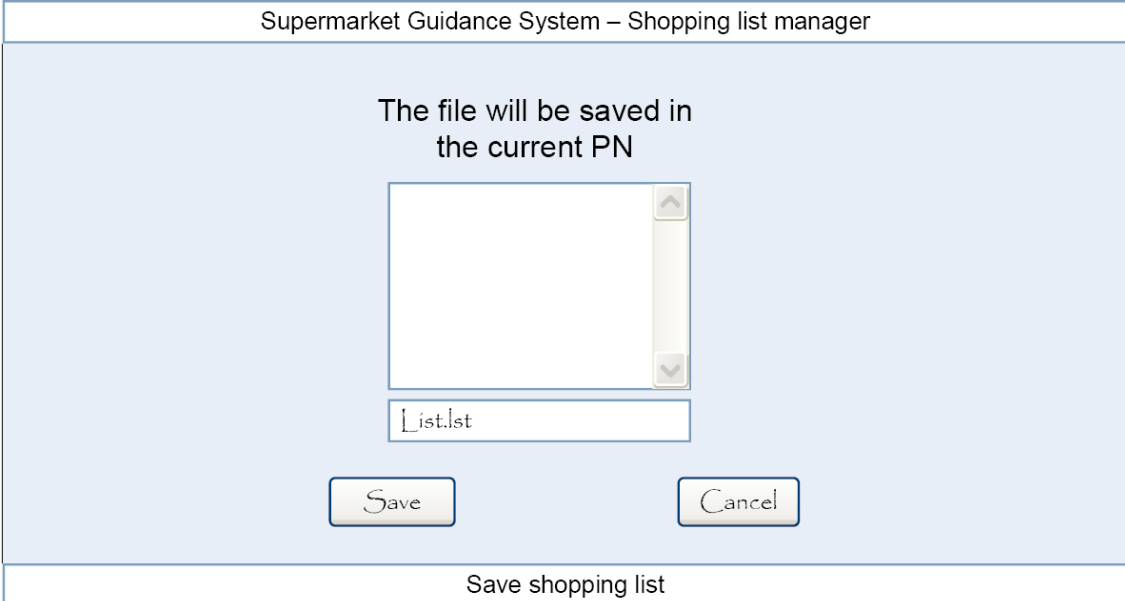


Figure A.12: Shopping List Manager - Edit Note



Supermarket Guidance System – Shopping list manager

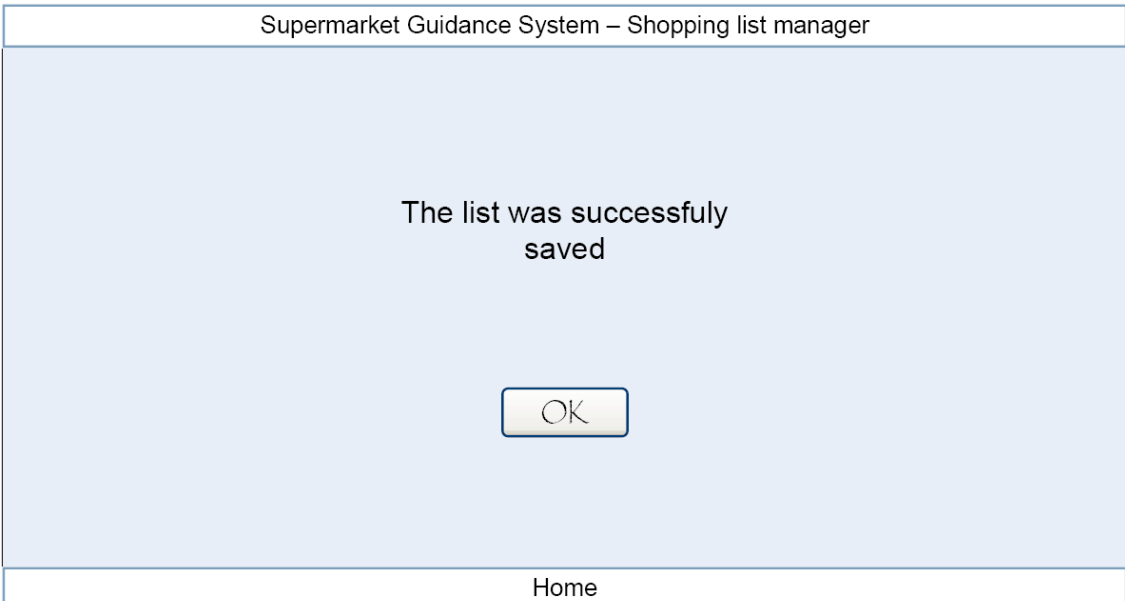
The file will be saved in
the current PN

