Software Requirements Specification

for

OK+ Project

Version 1.0 approved

SEYED ALI JANNATPOUR
UML - CEWP MOD5B GRB 2174

Prepared by :

Oliver Thomas
Frerejean Thibault
Caron Fabien
Rivoherinjakanavalona Nathalie

Table of Contents

1 Introduction

1.1 Purpose

Unified Modeling Language is a generic modeling language used in software engineering field, in order to provide a standard way to present the design of a system. The goal of doing a project for the UML course is precisely to develop a system, in a team, and to use the tools in UML to carry out the project.

We are a team of 4 people, and we decided to develop an application for an e-commerce company. It is a sale and purchase platform between individuals.

In this report, we will see first, the project analysis phase, then the front-end and back-end development of the system.

1.2 Intended Audience and Reading Suggestions

This document is intended for all people who want to know the operation, and the tools which are used for the Ok+ application. This is for developer, in order to help them to achieve, improve or update the project. This is for project manager to oversee the project, and to see the progress of the project.

2 Overall Description

2.1 Product features

The application Ok+ is a platform for individual where they are able to sale product or/and buy product. The user can sign up by providing personal information. Once registered, he can log in an start buy product. He has an area for selling product. He has wallet and cart. The user can log out anytime he wants.

2.2 User Classes and Characteristics

We can identify one user classes in our system, but the user is a seller and a buyer.

Customer : A customer is the principal user of the application. He has 2 roles which are seller and buyer. He has a wallet with an amount of money. He has a wallet in the application, and he can add money from his bank account, or by selling a product, he can earn money. He can also put money in his bank account from the wallet of the application.

Seller : The seller can add new product for sell. The user has to complete form about the product, which requires the name of the product, the price, a comment about the product, and the type of the product. Once the form completed, other users can see the new product in the shop.

Buyer : The buyer has access to the shop and can display products switch type of the product. The user can select a product to add it in his cart. He can remove some items from his cart, or remove all items. Once he has selected all items he needs, he can buy his cart's content.

