

Universitat Oberta de Catalunya



**Degree Dissertation
Report
Asset Management**

Author: Josep Lluís Cardona Tegen
Advisor: David Gañán Jiménez
Degree: Enginyeria tècnica d'informàtica de gestió
Spring 2006

Summary

When starting this dissertation I had the problem to decide how to combine the task to learn the ASP.NET environment with a topic that will offer a supplement value. Since the degree contains as well the computer science lectures as economy ones, I made the decision to write about asset management.

Looking up term asset management in the wikipedia page the following definition will be found: “Asset management is the method that a company uses to track fixed assets, for example factory equipment, desks and chairs, computers, even buildings. Although the exact details of the task varies widely from company to company, asset management often includes tracking the physical location of assets, managing demand for scarce resources, and accounting tasks such as amortization. The most common usage of the phrase asset management is in terms of the financial services industry. Here it is used to describe the management of assets invested on behalf of a range of sectors including: collective investment schemes, pension funds and so-called private banking or wealth management (typically for wealthy individuals).”

In this case, the management of fixed assets is of less interest than the financial service, especially the private banking area.

The objective of the dissertation is not to offer the same services as commercial products, their implementation in a company can cause costs over millions of Euros, but to use the basic knowledge in this sector to create a simple application.

On the technical side, the challenge is to apply all the acquired knowledge of software engineering, making an application in the ASP.NET environment. The application additionally will use the Microsoft SQL server, MS Reporting Services and XML, so that the learning effect is not reduced to the main topic of the dissertation only.

Index

Introduction	1
1.1 Project Description	1
1.2 Target of the Dissertation	2
1.3 User Interface Requisites	2
1.4 Strategy	3
1.5 Planning	3
1.6 Obtained Results	4
2 Analysis	4
2.1 Functional Description	5
2.1.1 Authorisation Subsystem	6
2.1.2 User Management Subsystem	6
2.1.3 Asset Management Subsystem	6
2.1.4 Customer Subsystem	6
2.1.5 Batch Process Subsystem	6
2.1.6 Reporting Subsystem	6
2.2 Use Cases	7
2.2.1 Authorisation Subsystem	7
2.2.2 User Management Subsystem	9
2.2.3 Asset Management Subsystem	11
2.2.4 Customer Subsystem	12
2.2.5 Batch Process Subsystem	18
2.2.6 Reporting Subsystem	18
2.3 Sequence diagrams	19
3 Design	21
3.1 Architecture	22
3.2 Classes	23
3.2.1 Class Diagram	24
3.2.2 Class Methods	25
3.2.3 Controller Class Diagram	26
3.3 Database Design	27
3.3.1 Entity - Relationship Model	27
3.3.2 Database Diagram	29
3.3.3 Table Description	30
3.4 Controller Design	31
3.5 Flow of Windows	32
3.5.1 Authorisation Subsystem	32

3.5.2	User Management Subsystem	33
3.5.3	Asset Management Subsystem	35
3.6	User Interface Windows	39
3.6.1	Authorisation Subsystem	40
3.6.2	User Management Subsystem	43
3.6.3	Customer Subsystem	44
3.6.4	Batch process subsystem	52
4	Implementation	52
5	Conclusions and Recommendations	54
6	Annex	55
6.1	Bibliography	55
6.2	Internet References	55
6.3	Glossary	55

Figures

<i>Figure 0-1 Weekly Time Planning</i>	4
<i>Figure 2-1 Interaction between the different Subsystems</i>	5
<i>Figure 2-2 Subsystem Authorisation use case</i>	7
<i>Figure 2-3 Subsystem User Management Use Case</i>	9
<i>Figure 2-4 Subsystem Asset Management Use Case</i>	12
<i>Figure 2-5 Subsystem Customer Management Use Case</i>	13
<i>Figure 2-6 Subsystem Batch process Use Case</i>	18
<i>Figure 2-7 Subsystem Reporting Use Case</i>	19
<i>Figure 2-8 Sequence Diagram Add a New User</i>	20
<i>Figure 2-9 Sequence Diagram Buy an Asset</i>	20
<i>Figure 2-10 Sequence Diagram Delete an Asset</i>	21
<i>Figure 3-1 MVC Design Pattern</i>	22
<i>Figure 3-2 Static Class Diagram</i>	24
<i>Figure 3-3 Controller Class Diagram</i>	27
<i>Figure 3-4 Entity-relationship Model</i>	28
<i>Figure 3-5 Database Design</i>	29
<i>Figure 3-6 Flow Change Password</i>	32
<i>Figure 3-7 Flow Change Password Rules</i>	33
<i>Figure 3-8 Flow Insert New User</i>	33
<i>Figure 3-9 Flow Change User Data</i>	34
<i>Figure 3-10 Flow Delete an Existing User</i>	34
<i>Figure 3-11 Flow Insert a New Asset</i>	35
<i>Figure 3-12 Flow View an Existing Asset</i>	36
<i>Figure 3-13 Flow Add Asset Value</i>	37
<i>Figure 3-14 Flow Change Data of an Existing Asset</i>	38

<i>Figure 3-15 Flow Delete an Existing Asset</i>	<i>39</i>
<i>Figure 3-16 Window Welcome to the Asset Management</i>	<i>40</i>
<i>Figure 3-17 Window Welcome to the User Management</i>	<i>41</i>
<i>Figure 3-18 Window Change Own Password</i>	<i>42</i>
<i>Figure 3-19 Window Show and Change the Password Rules</i>	<i>42</i>
<i>Figure 3-20 Window Insert New User</i>	<i>43</i>
<i>Figure 3-21 Window Modify an Existing User</i>	<i>43</i>
<i>Figure 3-22 Window Delete an Existing User</i>	<i>44</i>
<i>Figure 3-23 Window Customer Management Submenu</i>	<i>44</i>
<i>Figure 3-24 Window Insert New Customer</i>	<i>45</i>
<i>Figure 3-25 Windows Modify an Existing Customer</i>	<i>46</i>
<i>Figure 3-26 Window Add an Asset to a Portfolio</i>	<i>47</i>
<i>Figure 3-27 Window Buy Asset</i>	<i>48</i>
<i>Figure 3-28 Window Remove Asset from Portfolio</i>	<i>49</i>
<i>Figure 3-29 Window Sell Asset</i>	<i>50</i>
<i>Figure 3-30 Window Delete an Existing Customer</i>	<i>51</i>
<i>Figure 3-31 Window Start a Batch Process</i>	<i>52</i>
<i>Figure 4-1 Implemented Database Diagram</i>	<i>53</i>

Introduction

The amount of information, the complexity of the different new assets and the necessity to increase the productivity, forces portfolio managers to use tools that reduce the work on all these topics. Each asset management department will increase the number of customers that are managed by each employee. Big companies spend millions of Euros developing solutions that try to reproduce the different internal workflows.

“Private and Institutional Banking are growth businesses whose impressive returns on investment are considerably higher than those of comparable banking sectors. Challenging markets, rising competition, eroding margins and increasing customer sophistication are some of the most important market trends that are impacting the way in which business is carried out. To keep ahead of the competition while maintaining high profit levels, bank and fund managers today need to provide a high level of personalised services, above average returns and competitive conditions to retain existing and attract new customers.”¹

The objective of the dissertation is the development of a software for asset management, also called portfolio management, together with the use of all the tools and techniques that are needed for the development process.

1.1 Project Description

The system will allow two different types of users, first the administrator that is responsible for the maintenance of the system (create users, modify their data and blocked and unblocked them). This user is not able to administer assets.

The other user type, the portfolio manager, will work with the application. This user can create different new assets with all the relevant information, e.g. the maturity date of a bond and its interest rate. The user can also create a customer, with all the necessary information for him. Finally the user can assign a list of assets to each customer. The asset manager can also increase or decrease the amount of securities and insert new ones or delete some of them.

Another task of the user is to insert the daily values of some assets, e.g. the share price.

The application is based in a client / server architecture. At the server site are the database server (stores all the information) and the web server (offers the interface to communicate). The client will be a usual web browser that will create the connection to the server and communicate with it.

¹ From the Triple'A® product overview

Additionally the system allows to import data from an XML file and create reports showing the information of a user.

1.2 Target of the Dissertation

The targets of this dissertation are the following:

- **Learn the ASP.NET Technology.**
Creation of an application using the ASP.NET Technology. The use of the .NET framework, especially the use of the ADO.NET class library for the relational database access.
- **Learn to work with the Microsoft SQL Server**
The use of the database tools to work with an ASP.NET application.
- **Learn to work with XML files**
The usage of different kinds of XML files and file definitions, their advantages and disadvantages. How to work from the application with these files.
- **Learn to use of Microsoft Reporting services**
The creation of reports and the integration in a application.
- **Use of the tools and methods for application developing**
The use of the different tools, methods, diagrams, ... that are necessary for the specification, analysis and implementation of an application.

1.3 User Interface Requisites

ASP.NET is purely a server-side technology. Unlike client-side scripting, ASP.NET code is executed on the server and not in the browser. The code that is sent back to the browser is pure HTML and not ASP.NET code.

All these properties have the consequence that the requirements on the client side are minimal.

The user will only need a web interface (it's recommended the use of the internet explorer) and a network connection to the application server.

The user requires some experience in the use of a web browser.

1.4 Strategy

For the development of the application the methods of the software engineering will be used. The life cycle of such a project consists of four important phases.

The first one is the analysis (in chapter 2 Analysis) where the specification is made, considering only the functional part. The technical background is not regarded. Here the team has to bring in a cold print of all their knowledge. The more detailed the specification is made the less problems and delay will occur in the other phases. The computer specialist will use UML tools, among other things, to describe the complete model. This helps to reduce ambiguity in the definition of the system wanted by the customer. The developed model can be applied to all development environments, not only the one used in this project.

The second phase is the design (in chapter 3 Design). Once the analysis is done, the theoretical model is translated into the environment that will be used. Here a deep technical knowledge of the environment is decisive. The result is an adjusted solution for the selected environment.

Third one is the implementation (in chapter 4 Implementation). Where the design is converted into code.

The test step is when persons with functional knowledge check if the application does what it supposed to do according to the analysis model.

In this project less important, due to the shortage of time, but in a production environment more and more important are the step maintenance. Maintenance is the step when the bugs in the system will be managed and solved.

1.5 Planning

The planning of this project was conditioned by the delivery deadlines.

Description	13/3/2006
Design	18/4/2006
Implementation	22/5/2006
Final delivery	19/6/2006

Simultaneously to the analysis it was necessary to acquire the technical knowledge for the development environment, so that at the start of the design the basics for MS SQL Server and from the Visual Studio .NET were available.

In the design phase it was necessary to get a deeper knowledge of the C# programming. Otherwise it would have been more difficult to realise the implementation in the next step.

Finally in the implementation period the test and maintenance tasks were included to guarantee a top-quality result.

The weekly planning for the project is displayed in the next diagram. The blue marked areas are the times reserved for the project work.

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
8 ⁰⁰							
9 ⁰⁰							
10 ⁰⁰							
11 ⁰⁰							
12 ⁰⁰							
13 ⁰⁰							
14 ⁰⁰							
15 ⁰⁰							
16 ⁰⁰							
17 ⁰⁰							
18 ⁰⁰							
19 ⁰⁰							
20 ⁰⁰							
21 ⁰⁰							
22 ⁰⁰							

Figure 0-1 Weekly Time Planning

1.6 Obtained Results

The result of the dissertation is not only this document. Here a list of the components

- A database, which can be created with the createPortfolioManagement.sql file.
- An ASP.NET application.
- An executable file for batch processes.
- A report file.
- The XML files for the configuration and the addition of asset values in the database, with their validation files.
- The administrator and user handbook.
- The documentation about the created classes.

2 Analysis

The first step for the project is to define what the customer wants. This can be the origin of a lot of problems and misunderstandings between the customer and the computer specialist.

Normally the customer has no knowledge of IT, and the computer specialist no specialised knowledge in the area of the customer.

To avoid future problems there are different tools and techniques that allow to find a common language between the customer and the computer specialist. Currently the diagrams of the unified modelling language (UML) are the most popular ones, but others exist as well.

Due to the lack of time, only a selection of techniques was used. These are the most descriptive and important ones.

2.1 Functional Description

This functional description is an approximation to the complete description of the functionality. This one will be extended with other techniques, e.g. the use cases.

To make the functional background of the application more comprehensible, the system is divided into independent subsystems. Each of them will be easier to handle.