List.lst

Save Cancel

Save shopping list

Figure A.13: Shopping List Manager - Save List 1



Supermarket Guidance System – Shopping list manager

The list was successfully
saved

OK

Home

Figure A.14: Shopping List Manager - Save List 2

Shopping at the Supermarket

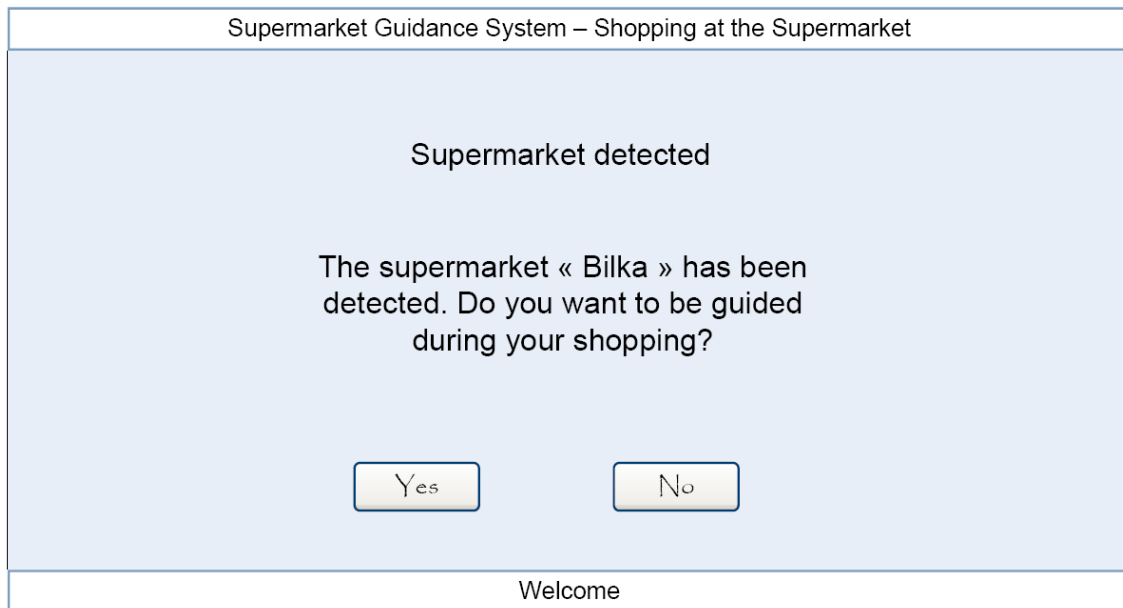


Figure A.15: Shopping at the Supermarket - Detection of the system

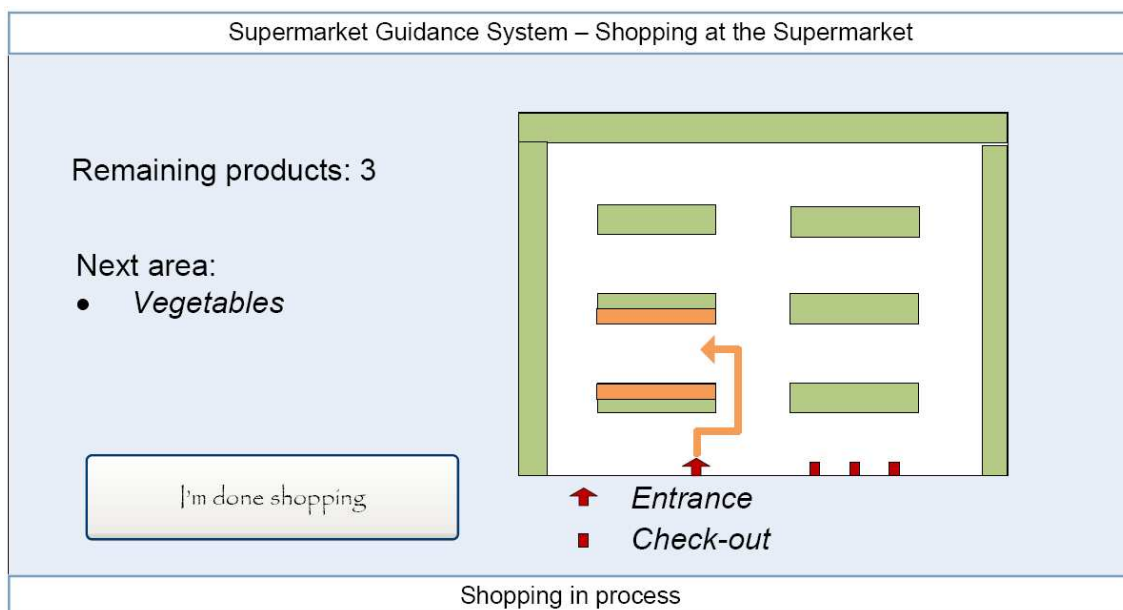


Figure A.16: Shopping at the Supermarket - Next Area Vegetables

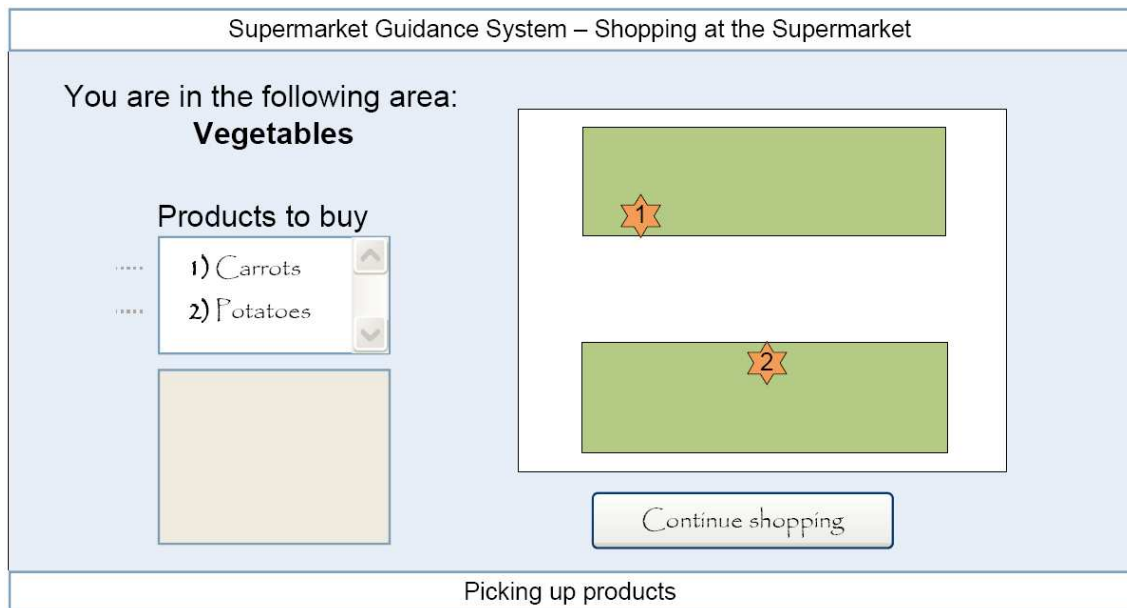


Figure A.17: Shopping at the Supermarket - Vegetables Area

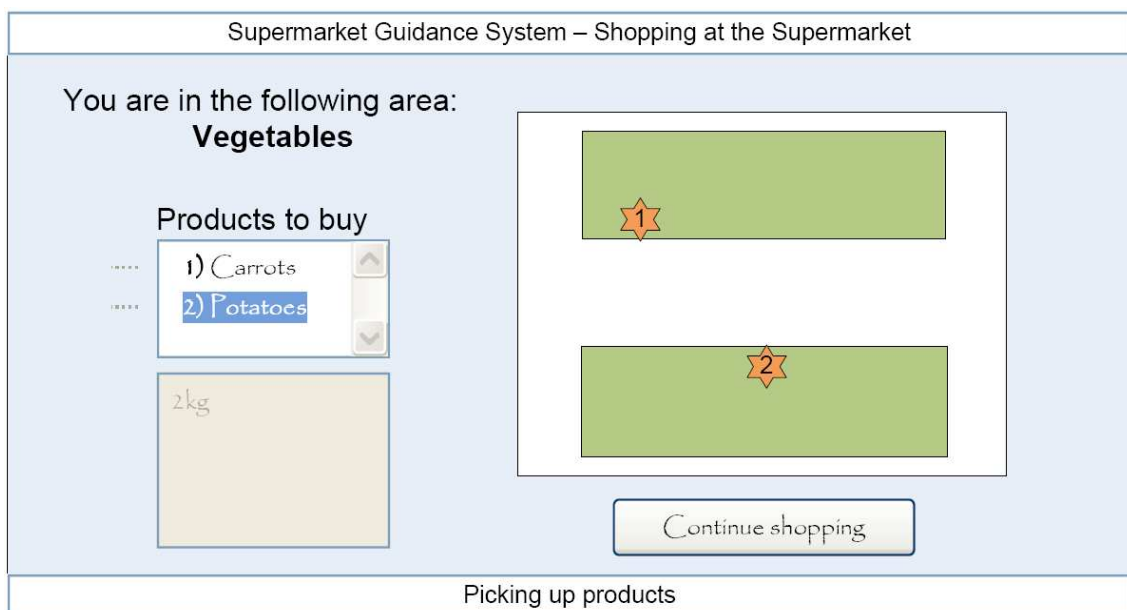


Figure A.18: Shopping at the Supermarket - Vegetables Area - Potatoes selected

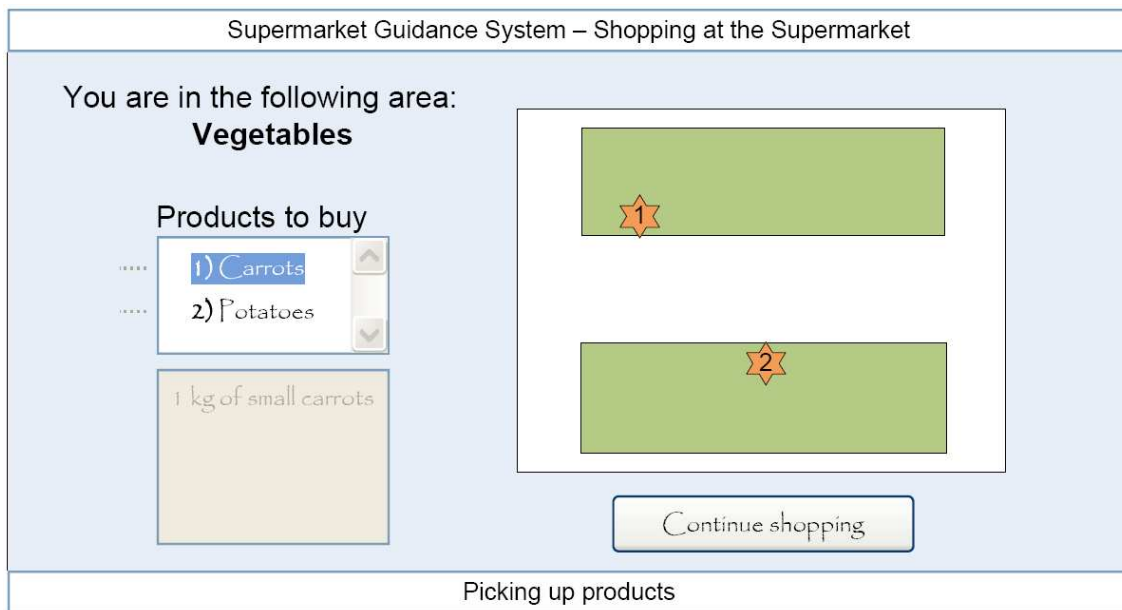


Figure A.19: Shopping at the Supermarket - Vegetables Area - Carrots selected

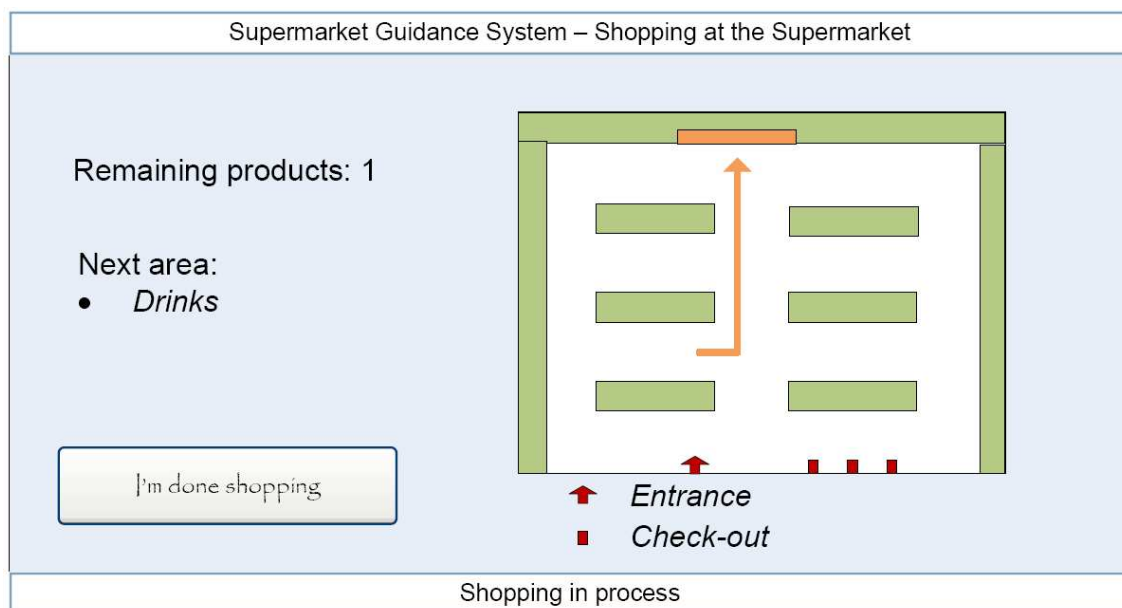


Figure A.20: Shopping at the Supermarket - Next Area Drinks