3 SYSTEM FEATURES

3.1 Use-Case Diagram

The following diagram is our Use-case Diagram. That summarizes actors of the system and what they are able to do or not to do. In our system, we have one actor, which is the customer. The customer is a seller and a buyer at the same time. So we have 3 different roles in the system.
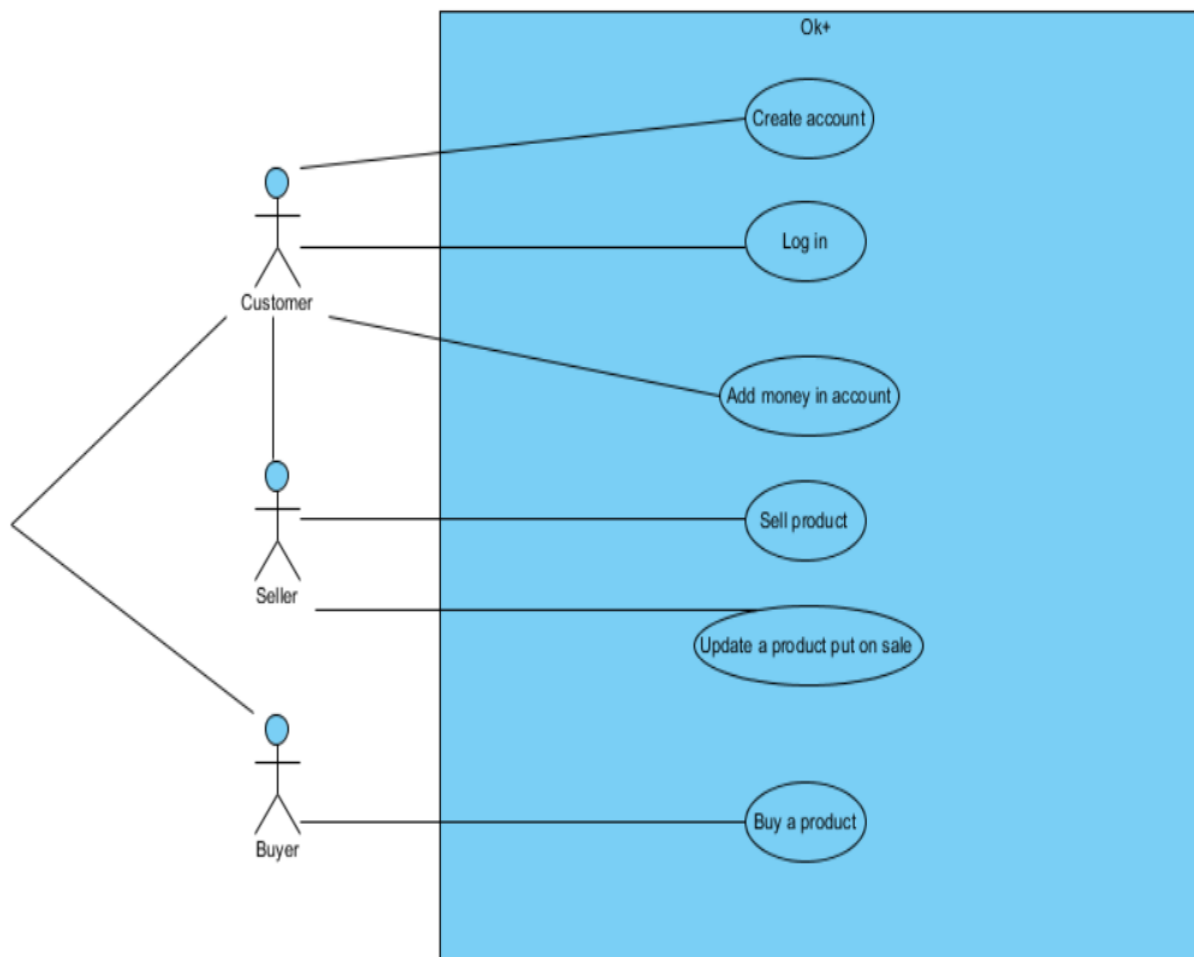

Figure 1 - Use-case Diagram

The Class Diagram allows us to see how classes are related to each other. For each class, it allows to know it private attributes, private and public methods.

Figure 2 - Class Diagram

Address

-number: int
-street: String
-city: String
-zipC: String

1..*
1

Customer

-id: int
-counterId: int
-firstN: String
-lastN: String
-balance: int
-email: String
-password: String
-cart: Arraylist<Product>

+getCart(): ArrayList<Product>
#updateCart(): void
+addMoney(money: int): void
+changePrice(product: Product, price: double): void
+sellP(price : double, name: String, description: String): void
+addCart(product: Product): void
+emptyCart(): void
+buy(): void
+buyP(product: Product): void

DisplayProduct

-product: Product

+display(): void

DisplayCustomer

-customer: Customer

+display(): void

DisplayAddress

-address: Address

+display(): void

«interface»
DisplayBehavior

CustomerContainer

#list: ArrayList<Customer>
-INSTANCE: CustomerContainer

+getInstance(): CustomerContainer
+addCustomer(firstN: String, lastN: String, email: String, password: String): void
+searchCustomer(email : String, password: String): Customer
+add(customer: Customer): void

1   1..*

RepositoryFactory

+getRepository(repType: String)

«Interface»
Repository

+load(): void
+save(): void

DbRepository

+load(): void
+save(): void

FileRepository

+load(): void
+save(): void

1
1..*

Product

-idCounter: int
-price: double
-id: int
-name: String
-Description: String
-owner: Customer

+getOwner(): Customer

1..*
0..1

ProductContainer

-INSTANCE: ProductContainer
#listProducts: ArrayLsit<Product>

+addProduct(product: Product): void
+suppressProduct(product: Product): void
+clearList(): void