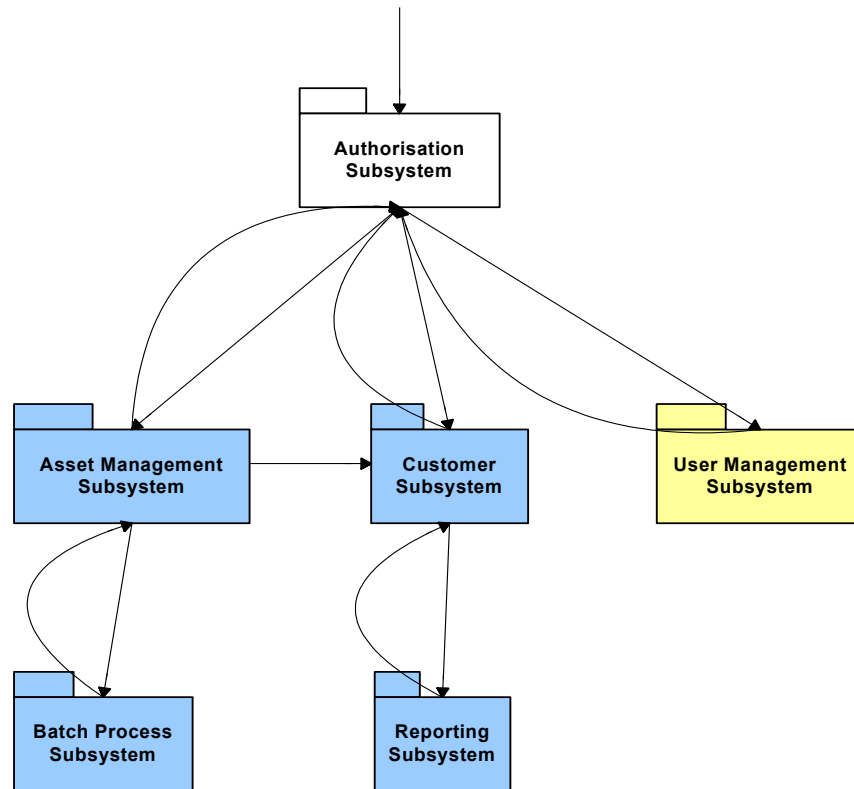


Figure 2-1 Interaction between the different Subsystems

In the following, a brief description of each subsystem is given.

2.1.1 Authorisation Subsystem

This subsystem will allow the user to connect to the application. The system will prompt for user code and password. The administrator will only see the pages for the creation and administration of the other users. All other users will see the pages for the asset management only (asset management subsystem and customer subsystem).

2.1.2 User Management Subsystem

This system allows the user admin to insert, modify or delete users. The admin user can't be modified (except the password).

2.1.3 Asset Management Subsystem

The user can create and modify (not delete) the assets of certain types. The deletion is not allowed because there must be a possibility to follow the history of the customer. The modified data will be saved in a history table. The re-creation of a deleted item will mean the creation of a new one, with a new identifier, and not the modification of the delete status of the first one. The supported assets are shares, investment funds, current accounts and bonds.

2.1.4 Customer Subsystem

The asset manager will be able to create and modify the data of the customers. The user will assign the assets to the customer portfolio here. The buy/sell assets and the add/remove of assets to a portfolio is also included in this subsystem. As in the case of the asset data all modifications will be stored for the history of the customer.

2.1.5 Batch Process Subsystem

On the server it will be possible to start a batch program reading data from a text file (with the daily values of the different assets) and inserting it into the database. This process will be started from the application, so that the authorisation is warranted over the log in process in the application.

2.1.6 Reporting Subsystem

It will be possible to create reports with the complete data of a customer. These reports can be printed and presented to the customer

2.2 Use Cases

The necessity to describe the functionality more accurately than possible in plain text, forces to use another techniques. One possible technique are Use Cases, which are part of the graphical language of UML.

“A use case describes a set of sequences, in which each sequence represents the interaction of the thing outside the system (its actors) with the system itself (and its key abstractions). These behaviours are in effect system-level functions that you used to visualize, specify, construct and document the intended behaviour of your system during requirements capture and analysis.”²

For the lack of space the use cases description for the asset management subsystem won't be shown.

2.2.1 Authorisation Subsystem

During the installation only the user admin is created with a default password. Each user has the capability to change his own password. The admin can change the different password rules. There is no authorisation at the function level.

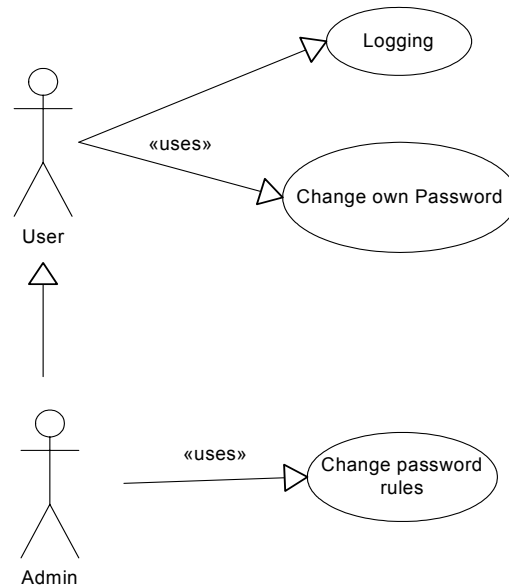


Figure 2-2 Subsystem Authorisation use case

² “The unified modeling language user guide “, written by Grady Booch, James Rumbaugh and Ivar Jacobson. Addison Wesley February 2004

Use Case: Login	
Target	Allow or deny a user to connect to the system
Role in the Work of the User	Mandatory for each user to connect with the software, without a successful log in it is not possible to continue working
Actors	Each user trying to work with the software
Related Use Cases	
Pre Condition	None
Post Condition	
Description	<p>The user can work with the software. If the user is admin he can only execute the user management tasks, otherwise he can work in the asset management subsystem and the customer subsystem</p> <ol style="list-style-type: none"> 1 Insert user code and password 2 Click the button ok 3 Confirmation of user name and password with the values stored in the database and access to the application
Exceptions	<ol style="list-style-type: none"> 1 If there is no successfully confirmation of user name and password, give an error message and go back to the log in page - after 3 intents the user name (if existing) will be blocked. 2 If there is a problem to access the database, show a message

Use Case: Change Own Password	
Target	Change the user's own password
Role in the Work of the User	Each user can change his login password to the application
Actors	All users
Related Use Cases	
Pre Condition	To be successfully connected
Post Condition	The new password is stored in the database
Description	<ol style="list-style-type: none"> 1 Insert the old password 2 Insert the new password twice 3 Confirmation of user name and password and storage of the new password in the database
Exceptions	<ol style="list-style-type: none"> 1 If there is no confirmation of user name and password, give an error message and go back to the first window 2 If the new password doesn't observe the rules for passwords show a message and go back to the <i>Change Own Password</i> window (the rules are stored in the database and can be changed by the admin) 3 If there is a problem to access the database, show a message

Use Case: Change Password Rules	
Target	The admin can change the rules for the passwords, these the user has to take into account when changing it
Role in the Work of the User	The admin can change the default values of the rules for security reasons
Actors	Only the admin user
Related Use Cases	
Pre Condition	To be successfully connected as admin
Post Condition	The new rules are stored in the database
Description	<ol style="list-style-type: none"> 1 Show all current values 2 Change one or more values (the values are: maximal and minimal length of the password, the requirement to use at least one non letter character) 3 Save the changes with the save button to the database
Exceptions	<ol style="list-style-type: none"> 1 If there is a problem to access the database, show a message

2.2.2 User Management Subsystem

The admin can create new users, modify existing users or delete users that are no longer needed. The user will only be logically deleted, otherwise it wouldn't be possible to follow the history. It will not be possible to add the admin rights to another user. The administrator can enable and disable access to the system for existing users

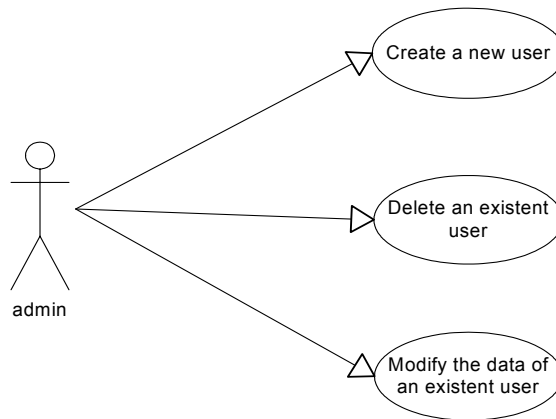


Figure 2-3 Subsystem User Management Use Case

Use Case: Create a New User	
Target	Create a new user account
Role in the Work of the User	The admin user will create a new account to allow a user to work with the application
Actors	Only the admin user
Related Use Cases	
Pre Condition	To be successfully connected as admin
Post Condition	The data for the new user is stored in the database
Description	<ol style="list-style-type: none"> 1 Insert the name, surname of the user, the assigned user code and a password 2 Test if the user code already exists 3 Click the save button 4 Save the data in the database
Exceptions	<ol style="list-style-type: none"> 1 If the user code already exist show a message and return to the insert new user window 2 If there is a problem to access the database, show a message

Use Case: Delete an Existing User	
Target	Delete a user account and log the action
Role in the Work of the User	The admin deletes a user account
Actors	Only the admin user
Related Use Cases	
Pre Condition	To be successfully connected as admin
Post Condition	The data is deleted from the database
Description	<ol style="list-style-type: none"> 1 Select a user from the list (user code field) 2 Click the delete button 3 Set the delete flag for the user to true 4 Log the deletion
Exceptions	<ol style="list-style-type: none"> 1 If there is a problem to access the database, show a message

Use Case: Modify an Existing User	
Target	Save modified data and log in database
Role in the Work of the User	The admin modifies the data referring to a user
Actors	Only the admin user
Related Use Cases	
Pre Condition	To be successfully connected as admin
Post Condition	The modified data is stored in the database
Description	<ol style="list-style-type: none"> 1 Select a user from the list 2 The data stored in the database will be read and presented in the window 3 The user modifies one or more of the following fields: name, surname, user code, password or blocked. 4 Click the save button 5 If the user code is changed, test if it already exists 6 Save the data in the database 7 Take record of the changes
Exceptions	<ol style="list-style-type: none"> 1 If the user code already exists, show a message and return to the <i>Modify User</i> window 2 If there is a problem to access the database, show a message

2.2.3 Asset Management Subsystem

One of the principal tasks of the user is to work with the different kinds of securities. He has to be able to create new assets, modify them, and at last delete them.

The user will use different types of identifications for the assets, e.g. the ISIN code, the Reuters code, the Bloomberg code, the IBAN code, etc. The system will control that the functional identification code is unique.

When the user creates a new asset he will be forced to fill some mandatory fields. Others are optional. Which fields exist depends on the type of asset.

After saving the data, the user will be able to change the assets. The way is to locate the asset, view it and then modify it. All changes will be stored, so that it is possible to follow which user made which change, and at what time.

The removal of assets is really simple, the system checks if the asset is assigned to a portfolio, if not the delete flag will be activated. The user will not be allowed to erase the data from the system to be able to view a history of all customer data. These tasks will also be tracked.

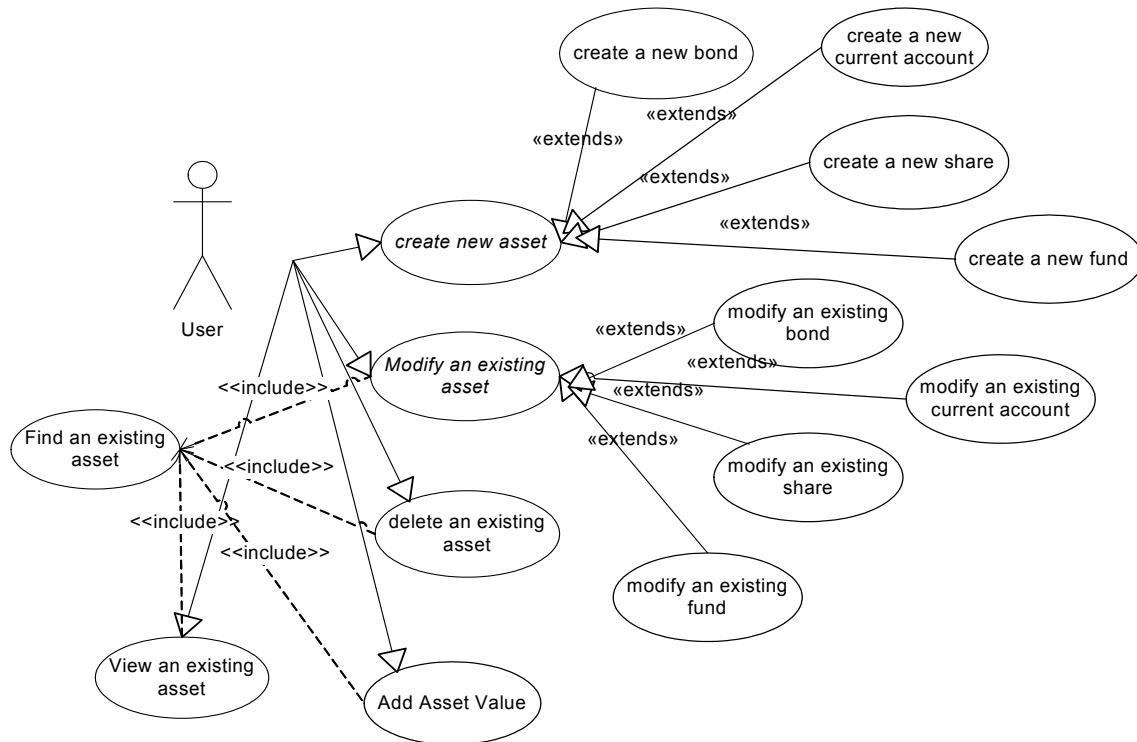


Figure 2-4 Subsystem Asset Management Use Case

2.2.4 Customer Subsystem

The principal objective of the software is to simplify the work with the customers. Therefore the different functionalities to manipulate the customer information are necessary.