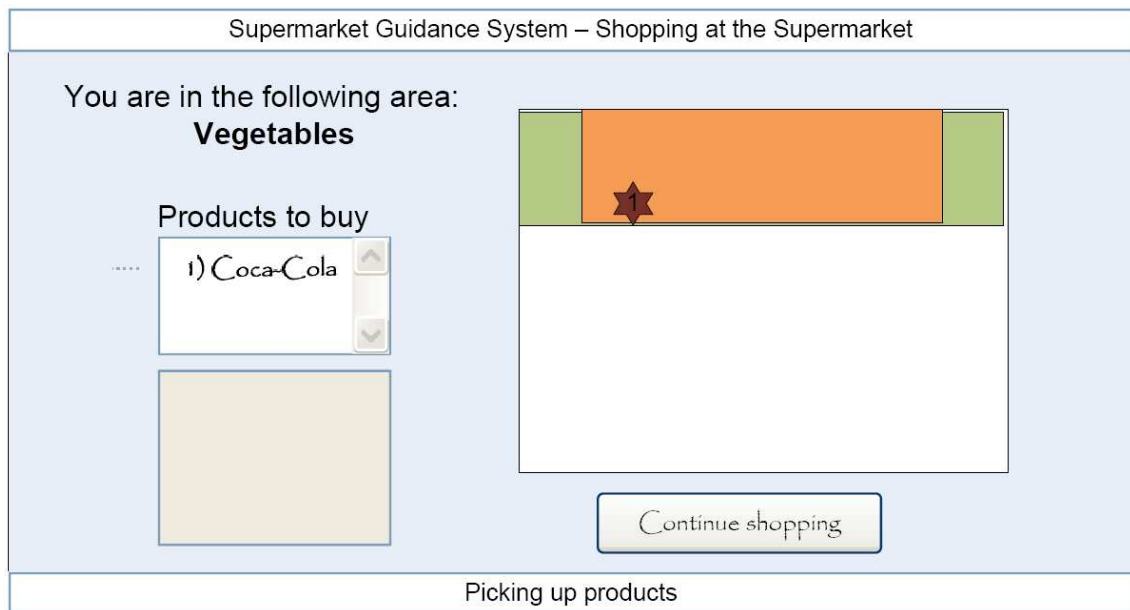


Figure A.21: Shopping at the Supermarket - Drinks Area

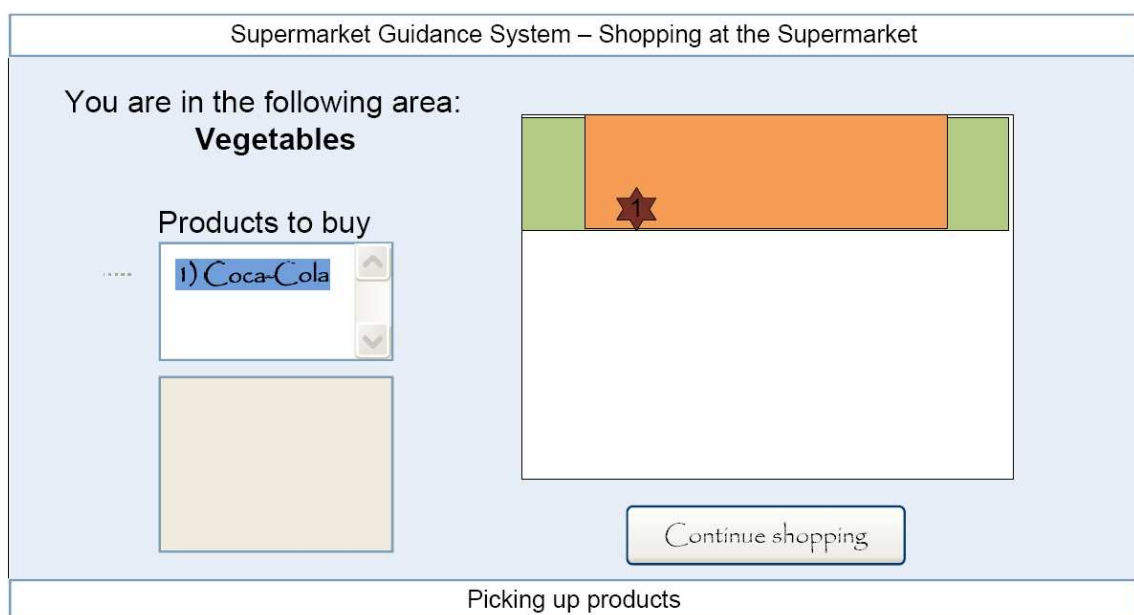


Figure A.22: Shopping at the Supermarket - Drinks Area - Coke selected

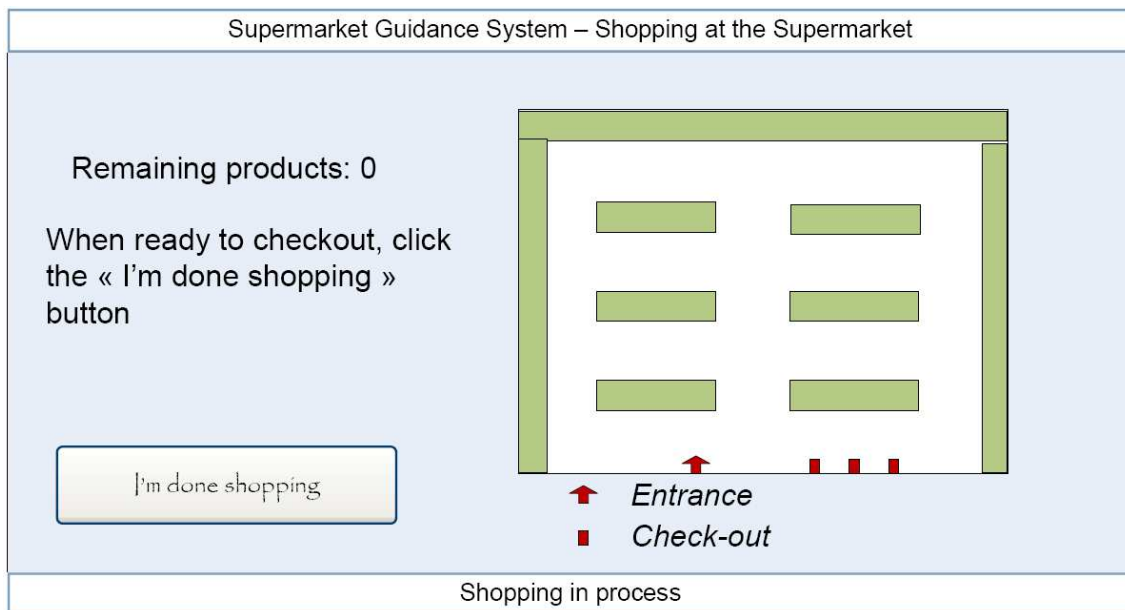


Figure A.23: Shopping at the Supermarket - End of the guidance

Queue Manager

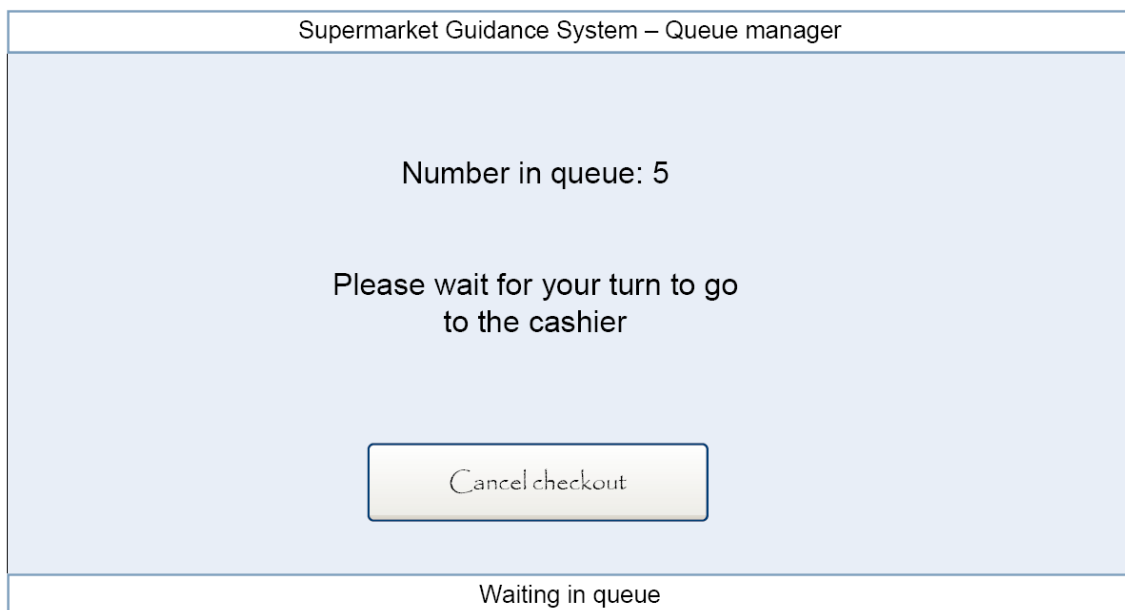


Figure A.24: Queue Manager 1

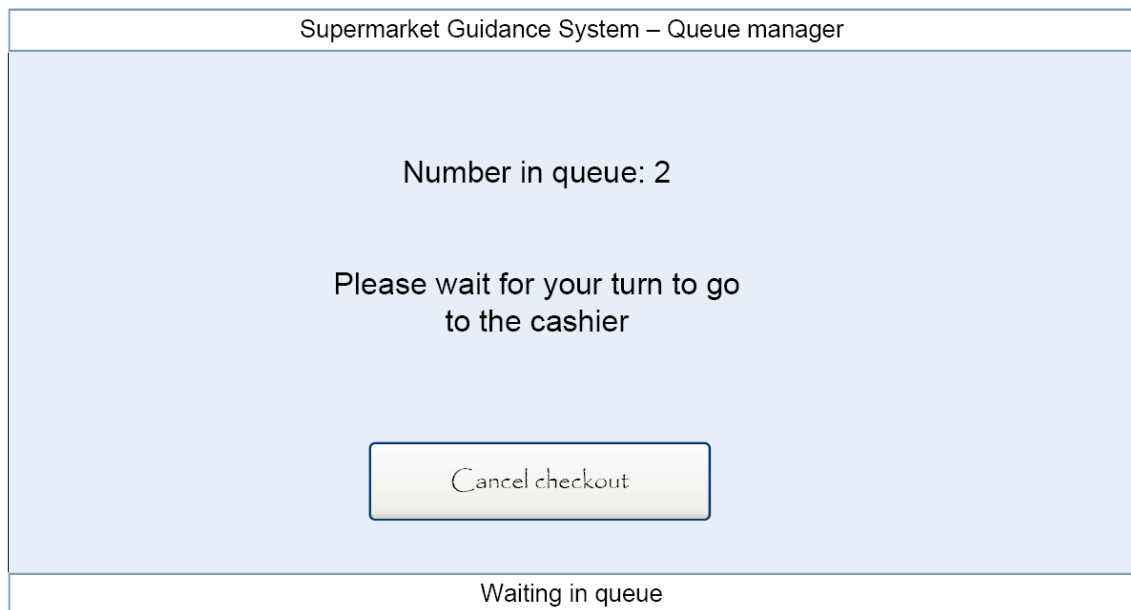


Figure A.25: Queue Manager 2

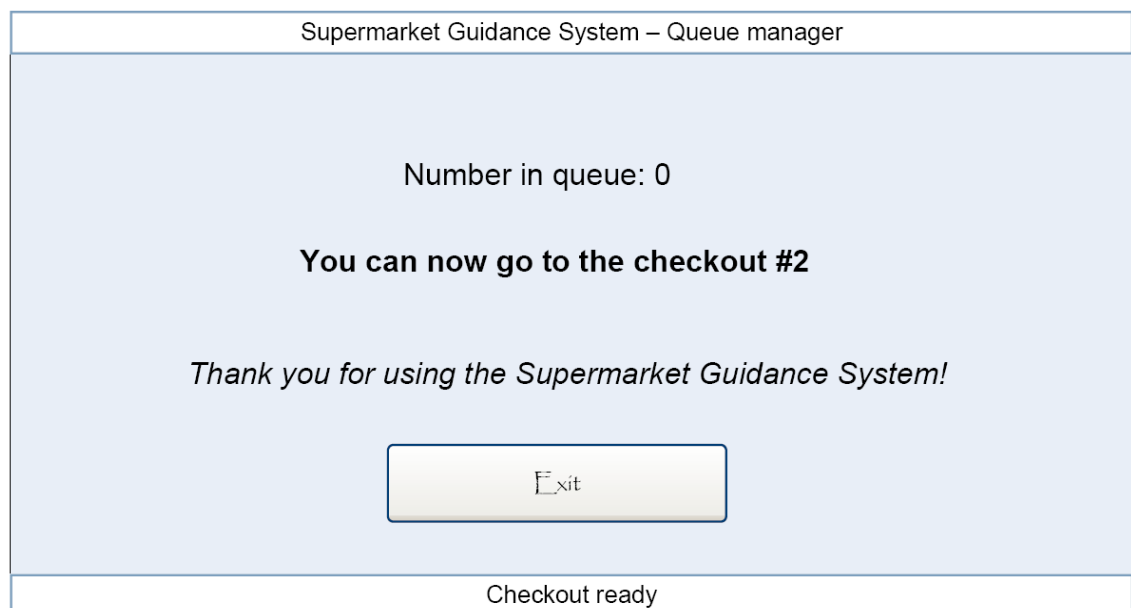


Figure A.26: Queue Manager 3

A.1.3 Updated Low-Fi Prototype

Shopping List Manager

Supermarket Guidance System – Shopping list manager

Produit	Notes
Potatoes	2kg

Home

Figure A.27: Updated Shopping List Manager - Home Page

Supermarket Guidance System – Shopping list manager

Produit	Notes
Potatoes	2kg

Home

Figure A.28: Updated Shopping List Manager - Home Page - Product Selected