1..*
0..1

Catalog

#cata: ArrayList<Product>

+addProduct(product : Product): void
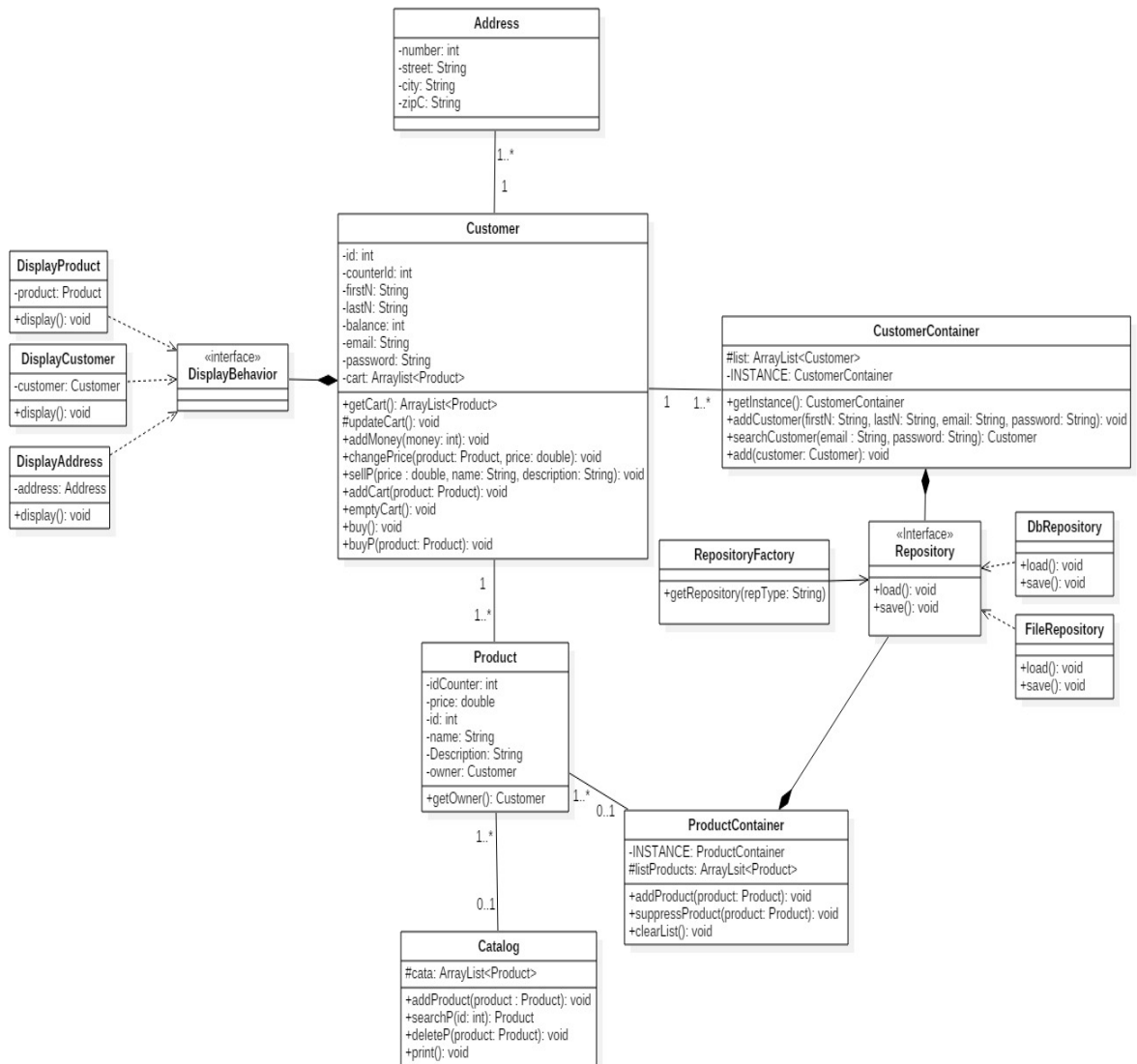+searchP(id: int): Product
+deleteP(product: Product): void
+print(): void

Figure 2 - Use-case Diagram

3.2 Create account Use-case

The following diagrams, are the scenario and sequence diagram of the "Create account" use-case.