The first step is to create a customer. As in the case of asset creation the system will generate an internal code to identify the customer. Some fields will be mandatory like name, surname and address and others will be optional like telephone number. Only natural persons are allowed as customers. For each customer it is mandatory to have a current account, without one it is not possible to assign other assets.

The user can also modify the customer information. These changes will be logged in the database.

The delete function will only activate a delete flag because it must be possible to track the history in the future. It is not possible to delete a customer having assets in his portfolio.

The most important task is the capability to assign assets to the customer. Where assign means an initial transfer from an existing account or a buy of the asset. In the second case the balance of the current account of the user will be reduced by the amount of “number of assets” * “price of asset”.

The user can sell an asset from a customers portfolio. The amount of “number of asset” * “price of the asset” will be added to the current account. The option remove an asset means that no changes in the current account will be done. It will be possible to sell or remove a part of the asset – e.g. if a customer wants to sell the half of his shares of company A.

Each of the actions add/buy or remove/sell will get an internal number (transaction number) that will be stored in the database together with the information about this transaction.

For the asset type current account it is not possible to buy or to sell (it doesn't make sense). Only the operations add and remove are allowed.

If the system has no daily price for the asset, a price can be inserted along with each transaction. This price will be stored in the database as asset value for the transaction day (not applicable for current accounts and bonds).

There is no possibility to modify the assets assigned to a portfolio. (The user can only buy/add or sell/remove)

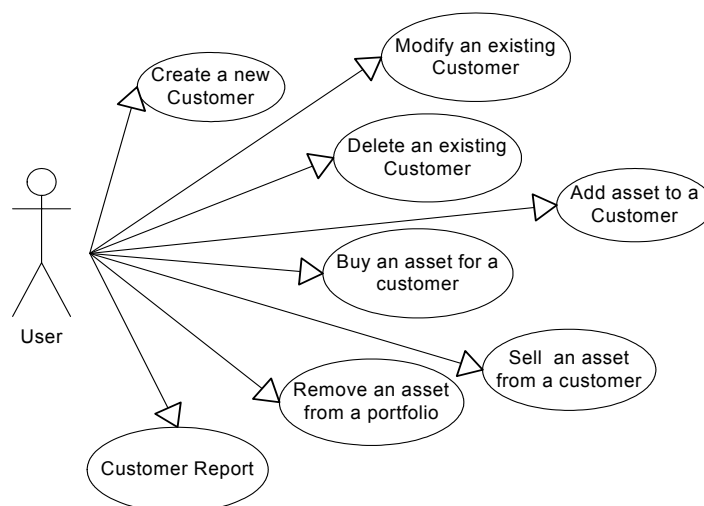


Figure 2-5 Subsystem Customer Management Use Case

Use Case: Create a New Customer	
Target	Create a new customer
Role in the Work of the User	The user works with customers that have to be created
Actors	All users except admin
Related Use Cases	
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The system calls the specific use case for the asset
Description	<ol style="list-style-type: none"> 1 The system generates an internal code for the customer 2 The user has to insert values for the mandatory fields (these fields are marked with an asterisk) 3 The user can insert values for all other fields 4 Save the data in the database
Exceptions	<ol style="list-style-type: none"> 1 If the user didn't fill the mandatory fields, show a message and return to the window 2 If the identity card number or passport number already exist, show an error message and return to the previous page 3 If there is a problem to access the database, show a message

Use Case: Modify an Existing Customer	
Target	Modify a customer account
Role in the Work of the User	Change some data for a customer
Actors	All users except admin
Related Use Cases	
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The modified data is saved in the database
Description	<ol style="list-style-type: none"> 1 The user selects a customer 2 The system accesses the database, reads the customer information and presents it in the window 3 The user changes the desired fields 4 Save the data in the database 5 Log the changes
Exceptions	<ol style="list-style-type: none"> 1 If there is a problem to access the database, show a message

Use Case: Delete an Existing Customer	
Target	Delete a customer account
Role in the Work of the User	To remove a customer from the system
Actors	All users except admin
Related Use Cases	
Pre Condition	To be successfully connected as user (except admin)
Post Condition	The modified data is saved in the database
Description	<ol style="list-style-type: none"> 1 Check if there are assets assigned to the customer. 2 Activate the delete flag of the customer 3 Save the modification in the database
Exceptions	<ol style="list-style-type: none"> 1 If there are assets assigned to the customer, show a message and return to the window 2 If there is a problem to access the database, show a message

Use Case: Add Asset to a Portfolio	
Target	Add an asset to a customer portfolio
Role in the Work of the User	The user will add an asset quantity to an existing portfolio
Actors	All users except admin
Related Use Cases	1 Find an existing asset
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The data is stored in the database
Description	<ol style="list-style-type: none"> 1 The user selects an asset code 2 The user selects a customer 3 The system inserts the current date in the date field 4 The user fills the rest of fields 5 The user saves the transaction 6 The system checks if the asset code and the customer code exists 7 The data is stored into the database 8 The system takes the record of the transaction
Exceptions	<ol style="list-style-type: none"> 1 If either the asset code or the customer code is not found, show a message and return to the <i>Add an Asset to a Portfolio</i> window 2 If there is a problem to access to the database, show a message

Use Case: Buy an Asset	
Target	Buy an asset for a customer
Role in the Work of the User	The user will buy an asset with the resource existing in the current account
Actors	All users except admin
Related Use Cases	1 Find an existing asset
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The data is stored in the database
Description	<ol style="list-style-type: none"> 1 The user selects an asset code (only investment funds, bonds and shares) 2 The user selects a customer 3 If the customer has only a current account it will be shown automatically, otherwise the user has to select one 4 The system inserts the actual date in the date field 5 The user fills the rest of fields 6 The user saves the transaction 7 The system checks if the asset code and the customer code exists 8 The data is stored into the database 9 The system calculates the price of the transaction and reduces the amount from the current account 10 The system records the transaction
Exceptions	<ol style="list-style-type: none"> 1 If there is no asset found, show a message and return to the <i>Buy an Asset</i> window 2 If there is a problem to access the database, show a message

Use Case: Remove Asset from a Portfolio	
Target	Remove an asset from a customer portfolio
Role in the Work of the User	The user will remove an asset quantity from an existing portfolio
Actors	All users except admin
Related Use Cases	1 Find an existing asset
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The data is stored in the database
Description	<ol style="list-style-type: none"> 1 The user selects an asset code 2 The user selects a customer 3 The system inserts the current date in the date field 4 The user fills the rest of fields 5 The user saves the transaction 6 The system checks if the asset code and the customer code exist 7 The system checks if the quantity of assets to be removed is less than or equal to the existing quantity of the portfolio 8 The data is stored into the database 9 The system records the transaction
Exceptions	<ol style="list-style-type: none"> 1 If either the asset code or the customer code are not found, show a message and return to the <i>Remove Asset from a Portfolio</i> window 2 If the quantity to be removed is bigger than the existing quantity, show a message and return to the <i>Remove Asset from a Portfolio</i> window 3 If there is a problem to access the database, show a message

Use Case: Sell an Asset	
Target	Sell asset from a customer portfolio
Role in the Work of the User	The user will sell an asset and add the value to the current account of the customer
Actors	All users except admin
Related Use Cases	1 Find an existing asset
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The data is stored in the database
Description	<ol style="list-style-type: none"> 1 The user selects an asset code (only investment funds, bonds and shares) 2 The user selects a customer 3 If the customer has only one current account it will be shown automatically, otherwise the user has to select one 4 The system inserts the current date in the date field 5 The user fills the rest of fields 6 The user saves the transaction 7 The system checks if the asset code and the customer code exists 8 The system checks if the quantity of assets to be sold is less than or equal to the existing quantity of the portfolio 9 The data is stored in the database 10 The system calculates the price of the transaction and increment the amount on the current account 11 The system records the transaction
Exceptions	<ol style="list-style-type: none"> 1 If either the asset code or the customer code are not found, show a message and return to the <i>Sell an asset</i> window 2 If the quantity to be removed is bigger than the existing quantity, show a message an return to the <i>Sell an asset</i> window 3 If there is a problem to access the database, show a message

Use Case: View Data of an Existing Asset	
Target	View the data of an existing asset
Role in the Work of the User	A user can ask for the asset data stored in the database
Actors	All users except admin
Related Use Cases	1 Find an existing asset
Pre Condition	To be successfully connected as a user (except admin)
Post Condition	The data is presented in a window
Description	<ol style="list-style-type: none"> 1 The user has to insert the asset code 2 The user clicks the view button 3 The system searches for the data in the database
Exceptions	<ol style="list-style-type: none"> 1 If there is no asset found, show a message and return to the <i>View Data of an Existing Asset</i> window 2 If there is a problem to access the database, show a message

2.2.5 Batch Process Subsystem

It's a very repetitive and time consuming task to keep the information of each asset in the system up to date. It is important to have a price for each asset stored every day. To select an asset manually and to save the current price is very slow and will produce a lot of errors.

The information about each asset will normally come from a backend system. The idea of this subsystem is to provide a tool that permits the transfer of this information from one system to the other. For this purpose a file format in XML is defined and a process to insert the data from the file into the database without the necessity of anyone to carry out the process manually.

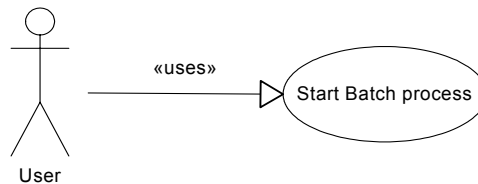


Figure 2-6 Subsystem Batch process Use Case

Use Case: Start Batch Process	
Target	Start Batch Process
Role in the Work of the User	The user wants to insert the current values of the assets for a day from a file into the database
Actors	All users except the admin
Related Use Cases	
Pre Condition	To be successfully connected as user (except admin)
Post Condition	The new data is stored in the database
Description	<ol style="list-style-type: none"> 1 Insert the path and file name 2 Click the start button 3 Check the data of the file 4 Save the data in the database
Exceptions	<ol style="list-style-type: none"> 1 If the check of an item in the file produces an error and generate an entry in a log file 2 If there is a problem to access the database, show a message

2.2.6 Reporting Subsystem

The best way to present the data of a customer is to use a reporting tool. A problem with an application on a browser is how to get a print of the data that can be presented to a client in a professional way. With this tool we will access the database, read the data of the customer, calculate the results and generate a report that can be printed.

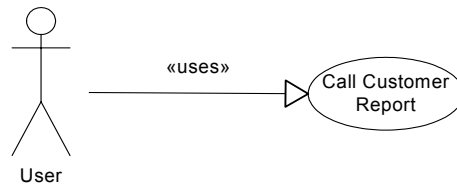


Figure 2-7 Subsystem Reporting Use Case

Use Case: Call Customer Report	
Target	Call a customer report
Role in the Work of the User	The user wants to present the data of a customer in a format to be printed
Actors	All users except the admin
Related Use Cases	
Pre Condition	To be successfully connected as user (except admin)
Post Condition	A report is generated
Description	<ol style="list-style-type: none"> 1 Select the user 2 Click the call report button 3 Call the reporting tool on the database 4 Generate the report
Exceptions	<ol style="list-style-type: none"> 1 If there is a problem to access the database, show a message

2.3 Sequence diagrams

“A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows a set of objects and the messages send and received by those objects. The objects are typically named or anonymous instances of classes, but may also represent instances of other things, such as collaborations, components, and nodes. You use sequence diagrams to illustrate the dynamic view of a system.”³

As in the other chapters, only a selection of sequence diagrams is presented here, the complete list of them would exceed the maximal length for this report.