Shopping at the Supermarket

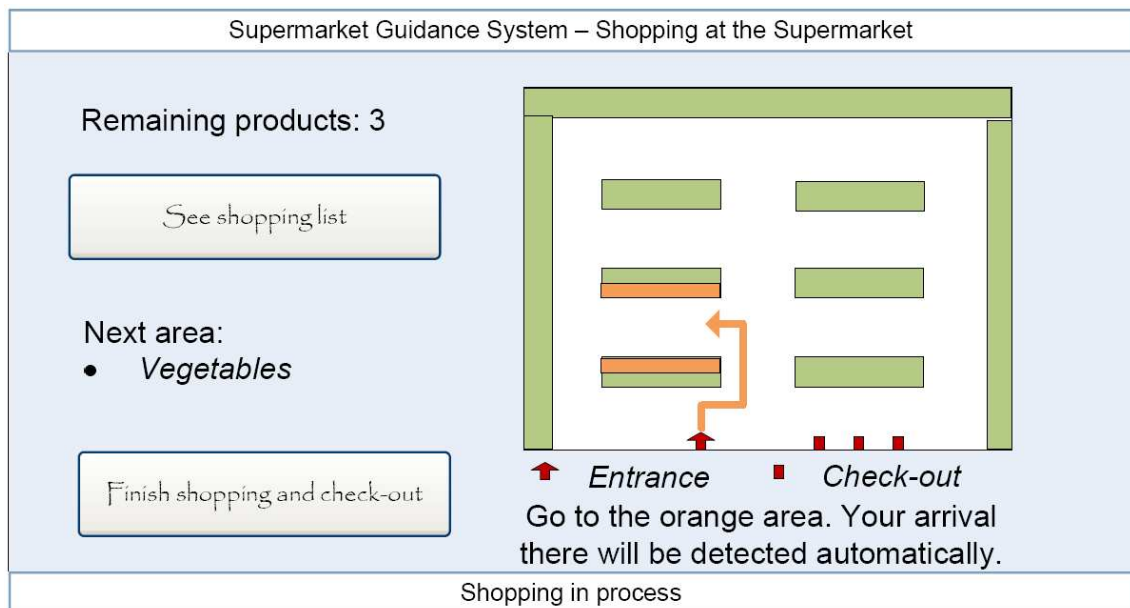


Figure A.29: Updated Shopping at the Supermarket - Next Area

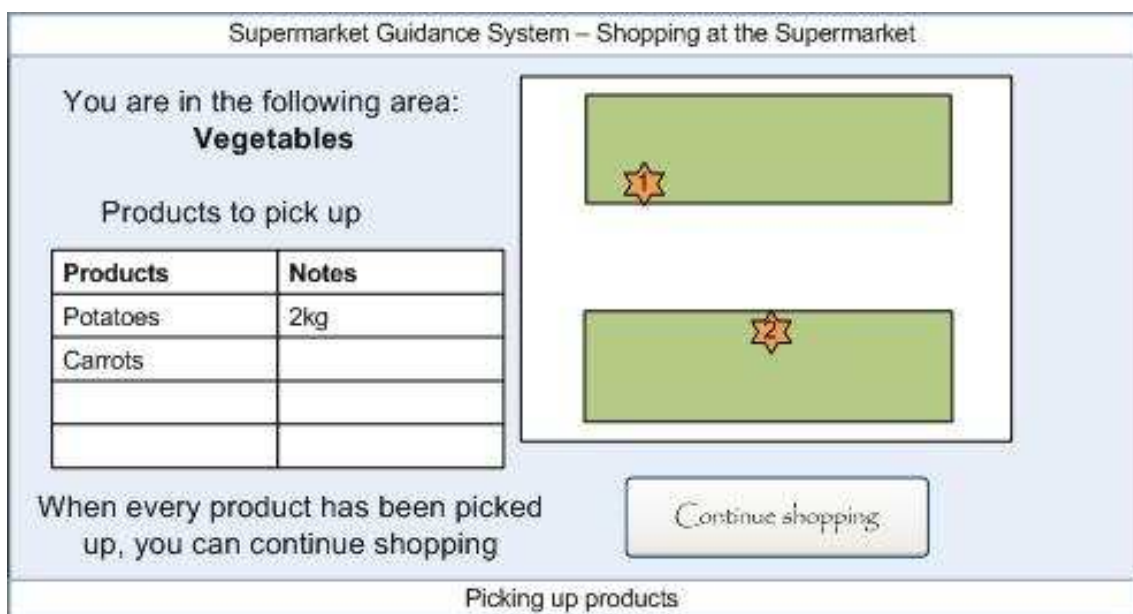


Figure A.30: Updated Shopping at the Supermarket - Area

Supermarket Guidance System – Shopping list manager	
Shopping list	
Products	Notes
Carrots	1kg of fresh carrots
Coca-Cola	
Potatoes	2kg
<div style="border: 1px solid black; padding: 10px; display: inline-block; background-color: #f0f0f0;">Close</div>	
Shopping list	

Figure A.31: Updated Shopping at the Supermarket - Shopping list screen

A.1.4 Answers to the survey

The application is easy to use

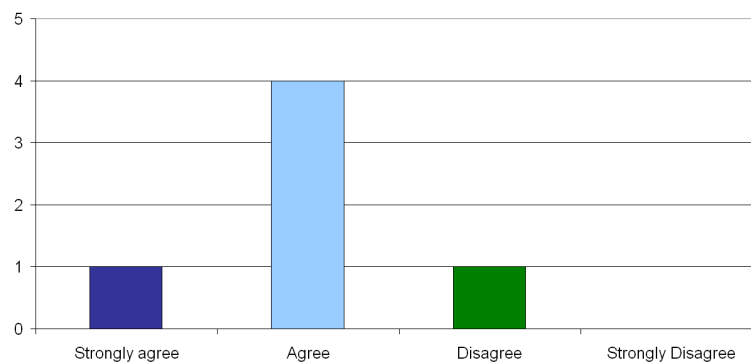


Figure A.32: Answers to "The application is easy to use"

The general design of the program is good

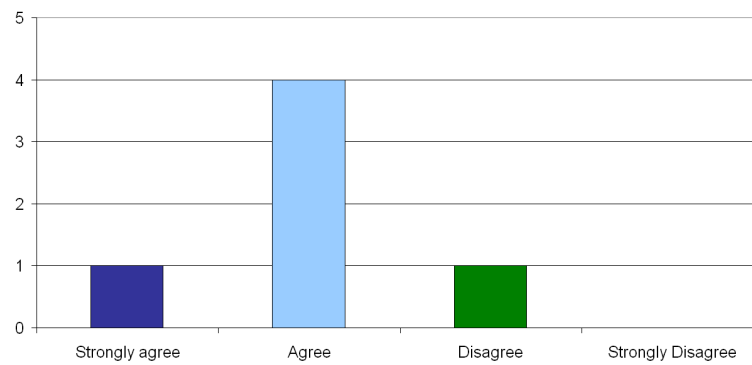


Figure A.33: Answers to "The general design of the program is good"