3.2.1 Scenario

Number	1	
Name	create account scenario	
Summary	The Customer wants to create an account	
Priority	5	
Preconditions	Open application	
Postconditions	Signing in would be possible	
Primary Actor(s)	Customer	
Secondary Actor(s)	DataBase	
Trigger	Pressing "Sign UP" button	
Main scenario	Step	Action
	1	Open the registration window
	2	Put your information
	3	The database will check if the email is already used
	4	The information are saved in database

Figure 3 - Scenario - Create account use-case

3.2.2 Sequence Diagram
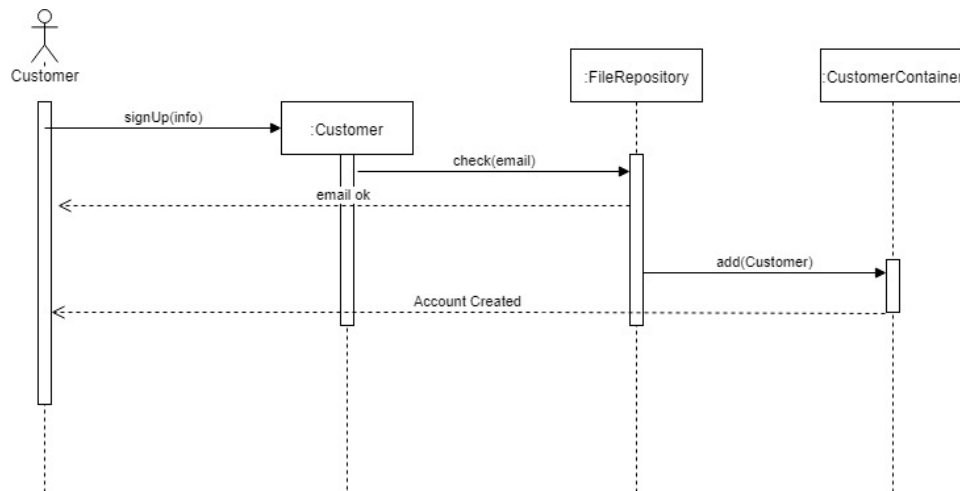
Figure 4 - Sequence Diagram – Create account

3.3 Log in Use-case

The following diagrams, are the scenario and sequence diagram of the "Log in" use-case.

3.3.1 Scenario

Number		2
Name		Log in scenario
Summary		The customer wants to log in the system
Priority		4
Preconditions		Open application and already registered
PostConditions		Access to the client session
PrimaryActor(s)		Customer
SecondaryActor(s)		DataBase
Trigger		Pressing "Sign In" button
Main Scenario	Step	Action
	1	Put email address and password
	2	Database check if the email address already exists and if it matches with a password
	3	Customer can access to his session