³ “The unified modeling language user guide “, written by Grady Booch, James Rumbaugh and Ivar Jacobson. Addison Wesley February 2004

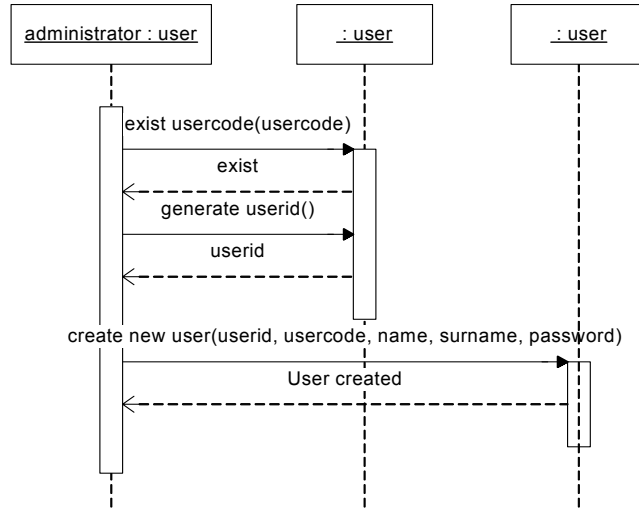


Figure 2-8 Sequence Diagram Add a New User

In this sequence diagram we can see the different steps that the program will follow for the creation of a new user account. First it must be verified, if the usercode exists. Next a new userid is generated and finally the new user is created.

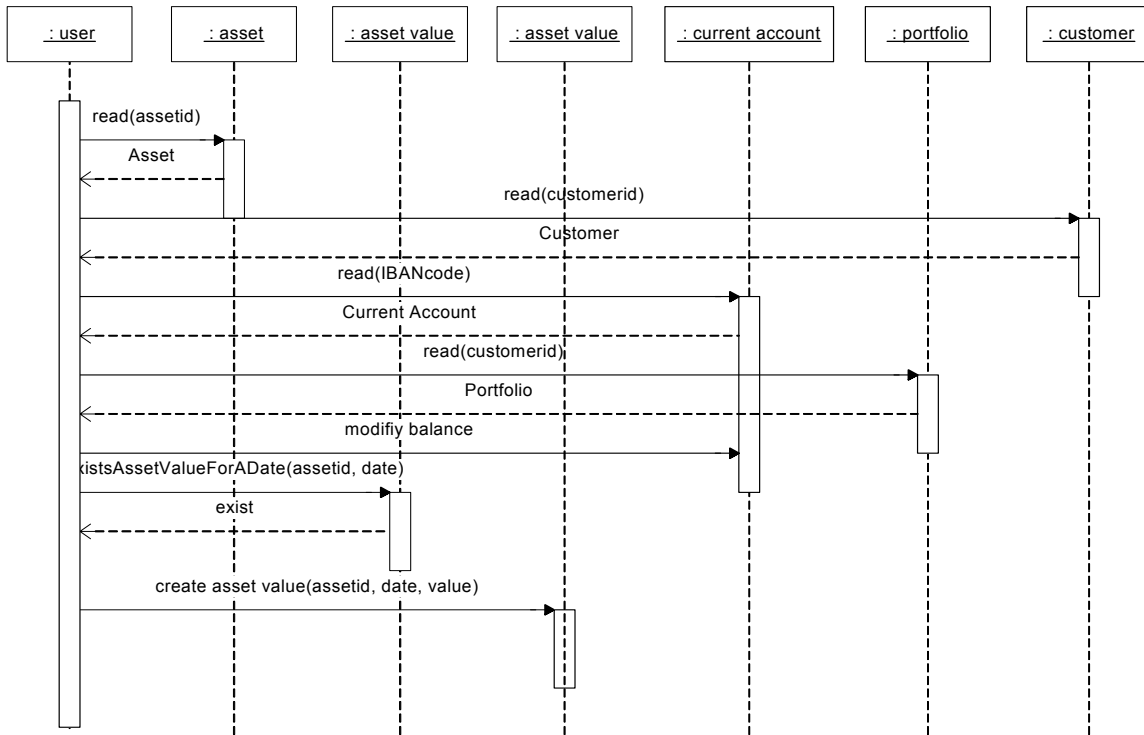


Figure 2-9 Sequence Diagram Buy an Asset

The first steps are to read the information about the asset, the customer, the current account, and the portfolio. When all the information is available the value of the balance in the current account will be modified. The existence of an asset value for the day and the asset id will be checked, if not available a new asset value will be created.

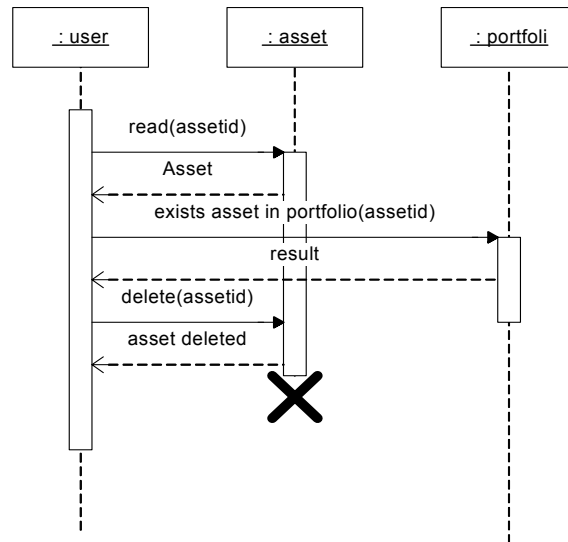


Figure 2-10 Sequence Diagram Delete an Asset

For deleting an asset, the asset information will be read. Afterwards it will be checked if the asset is included in at least one customer portfolio. The last step is to delete the asset object.

3 Design

Once an analysis of the project exists, the developer has to take the decisions how to transfer all the concepts from the analysis in the environment where the application is to be implemented.

First he will use a general model and afterwards he will adapt it to the development environment and its restrictions. That requires a deep knowledge of the system and it's a project phase where the customer is not present.

Due to the use of an object oriented programming language (in this case C# will be used) the class design is essential.

Not less important is the design of the database schema the application will use. The database used is Microsoft SQL Server.

3.1 Architecture

The model used to implement this application is the MVC. Model-View-Controller (MVC) is a classic design pattern often used by applications that need the ability to maintain multiple views on the same data. The MVC pattern hinges on a clean separation of objects into one of three categories — models for maintaining data, views for displaying all or a portion of the data, and controllers for handling events that affect the model or view(s).

Because of this separation, multiple views and controllers can interface with the same model. Even new types of views and controllers that never existed before can interface with a model without forcing a change in the model design.

The MVC abstraction can be graphically represented as follows:

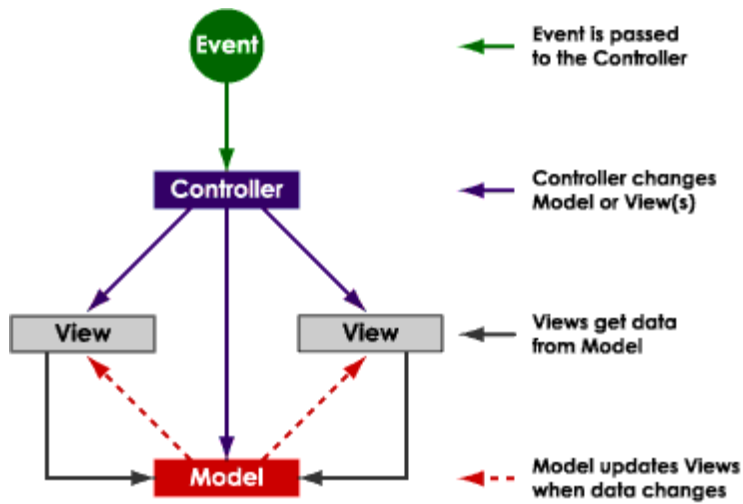


Figure 3-1 MVC Design Pattern

Events typically cause a controller to change a model, or view, or both. Whenever a controller changes a model's data or properties, all dependent views are automatically updated. Similarly, whenever a controller changes a view, for example, by revealing areas that were previously hidden, the view gets data from the underlying model to refresh itself.

Following are a few of the benefits of MVC design pattern.

- Since MVC handles the multiple views using the same enterprise model it is easier to maintain, test and upgrade the multiple system.
- It is easier to add new clients just by adding their views and controllers.

- Since the model is completely decoupled from the views, it allows lot of flexibilities to design and implement the model considering reusability and modularity. This model can also be extended for further distributed applications.
- It is possible to have development processes in parallel for model, view and controller.
- This makes the application extensible and scalable.

Microsoft in the document Implementing Model-View-Controller in ASP.NET describes the following benefits

- **Reduced dependencies.** An ASP.NET page allows the programmer to implement methods within a page. As the Single ASP.NET Page shows, this can be useful for prototypes and small short-lived Web applications. As the complexity of the page, or the need to share code between pages, increases, it becomes more useful to separate portions of the code.
- **Reduced code duplication.** The **GetRecordings** and **GetTracks** methods in the **DatabaseGateway** class can now be used by other pages. This eliminates the need to copy the methods into multiple views.
- **Separation of duties and concerns.** The skill set for modifying the ASP.NET pages is different from the skill set for writing code that accesses the database. Separating the view and the model, as shown earlier, allows specialists in each area to work in parallel.
- **Optimizing opportunities.** Separating the responsibilities into specific classes, as shown earlier, increases the opportunities for optimization. In the example described previously, the data is loaded from the database every time a request is made. It would be possible to cache the data in certain situations, which could improve the overall performance of the application. This, however, would be difficult or impossible without separating the code.
- **Testability.** Isolating the model from the view makes it possible to test the model outside the ASP.NET environment.

3.2 Classes

The work with an object oriented environment implies the use of classes. *Objects* are simply container for variables and functions that logically go together. In object-oriented parlance, variables and functions inside an object are called *properties* and *methods*.

The definition of the objects and their properties is done within a class diagram. The definition of the methods is done with a list of the parameters and a short description of how it works.

Furthermore to the classes required for application logic, classes associated to the Model View design are needed, where controllers have an associated class.

3.2.1 Class Diagram

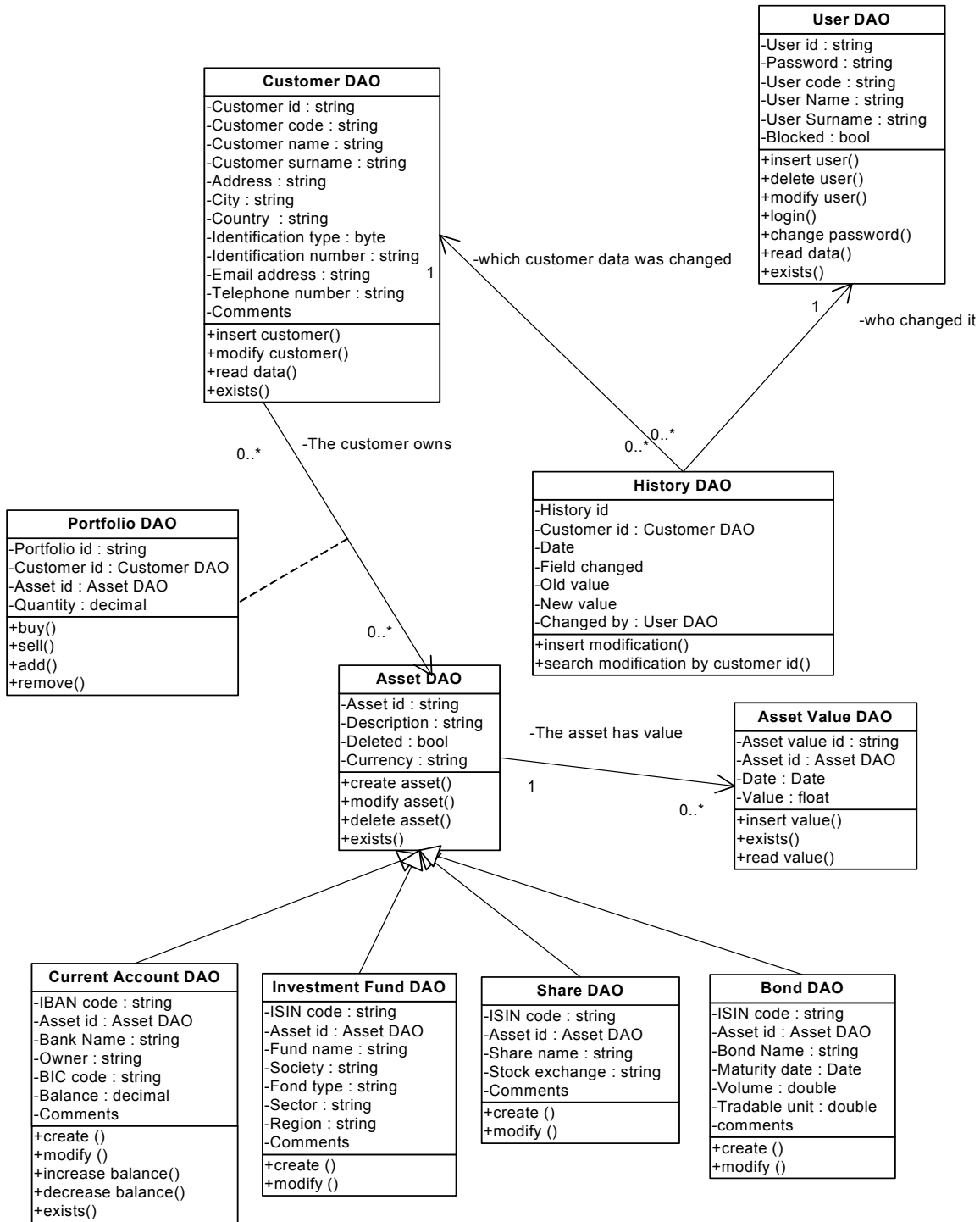


Figure 3-2 Static Class Diagram

3.2.2 Class Methods⁴

Customer Class

Insert customer(String customerCode; String customerName; String customerSurname; String address; String city; String country; Byte identificationType; String identificationNumber; String emailAddress; String telephoneNumber) – The system generates a customerId (it's a string beginning by a C and followed with a 6 digit number). Then it checks if the assigned customerCode already exists. And at last but not least stores data in the database.