It is easy to browse through the application without getting lost

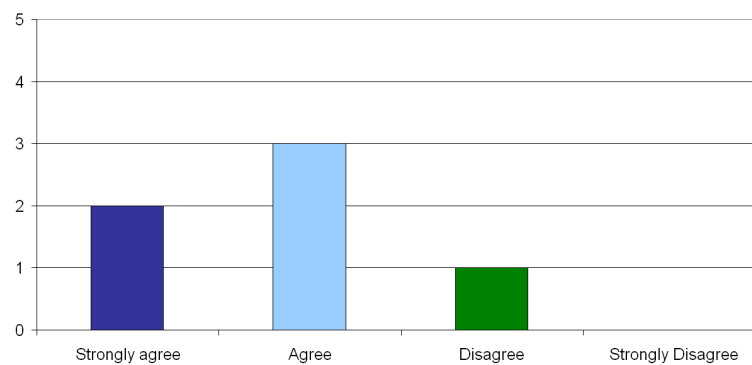


Figure A.34: Answers to "It is easy to browse through the application without getting lost"

The functioning of the program is logic and easy to remember

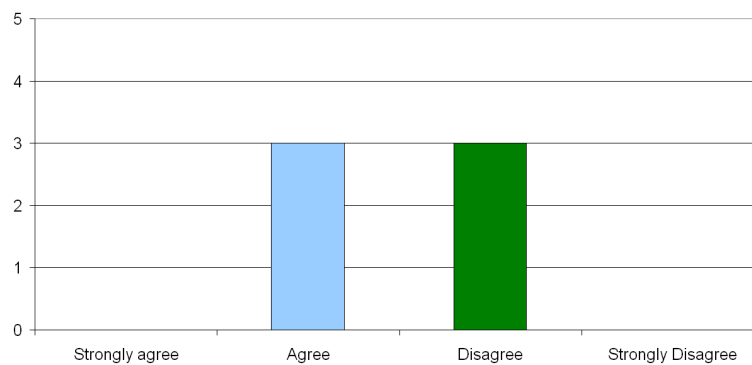


Figure A.35: Answers to "The functioning of the program is logic and easy to remember"

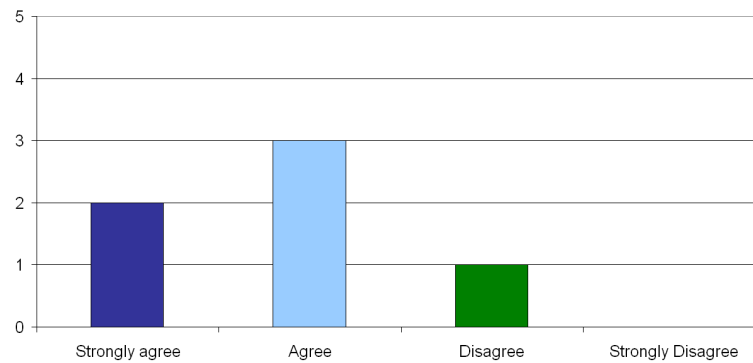
The application is enjoyable to use

Figure A.36: Answers to "The application is enjoyable to use"

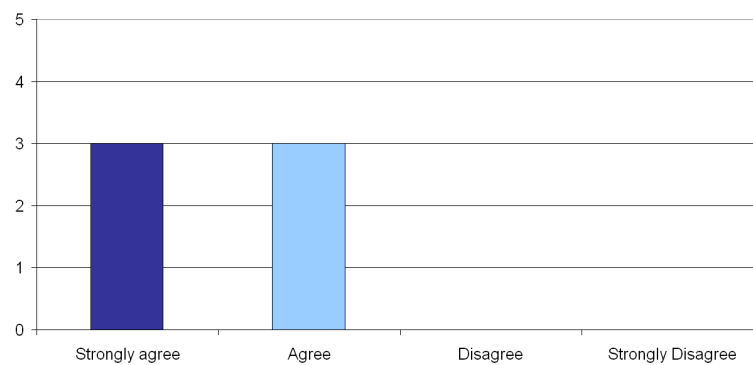
The application seems to be reliable

Figure A.37: Answers to "The application seems to be reliable"

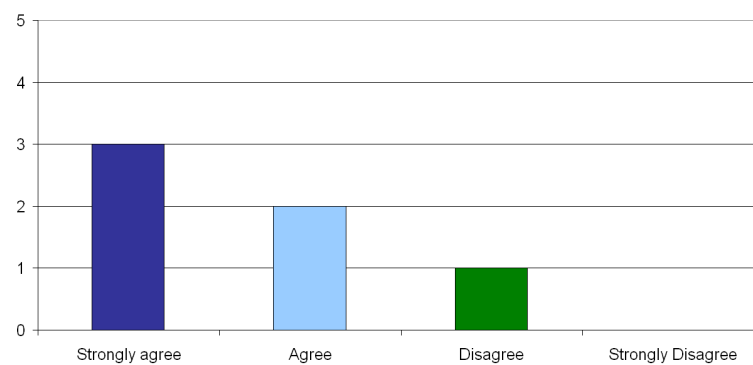
It is easy to build the shopping list

Figure A.38: Answers to "It is easy to build the shopping list"

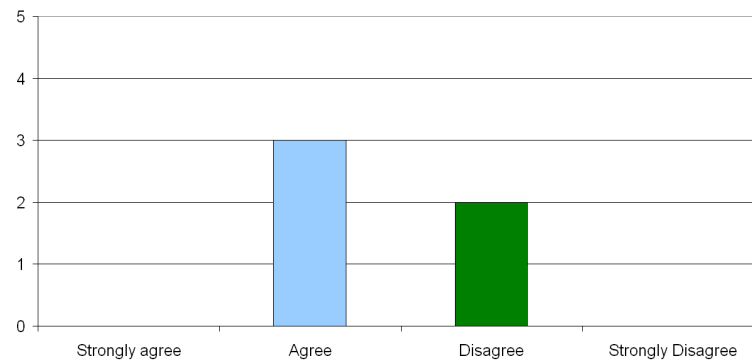
The way of building the shopping list is the best one

Figure A.39: Answers to "The way of building the shopping list is the best one"

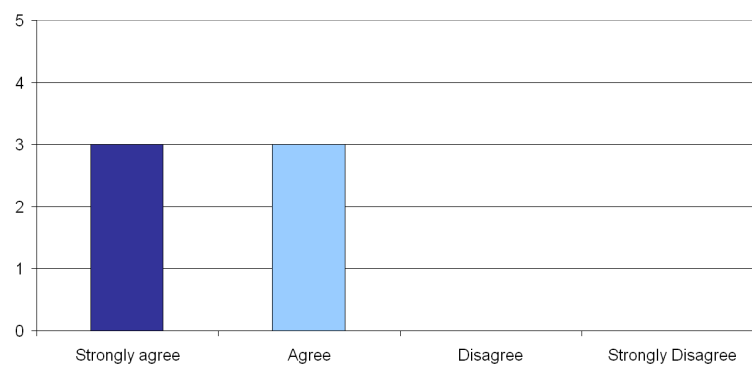
It is easy to understand the directions inside the supermarket

Figure A.40: Answers to "It is easy to understand the directions inside the supermarket"

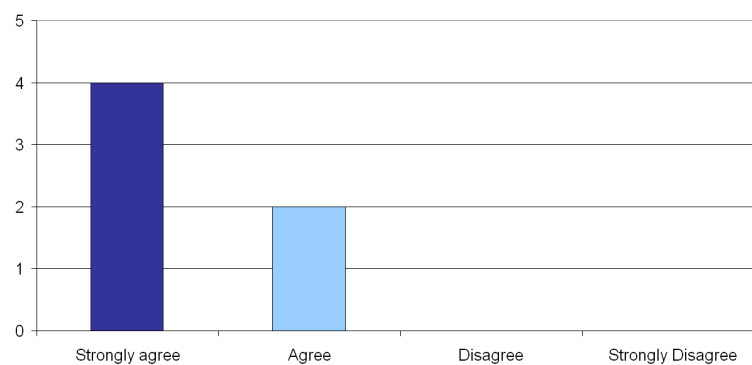
A map is a good way of providing directions

Figure A.41: Answers to "A map is a good way of providing directions"

The way of providing directions in the supermarket is the best one

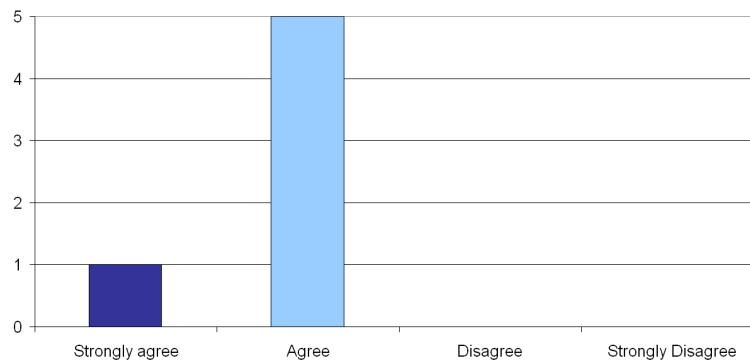


Figure A.42: Answers to "The way of providing directions in the supermarket is the best one"

It is annoying not to be able to see the whole shopping list and to modify it while shopping

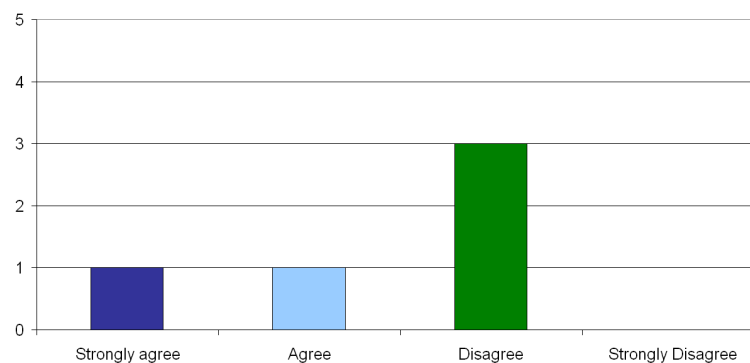


Figure A.43: Answers to "It is annoying not to be able to see the whole shopping list and to modify it while shopping"

A.2 Tests of the implemented system

A.2.1 Introduction text

You are part of a testing panel for the implementation of a mobile application. The goal of this application is to help the user shopping at a supermarket.

There are two different parts to the system and you will be asked to test both. You will be equipped with a tablet PC which contains the two applications you will be asked to test.

The first part is the Shopping List Manager, which is used to build a shopping list before going shopping. The tests of this part will be done in the main Kantine.