Figure 4 - Scenario - Log in use-case

3.3.2 Sequence Diagram
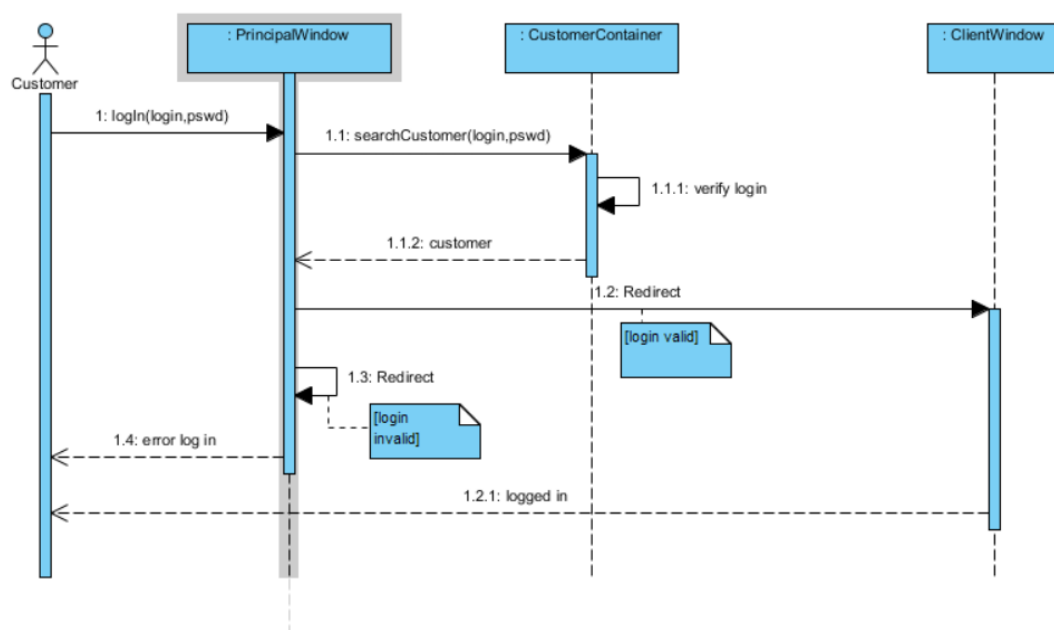
Figure 5 - Sequence diagram - Log in use-case

3.4 Buy product Use-case

The following diagrams, are the scenario and sequence diagram of the "buy product" use-case.

3.4.1   Scenario

Number		6
Name		Buy product scenario
Summary		The buyer wants to buy a product
Priority		4
Preconditions		Buyer has at least one item in his cart and has enough money
Post Conditions		Buyer 'balance changes
Primary Actor(s)		Customer
Secondary Actor(s)		DataBase
Trigger		Pressing "Buy" button
Main Scenario	Step	Action
	1	Get the product 'price
	2	Update balance of the buyer