Modify customer(String customerId; String field; String newValue) – The system probes if the customerId exists, and changes the value of the modified field. It calls the function insert modification() from the history class.

Read data(String customerId) – Returns all the stored data of that customer

Exists(String customerId) – Tests if the customerId already exists and returns a boolean value.

User Class

Insert user(String userCode; String password; String username; String userSurname) – The system generates a userId (it's a string beginning with a U and following a digit number). It checks if the userCode is already used and that the password follows the password rules. Afterwards the data is stored in the database. Returning the boolean value true if the data is correctly stored, otherwise returning false.

Modify user(String userCode; String field; String newValue) – The system saves the field containing the new value in the database.

Delete user(String userCode) – The system saves the value true in the blocked field

Login(String userCode, String password) – The system reads the data stored in the database and compares the values. A Boolean value is returned.

Change password(String userCode; String oldPassword; String newPassword) – The system calls the function login with the userCode and the oldPassword if it returns true the password field is filled with the newPassword value of the user in the database.

⁴ Only a selection of classes will be displayed, otherwise the length of the report would exceed the maximal size

Read data(String userId) – Returns all the stored data of that user.

Exists(String userId) – Checks if the userId already exists and returns a Boolean value as the result.

Portfolio Class

Buy(String assetId; String customerId; Decimal quantity; String currentAccount) – First test if the assetId, the customerId, and the currentAccount exists. Next test if the currentAccount belongs to the customerId. Calculate the value of the buy. Reduce the balance of the current account and store the data into the database. If the function reduceBalance returns an error code (the balance is less than the amount to draw) no changes will take place. Finally if there is an entry for this customer and this asset in the portfolio table, the field quantity will be increased, otherwise a new entry will be created. This function is not applicable to current accounts.

Sell(String assetId; String customerId; Decimal quantity; String currentAccount) – As in the buy function all the parameters will be probed. Additionally the system will test if the quantity of the asset to be sold is less or equal to the existing. In case that user sells the same quantity as he owns the quantity field in the table portfolio will be zero, the entry won't be deleted. Same as the buy function this is not applicable to current accounts.

Add(String assetId; String customerId; Decimal quantity) – This function differs from the buy function in the respect that the balance in the current account will not be changed.

Remove(String assetId; String customerId; Decimal quantity) – The same business logic is used as in the sell function with the difference that the balance of the current account is not increased.

3.2.3 Controller Class Diagram

For each ASP.Net window a controller will exist. The task of the controller is to get petition from the ASP.Net information and give the events to the model which will connect to the database and get all the necessary information. The Microsoft Visual Studio .Net provides the tools to generate a code-behind file with the extension .cs (for C#) .

All controllers have 3 common methods: Page_Load (called at the initialization of the window, so that it is possible to get the information for the different fields or change the configuration of the page), OnInit (called at the initialization of the page too, calls the InitializeComponent method and the OnInit method from the parent class) and the InitializeComponent (necessary for the use of a web formulary).

All controller classes inherit from the class System.Web.UI.Page.

The following diagram provides an overview of selected controller classes (the complete diagram with all classes would be too confusing)

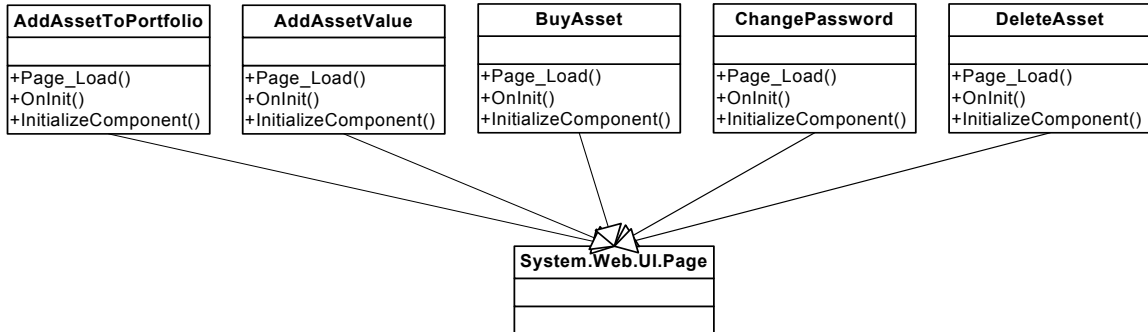


Figure 3-3 Controller Class Diagram

3.3 Database Design

3.3.1 Entity - Relationship Model

“The entity-relationship model (ER model) is a way of graphically representing the logical relationships of entities (or objects) in order to create a database.

In ER modelling, the structure of a database is portrayed as a diagram, called an entity-relationship diagram (ER diagram), which resembles the graphical breakdown of a sentence into its grammatical parts. Entities are rendered as points, polygons, circles, or ovals. Relationships are portrayed as lines connecting the points, polygons, circles, or ovals. Any ER diagram has an equivalent relational table, and any relational table has an equivalent ER diagram. ER diagramming is an invaluable aid to engineers in the design, optimization, and debugging of database programs.”⁵

⁵ Definition from the site whatis.com

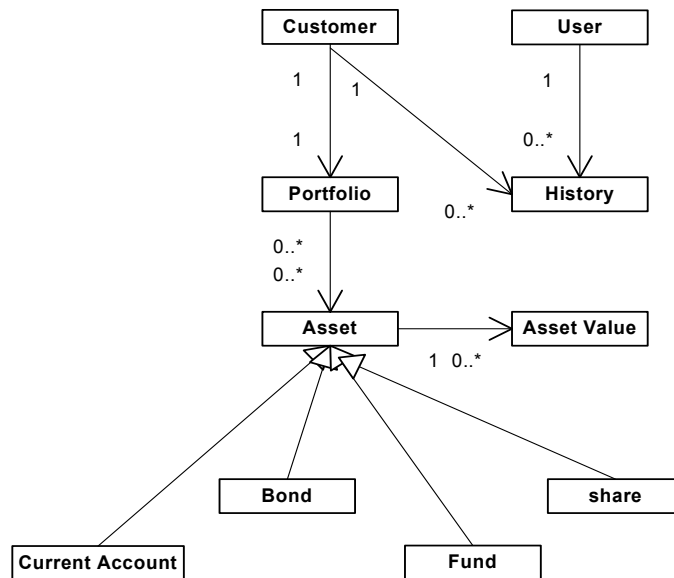


Figure 3-4 Entity-relationship Model

Here all the entities are identified with their relationships in our system.

Remarkable is the relationship between the entity asset and the entities current account, bond, fund and share. Since all the asset types have a common handling it's advisable to create the entity asset that contains the common attributes and the asset types inherit it. Another advantage is that the relationship to the other entities is more clear as when each asset type will have its relationship.

3.3.2 Database Diagram

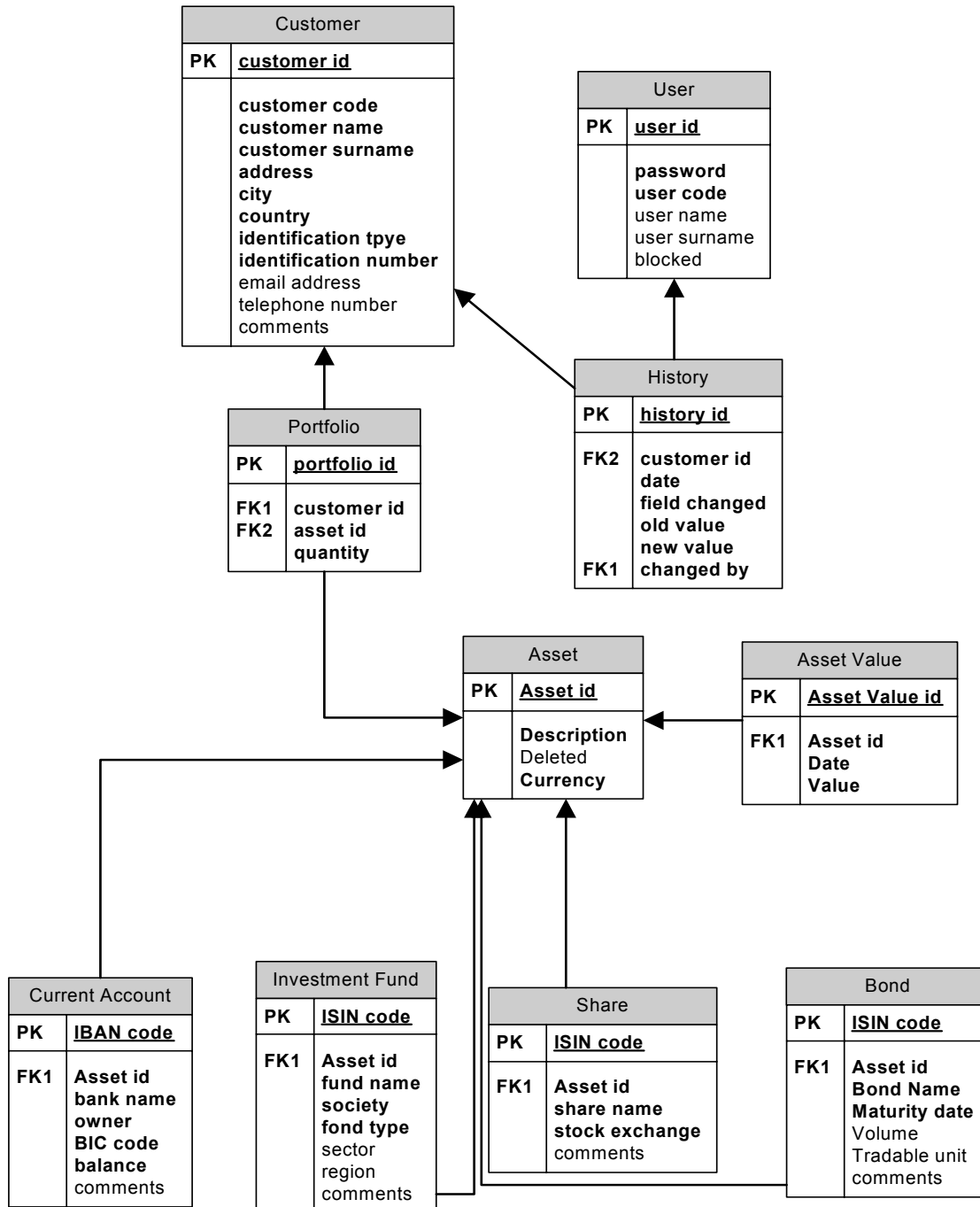


Figure 3-5 Database Design

All the objects are identified by an identifier. This identifier is invisible for the user. The use of the identifier will the logically deletion of an object. Additionally the database integrity is easier to guarantee.

3.3.3 Table Description⁶

Table Customer

Field	Type	Length	Key	Description	Null	Index
Customer Id	nchar	7	Primary	Internal generated customer id		y
Customer Code	nchar	20		Customer code assigned by the user		y
Customer Name	nchar	20		Customer's name		
Customer Surname	nchar	20		Customer's surname		
Address	nchar	50		Customer's address		
City	nchar	20		Customer's city		
Country	nchar	20		Customer's country		
Identification Type	tinyint			Customer's identification type		
Identification Number	nchar	10		Customer's identification number		
Email Address	nchar	40		Customer's email address	y	
Telephone Number	nchar	20		Customer's telephone number	y	
Comments	text			Comment's about the customer	y	

Table User

Field	Type	Length	Key	Description	Null	Index
User Id	nchar	7	Primary	Internal generated user id		y
User Password	nchar	10		User's password		
User Code	nchar	20		User's code		y
User Name	nchar	20		User's name		
User Surname	nchar	20		User's surname		
Blocked	bit			If true the user is blocked and can't connect to the system, the admin user can't be blocked	y	

Table Portfolio

Field	Type	Length	Key	Description	Null	Index
Portfolio Id	nchar	12	Primary	Internal generated customer Id		
Customer Id	nchar	7	Foreign	Identification of the customer who owns this asset		y
Asset Id	nchar	7	Foreign	Identification of the asset that the customer owns		y
Quantity	int			Quantity of the asset that the customer owns, if the asset is a current account the quantity is zero - the balance is stored in the field balance of the object current account		

⁶ Only a selection of tables will be displayed for lack of space

The use of the small date time data type is possible because only dates from 1st January 1900 through 6th June 2079 will be accepted.