The second part is the Shopping Guide, which is used to guide the user when he is in the actual supermarket. The tests of this part will be processed in the library.

For each part, we will launch the application for you.

You will be given a stylus that you will use through the interface. The second part of the test also implies actions such as walking in the supermarket, taking products and the last part of the tests implies

going to check out.

You will be provided a task list for both parts of the system. You will have to follow the task list closely.

We will take notes during the test. Our goal is to see which difficulties you encounter, how long it takes you to achieve the different tasks, etc. However, feel free to express your feeling anytime: we want to get as much feedback as possible. For the same purpose, you will also be asked to answer a survey at the end of the test.

About this survey, we ask you to express your own feeling. You should try to use the whole scale instead of using only the two answers "I'm satisfied" and "I'm not satisfied" through the whole survey.

If you still have questions, you can ask us now. When we are done with the questions, the test will start.

Survey

1. Personal details

- Do you often go shopping?
 - Yes, quite often
 - Sometimes
 - Rarely
- Did you participate in the usability tests of this system?
 - Yes, I tested the prototype on paper one month ago
 - No, this is my first test
- How comfortable are you with the use of mobile devices such as PDAs, Smartphones or Tablet PCs?
 - Very comfortable, I often use handheld devices and I'm often use their advanced features
 - Quite comfortable, I usually can find out how to do what i want to do without too much trouble
 - I'm not really comfortable with such devices and I often struggle to do what I want

2. Ratings

- The application is easy to use:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The general design of the program is good:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- It is easy to browse through the application without getting lost:
 - Strongly agree

- Agree
 - Disagree
 - Strongly disagree
- The functioning of the program is logic and easy to remember:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The application is enjoyable to use:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- The application works properly (there is no bug, etc.):
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- It is easy to build the shopping list:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree
- It is easy to understand what I have to do in the supermarket:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree

Suggestions?:

- It is better to shop with this system than to shop on my own as I usually do:
 - Strongly agree
 - Agree
 - Disagree
 - Strongly disagree

Why?:

- It is easy to shop and to hold/ interact with the tablet PC at the same time:
 - Strongly agree
 - Agree

- Disagree
- Strongly disagree

3. Comments

Do you have any comments?

A.2.2 Answers to the survey

The application is easy to use

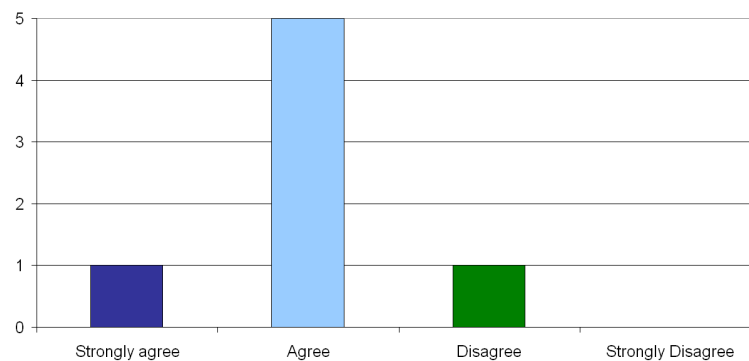


Figure A.44: Answers to "The application is easy to use"

The general design of the program is good

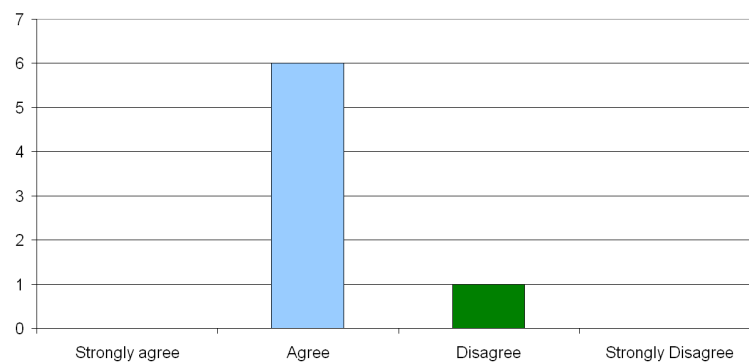


Figure A.45: Answers to "The general design of the program is good"

It is easy to browse through the application without getting lost

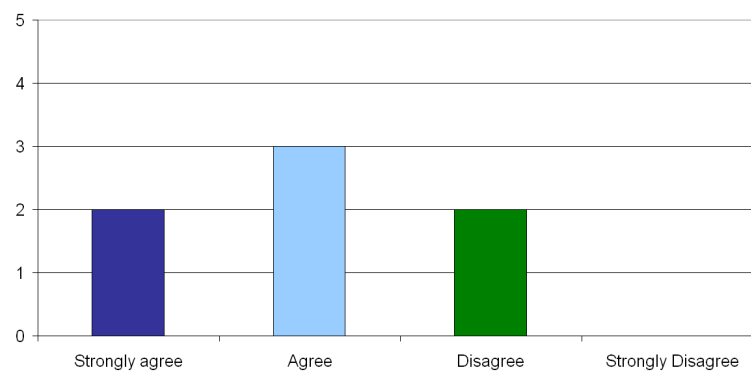


Figure A.46: Answers to "It is easy to browse through the application without getting lost"

The functioning of the program is logic and easy to remember

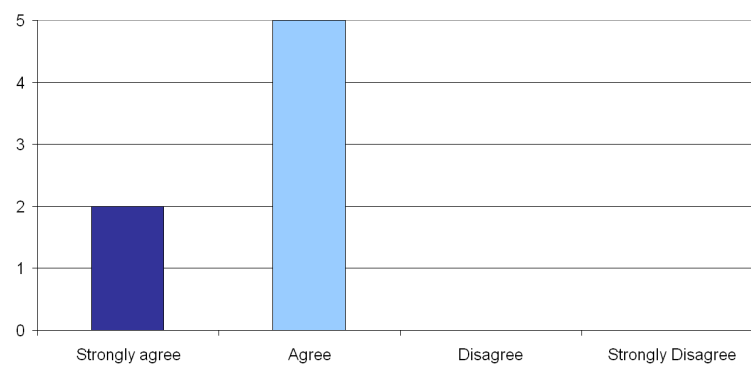


Figure A.47: Answers to "The functioning of the program is logic and easy to remember"

The application is enjoyable to use

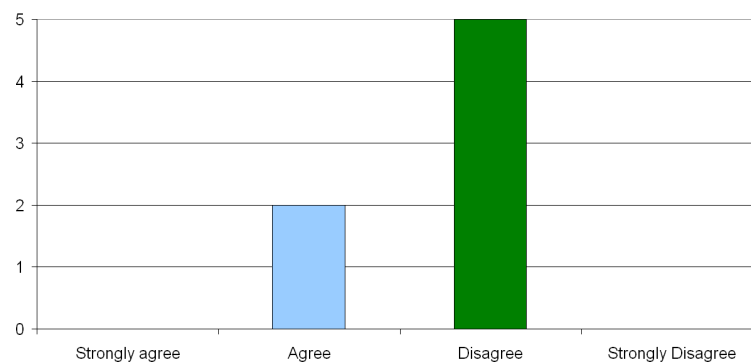


Figure A.48: Answers to "The application is enjoyable to use"

The application works properly (there is no bug, etc.)

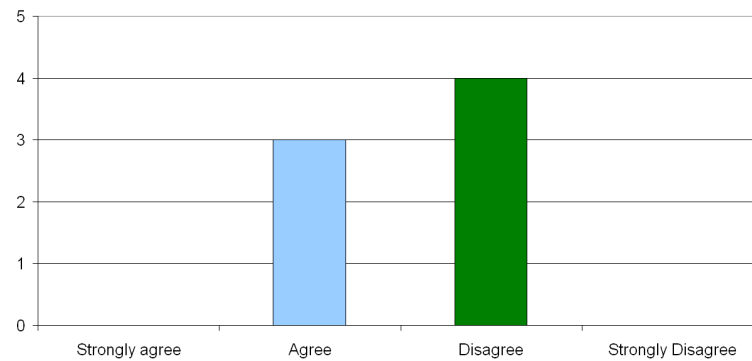


Figure A.49: Answers to "The application works properly"

It is easy to build the shopping list

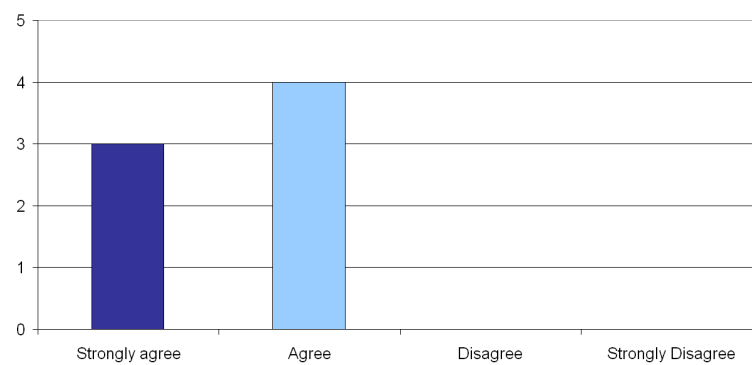


Figure A.50: Answers to "It is easy to build the shopping list"

It is easy to understand what I have to do in the supermarket

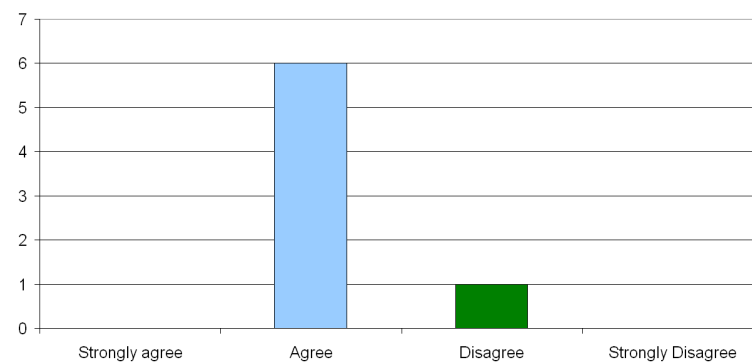


Figure A.51: Answers to "It is easy to understand what I have to do in the supermarket"