Figure 6 - Scenario - Buy product use-case

3.4.2   Sequence Diagram
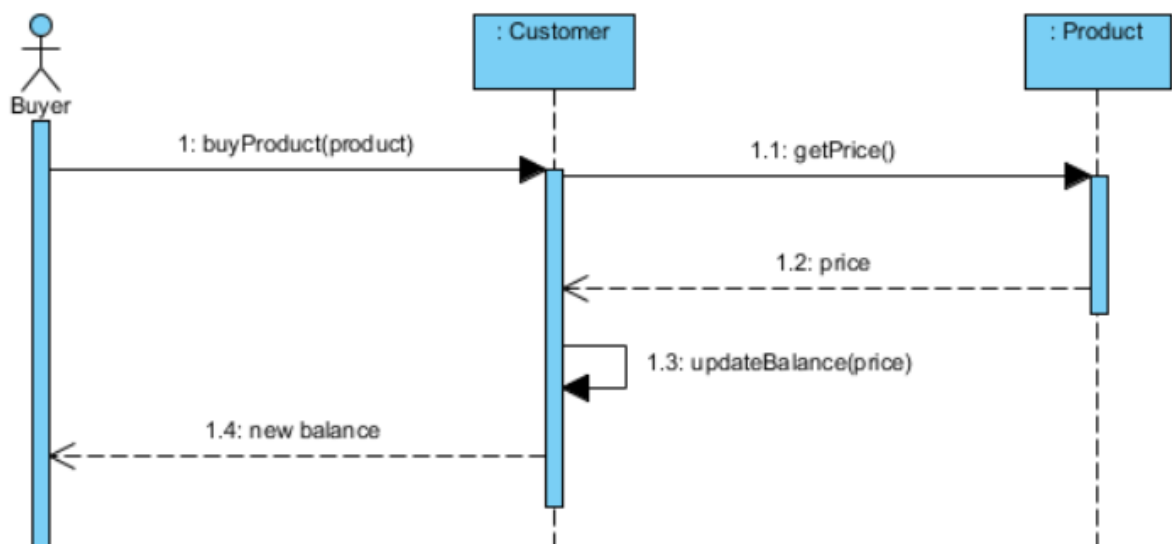
Figure 7 - Sequence Diagram - Buy product use-case

3.5 Sell product Use-case

The following diagrams, are the scenario and sequence diagram of the "Sell product" use-case.

3.5.1 Scenario

Number	4	
Name	Sell product scenario	
Summary	The customer wants to sell a product	
Priority	4	
Preconditions	Seller is in his session	
PostConditions	Add a product in Catalog	
PrimaryActor(s)	Customer	
SecondaryActor(s)	DataBase	
Trigger	Pressing "Sell product" button	
Main Scenario	Step	Action
	1	Put Information of the product
	2	Database add the product

Figure 8 - Scenario - Sell product use-case

3.5.2 Sequence Diagram
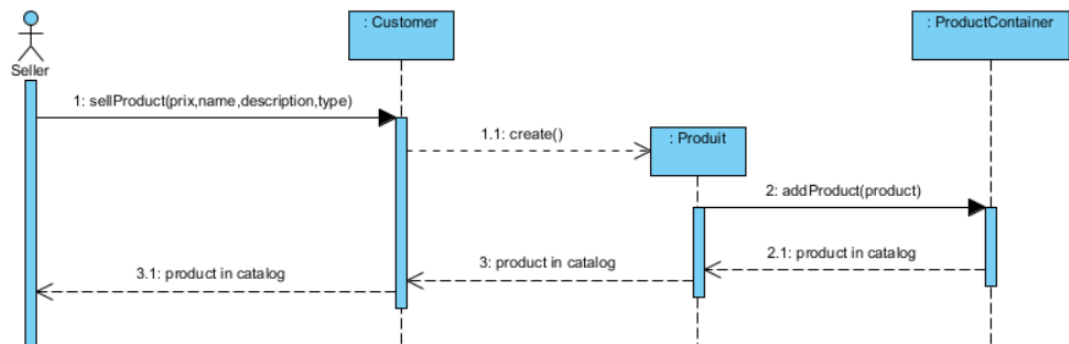
Figure 9 - Sequence Diagram - Sell product use-case

4 External Interface Requirements

4.1 4.1   User Interfaces

Swing library of Java is used to design the user interface.