The text data type is a variable length character that allows to store up to $2^{31} - 1$ characters, ideal to store long texts, as the comments in this system. This type of data is stored separately from the other data and it can take more time to get the value, but the fields that use this data type are only used for a view and not necessary for search and other time intensive operations. For other purposes it should be avoided.

Small money stores monetary data values from $-214.748,3648$ through $+214.748,3647$.

The use of the nchar/nvarchar allows to store Unicode data, allowing an easier internationalization. Disadvantage is that the double amount of space is used.

The tinyint data type stores the value in only one byte. The small int stores it in 2 bytes. And big int stores the data in 8 bytes.

3.4 Controller Design

The code-behind feature of the Microsoft Visual Studio .NET development system makes it easy to separate the presentation (view) code from the model-controller code. Each ASP.NET page has a mechanism that allows methods that are called from the page to be implemented in a separate class. This mechanism is facilitated by Visual Studio .NET and it has many advantages.⁷ (see Architecture)

The view will inherit from the class System.Web.UI.Page.

```
using System;
using System.Data;
using System.Data.Sql.Client;

public class ExampleController : System.Web.UI.Page
...
```

It has to be defined explicitly how the controller links the occurring events to the actions that must be performed. The InitializeComponent method and the Page_Load function allow managing it. The InitializeComponent procedure initializes the form and its various components. The Page_Load is called every time that the page is loaded.

⁷ Enterprise Solution Patterns Using Microsoft .NET, written by David Trowbridge, Dave Mancini, Dave Quick, Gergos Hohpe, James Newkirk and David Lavigne.

3.5 Flow of Windows

This kind of diagram based in the robustness diagram, uses the symbols of actor and boundary objects. “Boundary objects are objects with which actors will be interacting. These frequently include windows, screens, dialogs and menus.”⁸

The intention is to show the interaction between the user and the different windows.

Unfortunately, for a lack of space only the subsystems authorisation, user management and asset management will be presented in this document.

3.5.1 Authorisation Subsystem

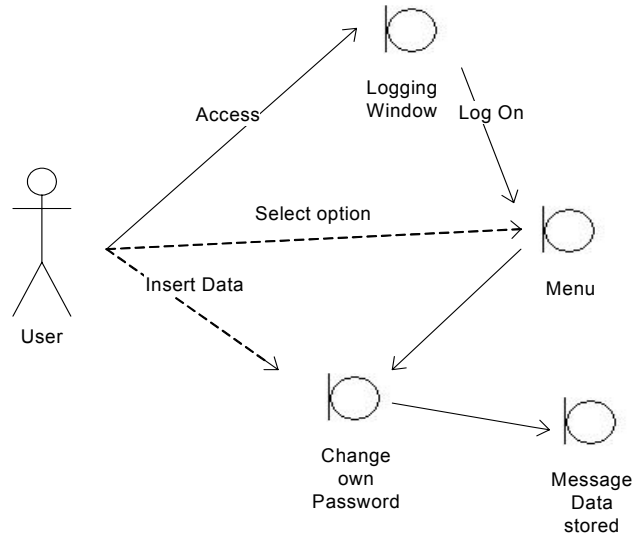


Figure 3-6 Flow Change Password

Once the user has logged into the system on the left side of the window, he can choose the option to change his password in the menu. Selecting that option he will be requested to enter the data to change the current password. After that he will receive a popup window announcing that the change has been stored in the system.

⁸ “Applying use case Drive Object Modeling with UML: An annotated e-Commerce Example” written by Doug Rosenberg and Kendall Scott. Addison Wesley June 2001

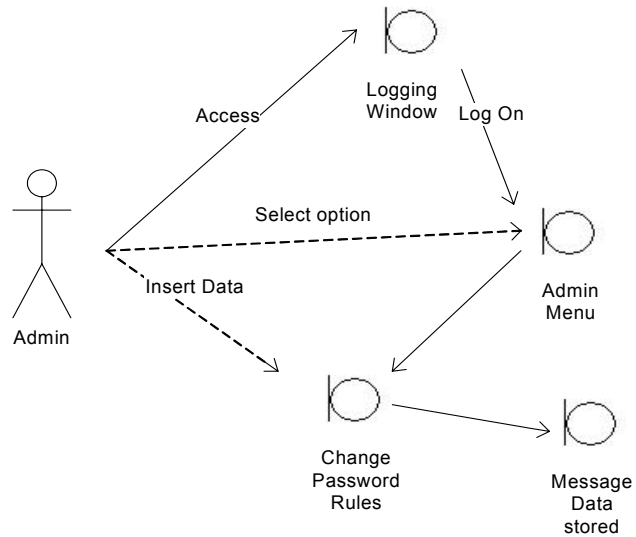


Figure 3-7 Flow Change Password Rules

After a successful login the admin user can change the password rules when he selects the option from the left side menu. In this case he will be shown all the current settings in a window and will be able to change them, receiving a verification message at the end.

3.5.2 User Management Subsystem

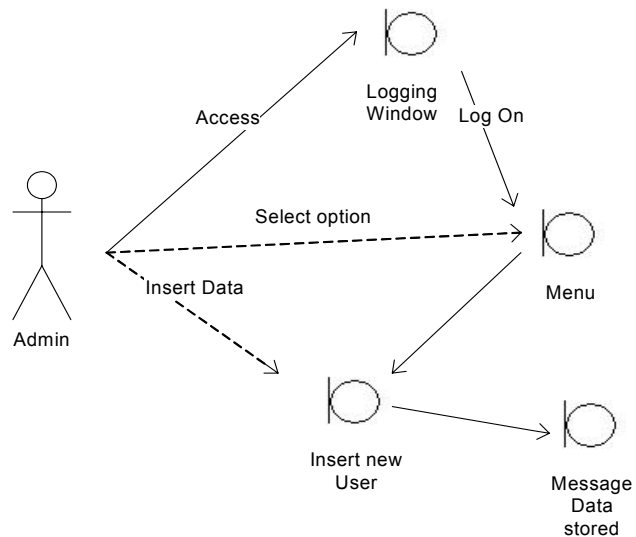


Figure 3-8 Flow Insert New User

If the admin user, logged into the system, wants to insert a new user, the system has to call the function Insert new user from the admin menu. There the all the necessary data can be entered and after saving it the user will get a confirmation message.

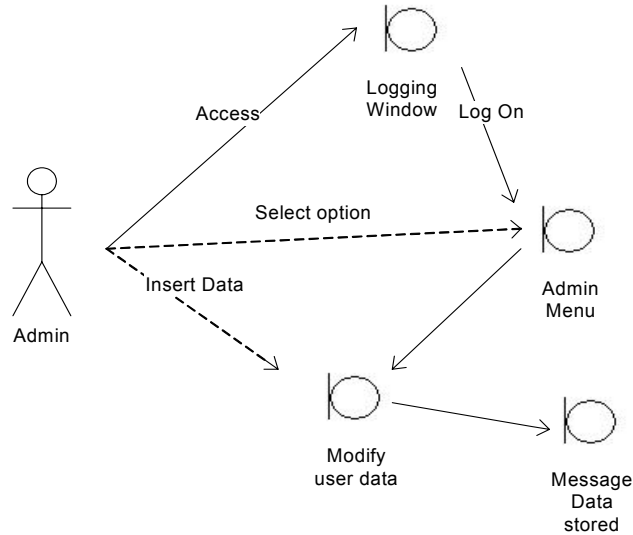


Figure 3-9 Flow Change User Data

The admin, when successfully logged in, can change any data from the user calling the *Modify data user* window from the administrator menu (situated at the left of the window). When he has finished changing the data, he can store the changes and will receive a confirmation message.

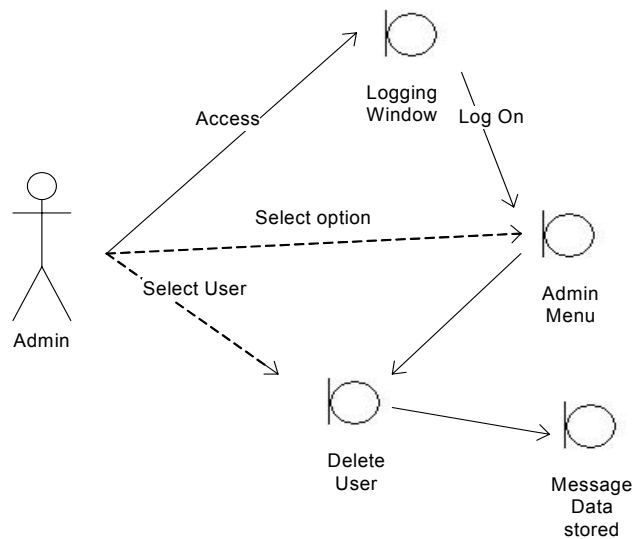


Figure 3-10 Flow Delete an Existing User

If the admin decides to remove an existing user, he has to log onto the system, choose the option *Delete a user*. In the *Delete a User* window he selects the user he wants to remove. When the task is completed a window with the message that the user has been deleted will appear.

3.5.3 Asset Management Subsystem

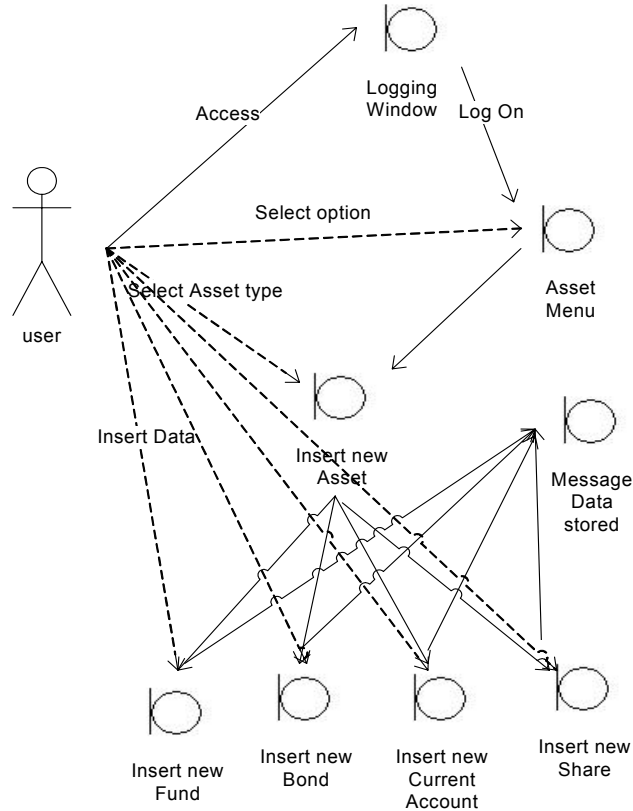


Figure 3-11 Flow Insert a New Asset

When the user decides to create a new asset, from the menu at the left frame – he will go to the insert new asset window and select the asset type he wants to insert. Depending on the type different windows will be called, and after entering all the data and clicking the save button, he will get a message showing that the data has been stored.

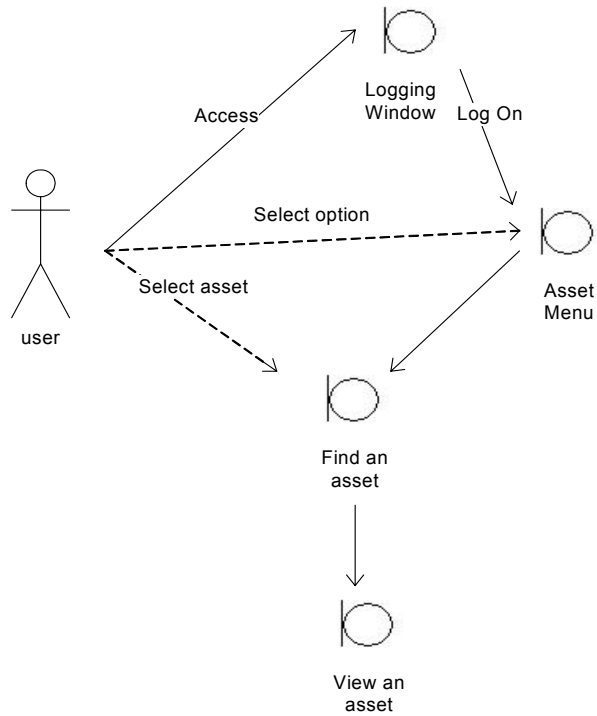


Figure 3-12 Flow View an Existing Asset

The portfolio manager (the user) when logged to the system can call the task view asset from the left menu and view the complete data of an asset.