It is better to shop with this system than to shop on my own as I usually do

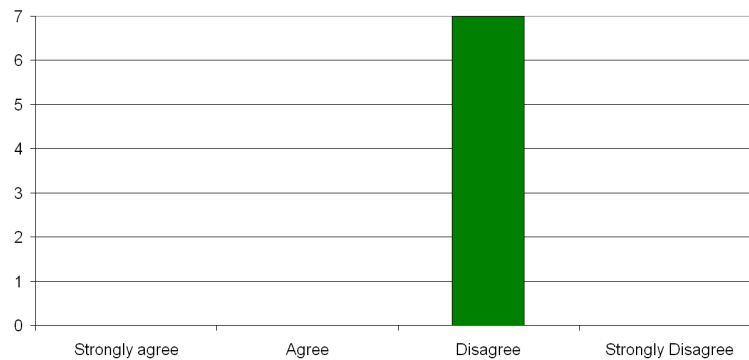


Figure A.52: Answers to "It is better to shop with this system than to shop on my own as I usually do"

It is easy to shop and to hold/interact with the tablet PC at the same time

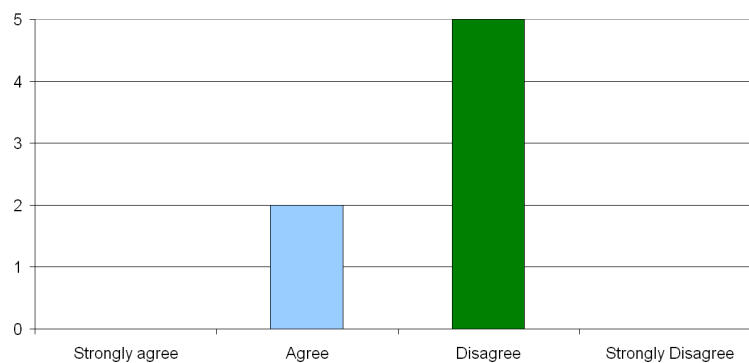


Figure A.53: Answers to "It is easy to shop and to hold/interact with the tablet PC at the same time"

Pictures of the final tests



Figure A.54: The library



Figure A.55: The library



Figure A.56: A tester picking up a product

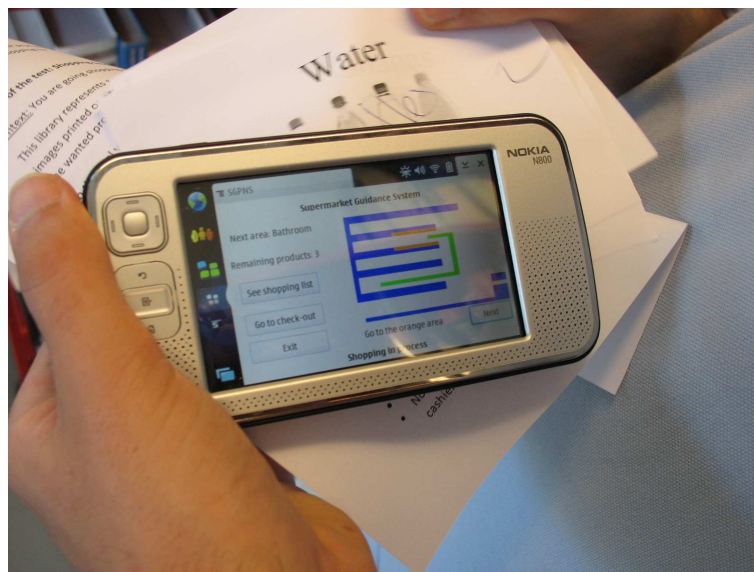


Figure A.57: The application running on the tablet PC



Figure A.58: A tester and the facilitator



Figure A.59: A tester having difficulty to interact with the mobile device while holding the "product"



Figure A.60: A tester confused by mistakes on the detailed map and struggling to find a product

A.3 Implementation of the saving of shopping lists

Initially, we only wanted to use one file to save the list. Every line would have had this structure:

```
0 1 Water Notes for water
0 0 Wine
0 1 Soda Notes for soda
```

The problem is that it was too complicated to get only the notes as we could not know for each product how many characters the name of the product has. A solution would be to write the notes on the next line:

```
0 1 Water
Notes for water
0 0 Wine

0 1 Soda
Notes for soda
```

But another problem accured because the user is allowed to write several lines of notes. So we would need to store somewhere the number of lines of the notes to be able to differentiate everytime the lines of notes and the lines of products in the shopping list file. We thought that it was a too complicated solution; therefore, we choose that it would be easier to use two files, one containing the products and another one containing the notes.

The structure of the shopping list file is shown below. This file is divided into two parts: the categories and the products.

- The categories part: On the first line, the number corresponds to the total number of categories. In our example, we have four categories. Then each category is described; the first number is the number of the category, the second value is the number of products belonging to this category and then the name of the category ends the line. When every category is defined, a '#' is written to announce the beginning of the products part.
- The products part: The products are classified by category in an increasing order. The first value is the number of the category corresponding to the product, the second one is a boolean (0 if the products does not belong to the shopping list, 1 if it part of the list) and finally, the name of the product ends the line. When every product has been defined, the characters '###' end the file.

```
4
0 3 Drinks
1 5 Fruits
2 4 Vegetables
3 2 Meat
#
0 1 Water
0 0 Wine
0 0 Soda
1 0 Apples
1 0 Bananas
1 0 Grapes
1 0 Pears
1 0 Oranges
2 1 Potatoes
2 0 Carrots
2 0 Courgettes
2 0 Concombre
3 0 Pork
3 0 Beef
###
```

The structure of the notes file is shown below. For each product, the name of the product is written, then on the following line are written the notes corresponding to the product whose name was written before. When there is not anymore products, the characters '###' end the file.

Notes: At the beginning, we wanted to allow the user to input several lines of notes. But during the implementation of the text fields in the interface, we realized that it would be nicer to have only one single line of notes and that anyway the user should not need so much space for the notes.

Water
1L Evian
Wine

Soda

Apples

Bananas

Grapes

Pears

Oranges

Potatoes

3 kg

Carrots

Courgettes

Concombre

Pork

Beef

###

A.4 Design of the map display via xpm data

This is an example of a xpm file:

```

exemple_xpm=[
"16 16 3 1",
"    c None",
".    c #000000",
"X    c #FFFFFF",
"    ",
".....",
".XXX.X.",
".XXX.XX.",
".XXX.XXX.",
".XXX.....",
".XXXXXXXX.",
".XXXXXXXX.",
".XXXXXXXX.",
".XXXXXXXX.",
".XXXXXXXX.",
".XXXXXXXX.",
".XXXXXXXX.",
".XXXXXXXX.",
".....",
"    "]

```

Xpm data (X Pixmap data) is a table of one dimension. The first part of this table is the configuration data, the second part is the pixel data. The configuration data specifies the size of the image and the color corresponding to every character of the pixel data. In our example, the size of the image is 16*16 and the character '.' correspond to the color #000000 in RGB (i.e. black). Each pixel of the image is following. When the image will be displayed every character of this pixel data will be associated to the color previously specified.

For our project, we thought to use this technology for the display of the maps during the guidance of the user in the supermarket. In that case, the values of the integers correspond to if the pixel is part of a shelf, the floor, a cashier or the path (the arrow) the customer should follow to go to the wanted area. These data will then be displayed as an image on the screen.

A.5 Code of the server - guidance

```

while (found == false)
{
strQuery = "SELECT x, y, x_parent, y_parent FROM customers_path " +
"WHERE id_customer = " + id_customer + " AND cost = " + cost;
connection = DatabaseManager.getConnection();
results = DatabaseManager.executeQuery(connection, strQuery);
//Going through all the cases with the same cost
while (results.next() == true && found == false)
{
strQuery = "SELECT x, y, type, detail FROM map1 " +
"WHERE (type = 3 OR type = 4) AND (" +
"(x = " + (results.getInt(1) - 1) + " AND y = " + (results.getInt(2)) + ") OR " +
"(x = " + (results.getInt(1) + 1) + " AND y = " + (results.getInt(2)) + ") OR " +
"(x = " + (results.getInt(1)) + " AND y = " + (results.getInt(2) - 1) + ") OR " +
"(x = " + (results.getInt(1)) + " AND y = " + (results.getInt(2) + 1) + "))";
Connection connection2 = DatabaseManager.getConnection();
ResultSet results2 = DatabaseManager.executeQuery(connection2, strQuery);
//Going through all the next cases from the current case
while (results2.next() == true && found == false)
{
//Make sure the current case is not in the customers_path table already
strQuery = "SELECT x FROM customers_path WHERE id_customer = " + id_customer +
AND x = " + results2.getInt(1) + " AND y = " + results2.getInt(2);
Connection connection3 = DatabaseManager.getConnection();
ResultSet results3 = DatabaseManager.executeQuery(connection3, strQuery);
if(results3.next() == false) //The case is not in customers_path yet
{
//Check if it's a spot which is already in the customers_spots list
strQuery = "SELECT x, y, category FROM customers_spots WHERE " +
"id_customer = " + id_customer + " AND x = " + results2.getInt(1) +
" AND y = " + results2.getInt(2);
Connection connection4 = DatabaseManager.getConnection();
ResultSet results4 = DatabaseManager.executeQuery(connection4, strQuery);
if (results4.next() == true) //We found the destination !!!
{
(...)
/*This part takes care of everything that requires attention once the closest spot has been found*/
}
else
{
strInstruction = "INSERT INTO customers_path (id_customer, x, y,
_parent, y_parent, cost) " +
"VALUES (" + id_customer + ", " + results2.getInt(1) + ", " + results2.getInt(2) + ", " +
results.getInt(1) + ", " + results.getInt(2) + ", " + (cost+1) + ")";
DatabaseManager.executeUpdate(strInstruction);
}
results4.close();
connection4.close();
}
results3.close();
connection3.close();
}
cost++;
results.close();
connection.close(); }