4.1.1   Principal Window
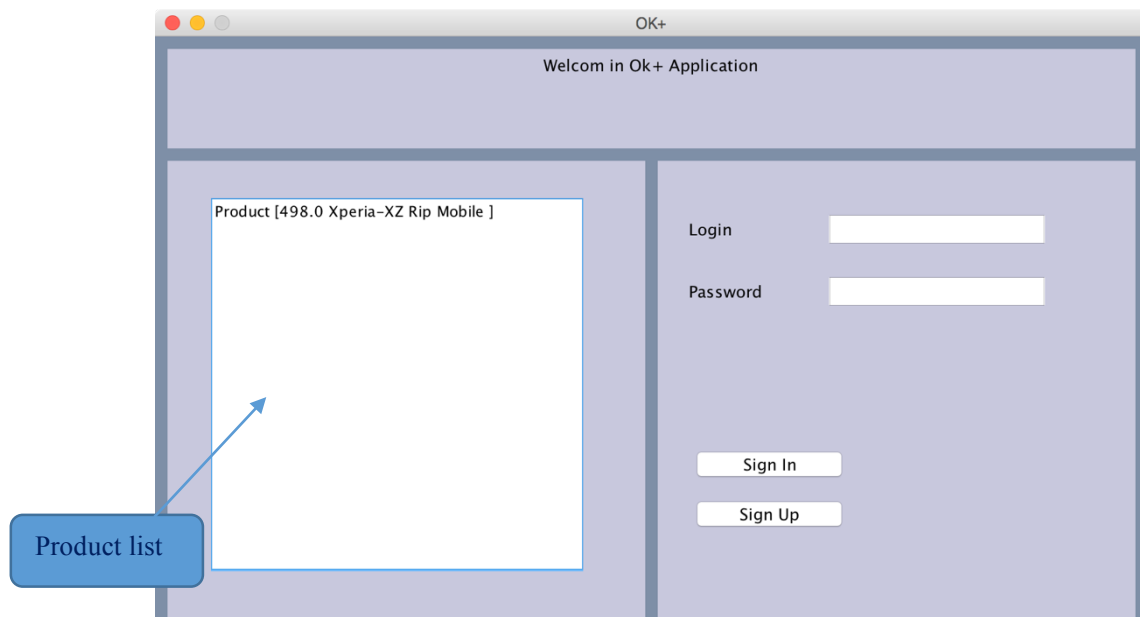
Figure 10 - Princiapl window
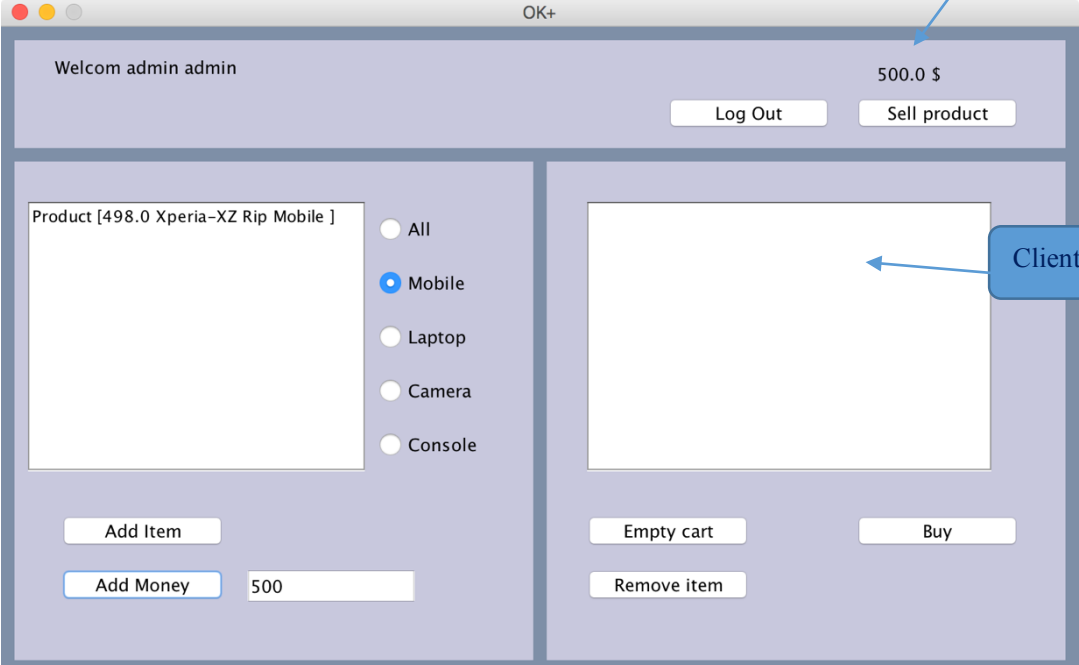
4.1.2   Sign Up Window