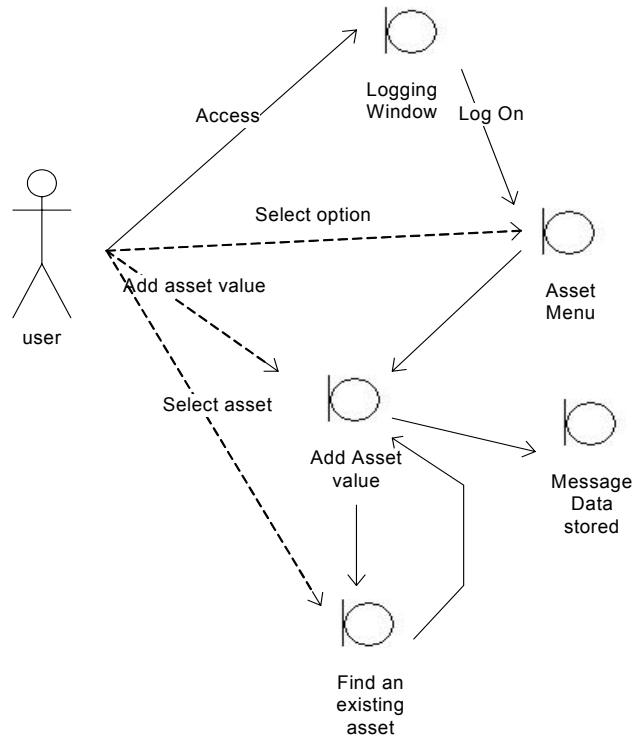


Figure 3-13 Flow Add Asset Value

For any calculations the system needs to know the current prices the assets share and investment fund have. Therefore it is recommended to take care to obtain the current price of the assets. The user can insert the current price of an asset for every day.

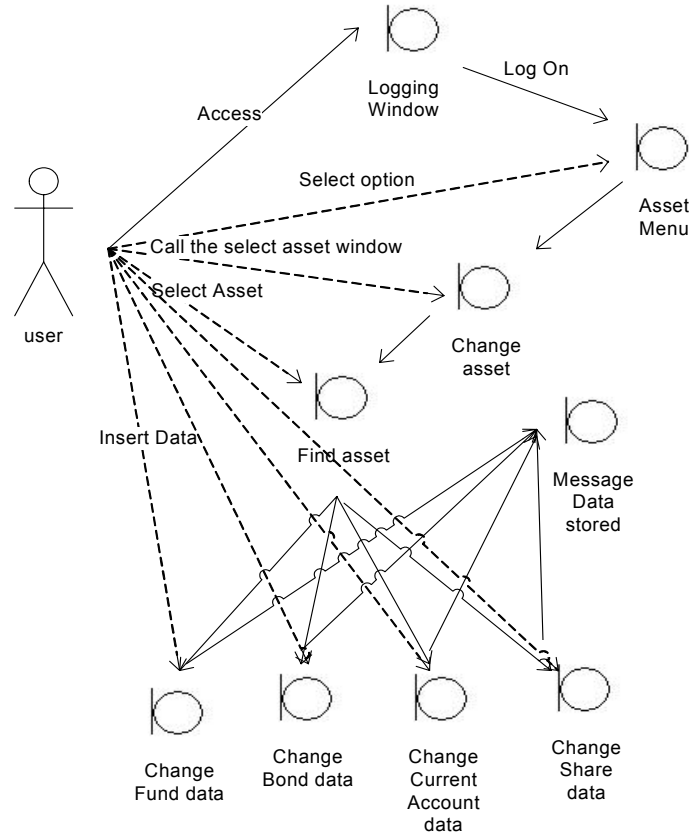


Figure 3-14 Flow Change Data of an Existing Asset

If the user who wants to change some data of an asset he has the possibility to do it by selecting the corresponding option in the left menu. This will call the *Change Asset* window, where the user will select the find asset option. Once he has selected the asset he wants to modify the matching window appears where he can change the data. After saving he will get a confirmation message.

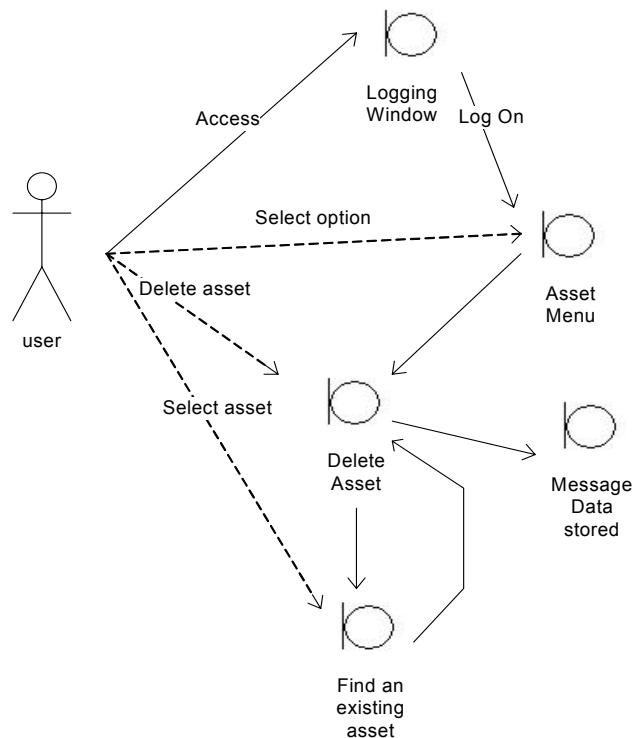


Figure 3-15 Flow Delete an Existing Asset

The last function the user can call in the asset management subsystem is delete an existing asset. To do so he first has to find the asset through the *Find an Asset* window. Once he has selected the asset to be deleted he will press the delete button and wait for a confirmation message.

3.6 User Interface Windows

To separate the implementation from the graphical interface at a maximum a CSS (Cascading Style Sheets) file will be used.

“The Cascading Style Sheets were created to provide a powerful, yet flexible means for formatting HTML content. CSS works much like style sheets in a word processing program – you define a “style” that contains formatting options that can be applied to document elements”⁹

To make easier for the user to identify which fields are mandatory, those fields will be marked with an asterisk.

⁹ „HTML, XHTML, and CSS Bible“ written by Brian Pfaffenberger, Steven M. Schafer, Charles White and Bill Karow. Wiley Publishing Inc. 2004

For the lack of space only the windows for the authorisation, the user management and the customer management subsystems will be presented in this document.

3.6.1 Authorisation Subsystem

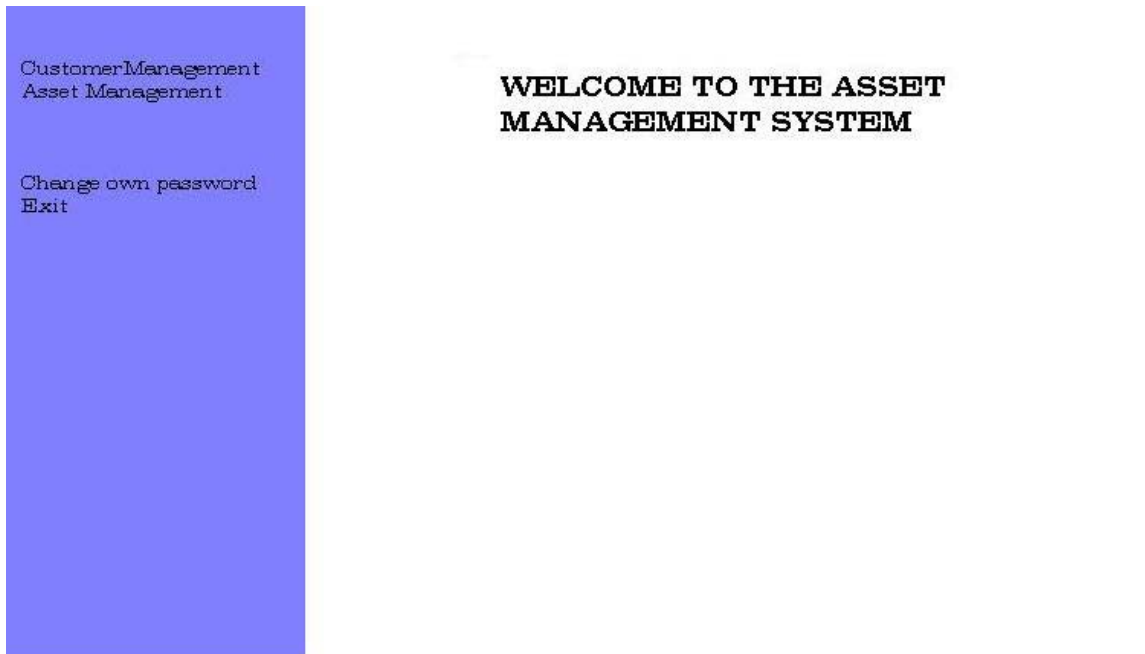


Figure 3-16 Window Welcome to the Asset Management

When a user is successfully logged into the system this window will be shown. At the left side he has the top level of the Asset Management menu.

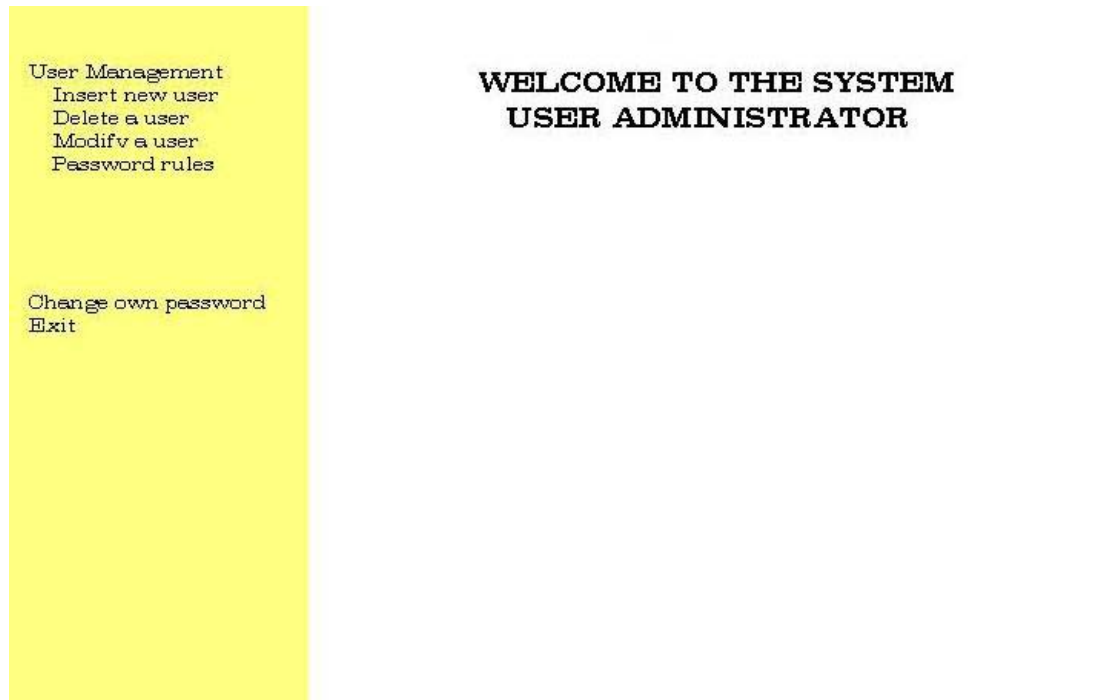


Figure 3-17 Window Welcome to the User Management

This is the window that the user Admin will see when he has successfully logged into the system. At the left side of the window a menu with all possible options for the specific user appears.