```

A.6 Database

```

CREATE TABLE 'checkouts' (
'id' tinyint(3) unsigned NOT NULL default '0',
'free' enum('0','1') NOT NULL default '0',
PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'customers_path' (
'id_customer' int(10) unsigned NOT NULL default '0',
'x' tinyint(3) unsigned NOT NULL default '0',
'y' tinyint(3) unsigned NOT NULL default '0',
'x_parent' tinyint(3) unsigned NOT NULL default '99',
'y_parent' tinyint(3) unsigned NOT NULL default '99',
'cost' tinyint(3) unsigned NOT NULL default '0',
PRIMARY KEY ('id_customer','x','y')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
CREATE TABLE 'customers_position' (
'id' int(10) unsigned NOT NULL default '0',
'x' tinyint(3) unsigned default NULL,
'y' tinyint(3) unsigned default NULL,
PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'customers_queue' (
'place' int(10) unsigned NOT NULL auto_increment,
'id_customer' int(10) unsigned NOT NULL default '0',
PRIMARY KEY ('place')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'customers_ref_products' (
'id_customer' int(10) unsigned NOT NULL default '0',
'intOnZoomedMap' tinyint(3) unsigned NOT NULL default '0',
'label' varchar(30) NOT NULL default "",
'id_product' tinyint(3) unsigned NOT NULL default '0',
PRIMARY KEY ('id_customer','intOnZoomedMap')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'customers_shopping_list' (
'id' int(10) unsigned NOT NULL default '0',
'id_product' int(10) unsigned NOT NULL default '0',
'taken' enum('0','1') NOT NULL default '0',
PRIMARY KEY ('id','id_product')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'customers_spots' (
'id_customer' int(10) unsigned NOT NULL default '0',
'x' tinyint(3) unsigned NOT NULL default '0',
'y' tinyint(3) unsigned NOT NULL default '0',
'category' tinyint(3) unsigned NOT NULL default '0',
PRIMARY KEY ('id_customer','x','y')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

```

CREATE TABLE 'customers_zoomed_map' (
'id_customer' int(10) unsigned NOT NULL default '0',
'x' tinyint(3) unsigned NOT NULL default '0',
'value' tinyint(3) unsigned NOT NULL default '0',
PRIMARY KEY ('id_customer','x')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'map1' (
'x' tinyint(3) unsigned NOT NULL default '0',
'y' tinyint(3) unsigned NOT NULL default '0',
'type' tinyint(3) unsigned default NULL,
'detail' tinyint(3) unsigned default NULL,
PRIMARY KEY ('x','y')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

CREATE TABLE 'ref_categories' (
'id' int(10) unsigned NOT NULL default '0',
'label' varchar(20) NOT NULL default '',
PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO 'ref_categories' VALUES (1, 'Drinks');
INSERT INTO 'ref_categories' VALUES (2, 'Fruits');
INSERT INTO 'ref_categories' VALUES (3, 'Vegetables');
INSERT INTO 'ref_categories' VALUES (4, 'Meat');
INSERT INTO 'ref_categories' VALUES (5, 'Dairy products');
INSERT INTO 'ref_categories' VALUES (6, 'Breakfast');
INSERT INTO 'ref_categories' VALUES (7, 'Bathroom');
INSERT INTO 'ref_categories' VALUES (8, 'Kitchen');

CREATE TABLE 'ref_products' (
'id' int(10) unsigned NOT NULL default '0',
'id_category' int(10) unsigned NOT NULL default '0',
'label' varchar(20) NOT NULL default '',
PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO 'ref_products' VALUES (1, 1, 'Water');
INSERT INTO 'ref_products' VALUES (2, 1, 'Wine');
INSERT INTO 'ref_products' VALUES (3, 1, 'Soda');
INSERT INTO 'ref_products' VALUES (4, 2, 'Apples');
INSERT INTO 'ref_products' VALUES (5, 2, 'Bananas');
INSERT INTO 'ref_products' VALUES (6, 2, 'Grapes');
INSERT INTO 'ref_products' VALUES (7, 2, 'Pears');
INSERT INTO 'ref_products' VALUES (8, 2, 'Oranges');
INSERT INTO 'ref_products' VALUES (9, 3, 'Potatoes');
INSERT INTO 'ref_products' VALUES (10, 3, 'Carrots');
INSERT INTO 'ref_products' VALUES (11, 3, 'Courgettes');
INSERT INTO 'ref_products' VALUES (12, 3, 'Concombre');
INSERT INTO 'ref_products' VALUES (13, 4, 'Pork');
INSERT INTO 'ref_products' VALUES (14, 4, 'Beef');

```

```
INSERT INTO 'ref_products' VALUES (15, 5, 'Cheddar');
INSERT INTO 'ref_products' VALUES (16, 5, 'Milk');
INSERT INTO 'ref_products' VALUES (17, 5, 'Yoghurt');
INSERT INTO 'ref_products' VALUES (18, 6, 'Coffee');
INSERT INTO 'ref_products' VALUES (19, 6, 'Cereals');
INSERT INTO 'ref_products' VALUES (20, 6, 'Tea');
INSERT INTO 'ref_products' VALUES (21, 7, 'Soap');
INSERT INTO 'ref_products' VALUES (22, 7, 'Toothbrush');
INSERT INTO 'ref_products' VALUES (23, 7, 'Toothpaste');
INSERT INTO 'ref_products' VALUES (24, 8, 'Plates');
INSERT INTO 'ref_products' VALUES (25, 8, 'Knives');
INSERT INTO 'ref_products' VALUES (26, 8, 'Forks');
INSERT INTO 'ref_products' VALUES (27, 8, 'Spoons');
```

B

Glossary

Edge Router (ER): The Edge Router , which is not mandatory in the PN architecture, can aid the PN organization and networking by implementing some of the PN functionality which are normally handled by the gateway node. This will improve the performances of the PN.

Personal Network (PN): The Personal Network is the network that connects every device of one specific user (hence the PN is user-centric).

A PN is made of clusters, which are Personal Area Networks (def) (the home cluster which is made of the devices that are located at home, the office cluster, etc.), which are interconnected, in most cases through Internet.

Personal Area Network (PAN): A computer network made of devices such as computers, smartphones, PDAs, etc. The devices can be connected over-the-air (with bluetooth, infrared connexion, etc.) or in a wired way. Usually, the range of a PAN is of a few meters.

C

Bibliography

C.1 Personnal Network and Magnet Technology

<http://www.ist-magnet.org/>, april 2008

C.2 Tablet PC

http://en.wikipedia.org/wiki/Tablet_PC, may 2008

<http://www.microsoft.com/windowsxp/tabletpc/default.mspx>, may 2008

C.3 Intelligent supermarkets

<http://www-03.ibm.com/industries/retail/doc/content/bin/stop26shopfinal.pdf>, april 2008

http://www.springboardnetworks.com/assets/resources/TechPoint_BR_2007.pdf, april 2008

C.4 The location of the customer in the supermarket

<http://en.wikipedia.org/wiki/Gps>, april 2008

<http://en.wikipedia.org/wiki/Rfid>, april 2008

<http://en.wikipedia.org/wiki/Bluetooth>, april 2008

<http://www.commentcamarche.net/guide-achat/accessoires-pda-203-prix>, may 2008

<http://www.righttag.com/buyrfid>, may 2008

<http://www.rfid.co.uk>, may 2008

C.5 The automatic bill

<http://en.wikipedia.org/wiki/Barcode>, april 2008

http://searchcio.techtarget.com/sDefinition/0,,sid182_gci213536,00.html, april 2008

<http://www.barcodesinc.com/generator/index.php>, april 2008

<http://www.tec-it.com/online-demos/tbarcode/barcode-generator.aspx?LANG=en>, april 2008

C.6 Choice of development language - Mobile Programming

http://maemo.org/development/documentation/how-tos/4-x/maemo_architecture.html, may 2008

<http://danny.damours.net/wordpress/index.php/archive/java-and-erlang-on-the-maemo-platform-nokia-n8>

may 2008 http://wiki.forum.nokia.com/index.php/Getting_started_with_Java_on_maemo, may 2008

C.7 Design and implementation of the mobile application

Jenny Preece, Yvonne Rogers and Helen Sharp, *Interaction Design: beyond human-computer interaction*, 2nd edition

<http://www.id-book.com/>

Gérard Swinnen, *Apprendre à Programmer avec Python*, 2nd edition, O'REILLY

<http://maemo.org/>, may 2008

<http://faq.pygtk.org>, may 2008

<http://python.developpez.com/cours/pygktutorial>, may 2008

<http://mail.python.org>, may 2008

<http://svn.python.org/projects/python/trunk/Doc/library/xmlrpclib.rst>, may 2008

<http://hgdeoro.blogspot.com/2007/10/comunicacin-java-python-con-xml-rpc.html>, may 2008

C.8 xmlrpc

<http://www.xmlrpc.com/>, may 2008

Simon St. Laurent, Joe Johnston, Edd Dumbill, *Programming Web Services with XML-RPC*, O'Reilly, 2001

C.9 The server

Claude Delannoy, *Programmer en Java*, 3rd edition, EYROLLES, 2003

C.10 The report: building and orthographe

Walter Appel, Céline Chevalier, Emmanuel Cornet, Sébastien Destreux, Jean-Julien Fleck and Paul Pichaureau, *LATEX pour l'impatient*, 2nd edition, MiniMax, june 2007

Dictionnary *Le Robert & Collins*, 8th edition, 2006

<http://www.wordreference.com/>, may 2008



References

[CONCIERGE]: http://www.springboardnetworks.com/assets/resources/TechPoint_BR_2007.pdf, april 2008 (©2006 Springboard Retail Networks Inc.)

[PN/FED/IMAGE]: Document Personal Network D2.3.1., page 73, 2008

[PN/Agent/IMAGE]: Document Personal Network D2.3.1., page 12, 2008

[PN/IMAGE]: <http://www.ist-magnet.org/technicalapproach>, may 2008

[RFID/TAG]: www.totaltags.fr, may 2008

[TabletPC/N800]: <http://www.nseries.com>, may 2008

[TabletPC/N810]: <http://www.nseries.com>, may 2008