This window shows up when the user click on "Sign up" button from principal window.
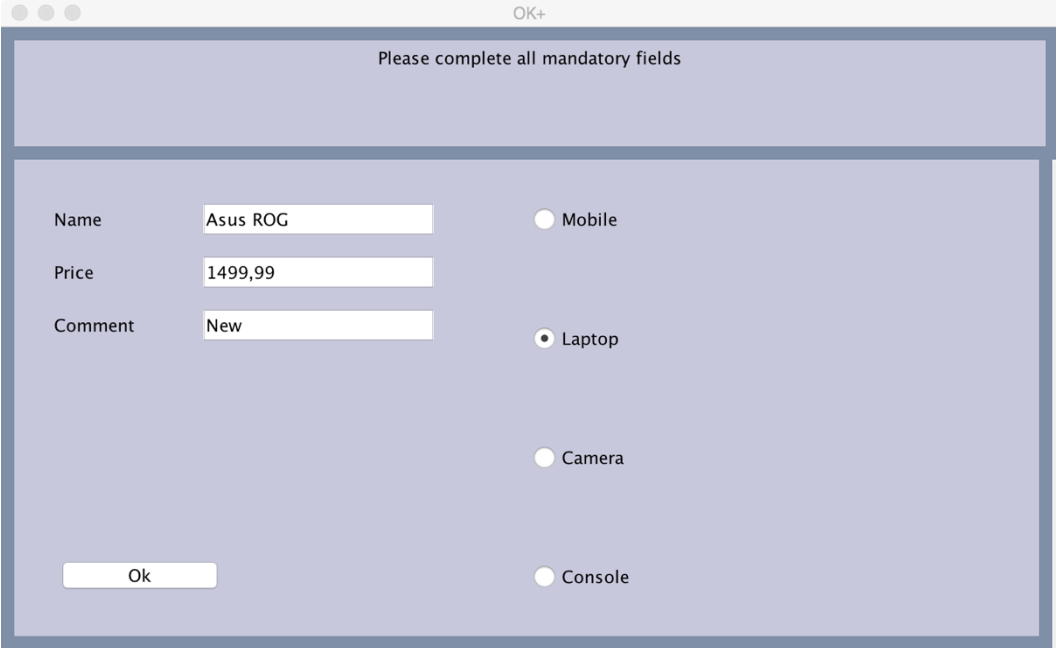

Figure 11 - Sign Up window

4.1.3 Client Window

This window shows up when a user log in the application. It is the user's session.

4.1.4 Sell Product Window

This window shows up when the user wants to add an item for sale.

4.2 Software Interfaces

4.2.1   Database

Database of the project is handle by file in the application. When the application is runned, it imports all data for users and products from different file. When the application is closed, it exports all users and products. The addition, modification, deletion or any operation done on a data is not done in real time in the files.

2 files are required for the application :
- Product.txt : products list
- Customer.txt : customers list

4.3 Communications Interfaces

4.3.1   Containers

Customer and product container are SINGLETON. Each time a product is added or a new user is created, it is added in the corresponding container. There is only one user container and one product container. All windows share the same user and product container.

4.3.2   Session

Each window which is related to user session has in parameter the user of the current session. We retrieve the user in the principal window when the user is logging in.

4.3.3   Factory Design pattern

In our application we choose to implement some kind of database in order to able some persistence for the data. We first implemented a file database, but in the future it was suppose to evolve and used a real Database, like Mysql or Sqlite. We decided to use the Design pattern strategy to implement the directory, so if we want to switch from file directory to database, it will be easier and more manageable.

4.3.4   Strategy Design pattern

We are using the design pattern strategy for display purpose. So, there is a display interface that is implemented by different classes. Indeed for each object there is an alternative implementation of the display method (for exemple to display a Customer or to display an Item). Then we can use the display we need.

5 Appendix A : Issues list

	#	Descriptions	Solutions
Interfaces	1	No image for product	Re-design the diplay of product list
	2	We can't exit Sign up window without sign up	Add "Cancel" button
	3	We can't exit Sell product window without sign up	
Data	1	Critical section problem	Use thread to synchronize data
	2	We can't update data in real time	Use database
	3	Storage issue	

6 Appendix B : Table of figures