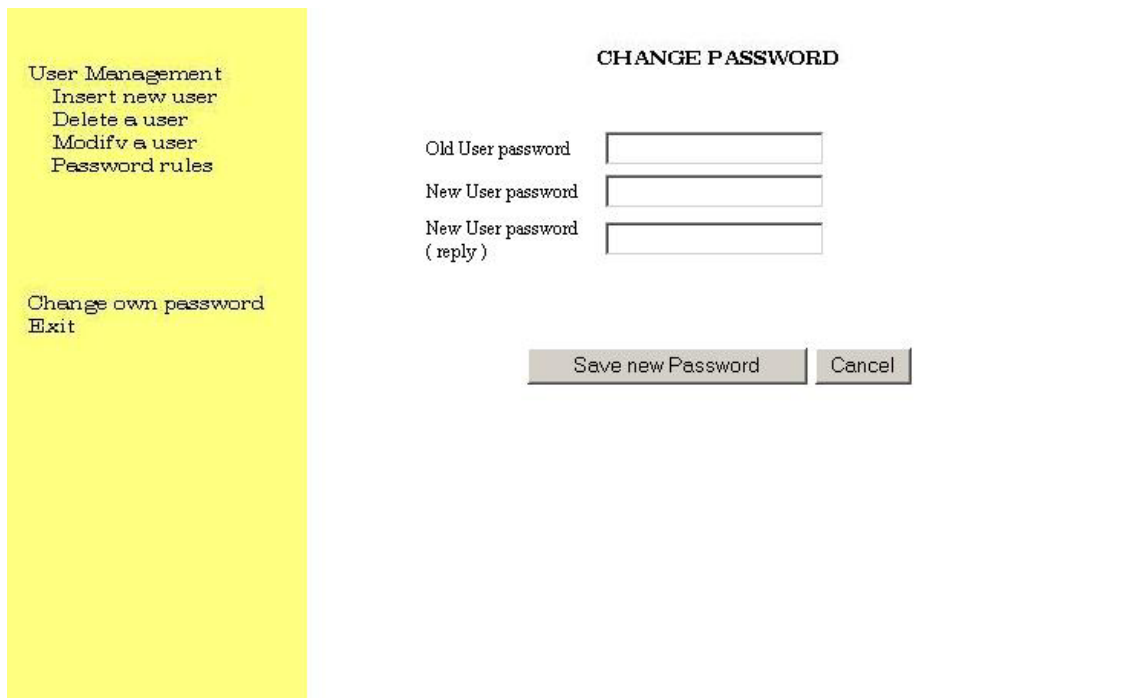


Figure 3-18 Window Change Own Password

This is the window that each user will use to change its own password. (For the non administrator user the menu at the left side will be different).

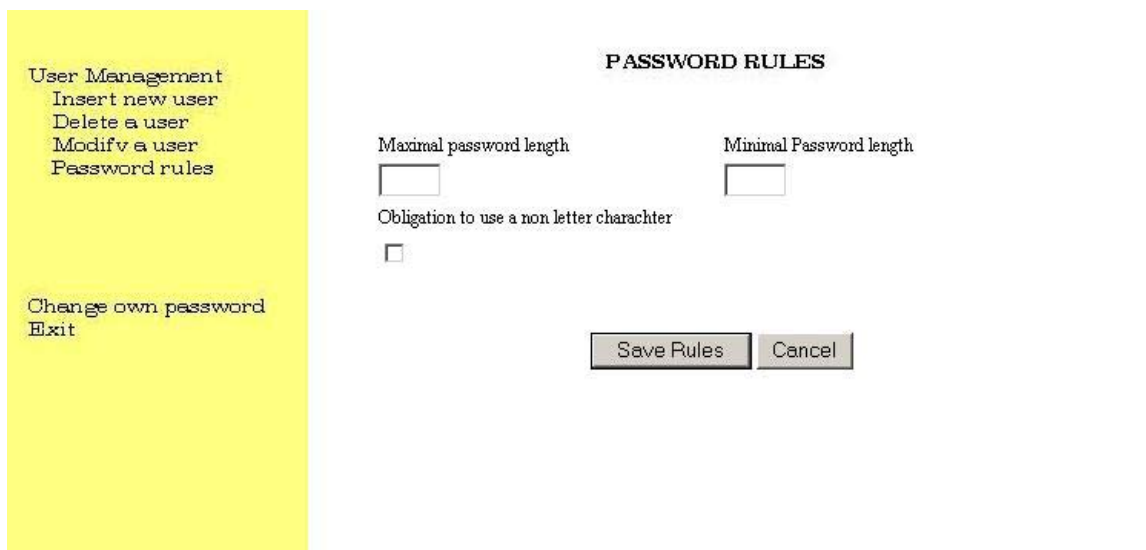
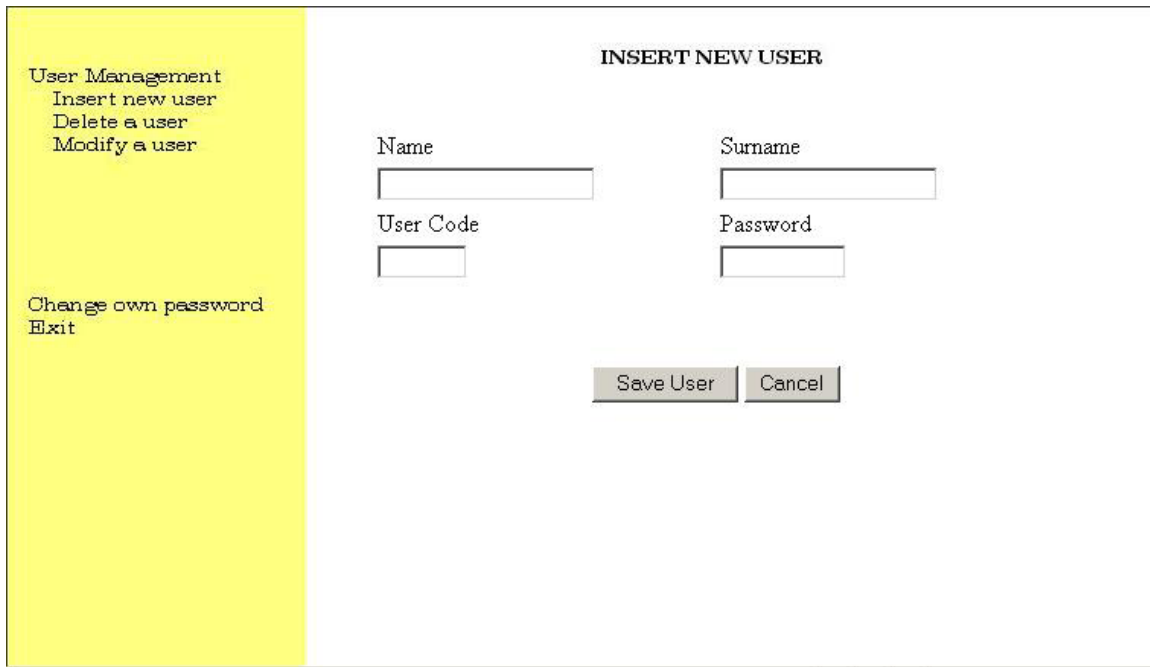


Figure 3-19 Window Show and Change the Password Rules

With this window the admin can see and modify the rules that have to be obeyed when creating a new password.

3.6.2 User Management Subsystem

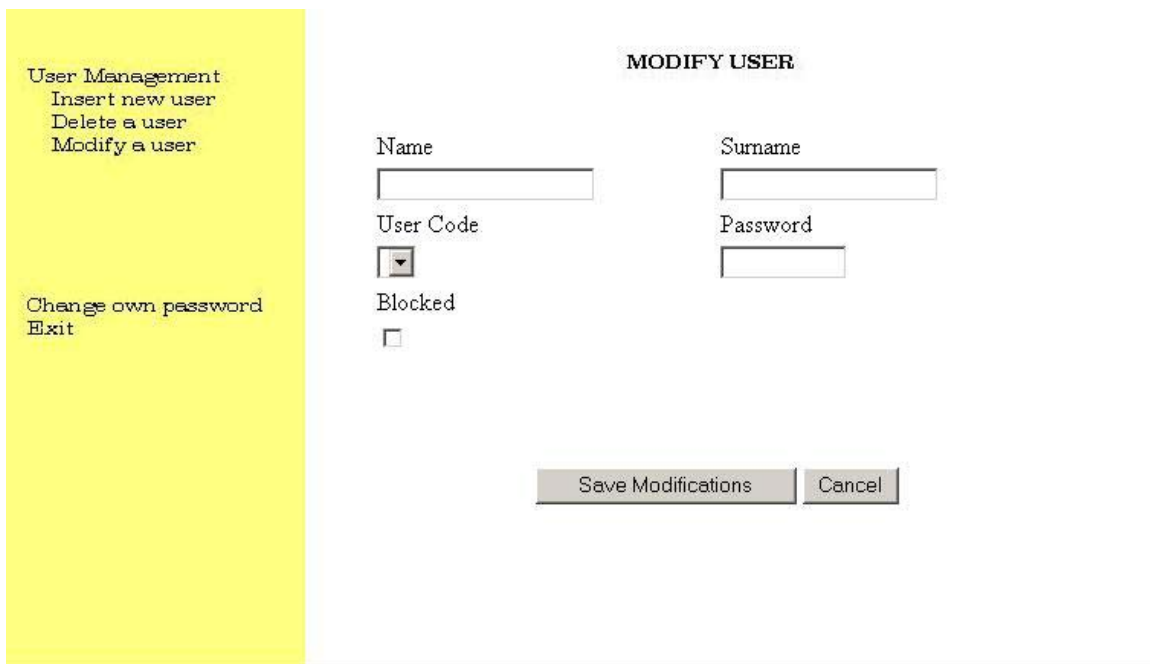


The screenshot shows a web application window titled "INSERT NEW USER". On the left, there is a yellow sidebar menu with the following items: "User Management", "Insert new user", "Delete a user", "Modify a user", "Change own password", and "Exit". The main content area contains the following form fields and buttons:

- Title: INSERT NEW USER
- Name:
- Surname:
- User Code:
- Password:
- Buttons: Save User, Cancel

Figure 3-20 Window Insert New User

In this window the admin user will insert the data of a new user. After clicking the *Save User* button the data will be stored into the database.



The screenshot shows a web application window titled "MODIFY USER". On the left, there is a yellow sidebar menu with the following items: "User Management", "Insert new user", "Delete a user", "Modify a user", "Change own password", and "Exit". The main content area contains the following form fields and buttons:

- Title: MODIFY USER
- Name:
- Surname:
- User Code:
- Password:
- Blocked:
- Buttons: Save Modifications, Cancel

Figure 3-21 Window Modify an Existing User

The administrator, as soon as he selected a user and the information is displayed in the window, can modify each field (including the user code). The field blocked will allow the administrator to control weather or not the user can log into the system.

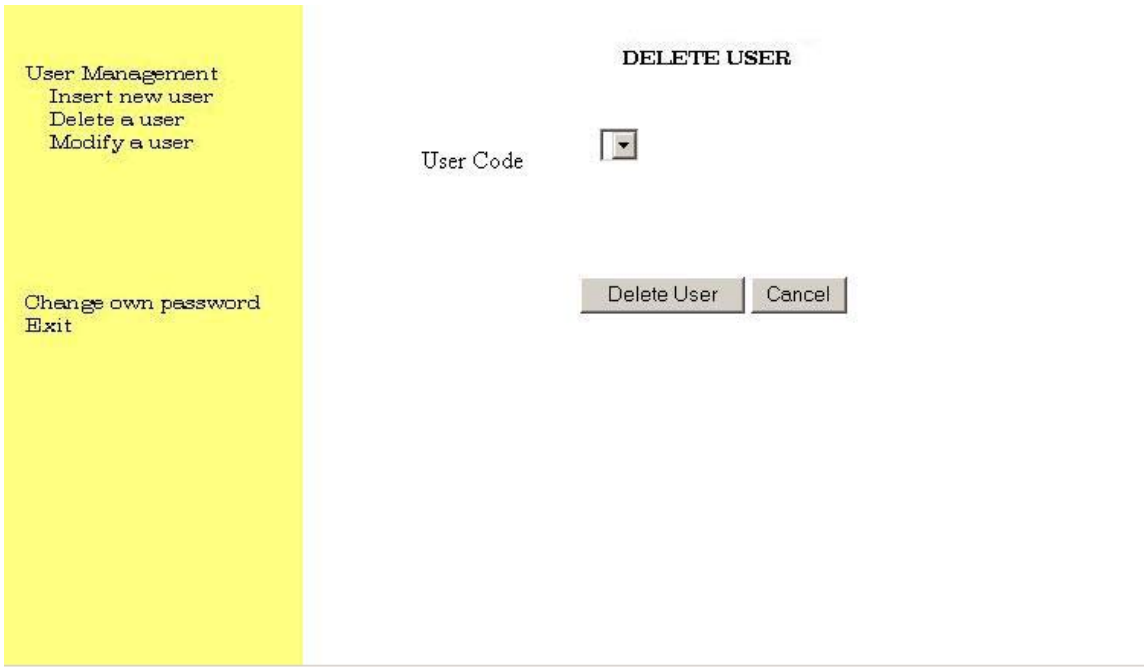


Figure 3-22 Window Delete an Existing User

If the administrator wants to delete a user he only has to select the user code of the user and click the *Delete User* button.

3.6.3 Customer Subsystem

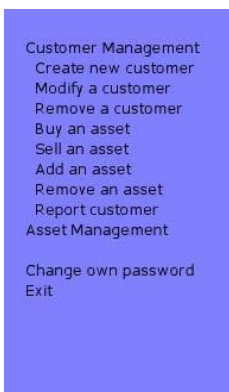
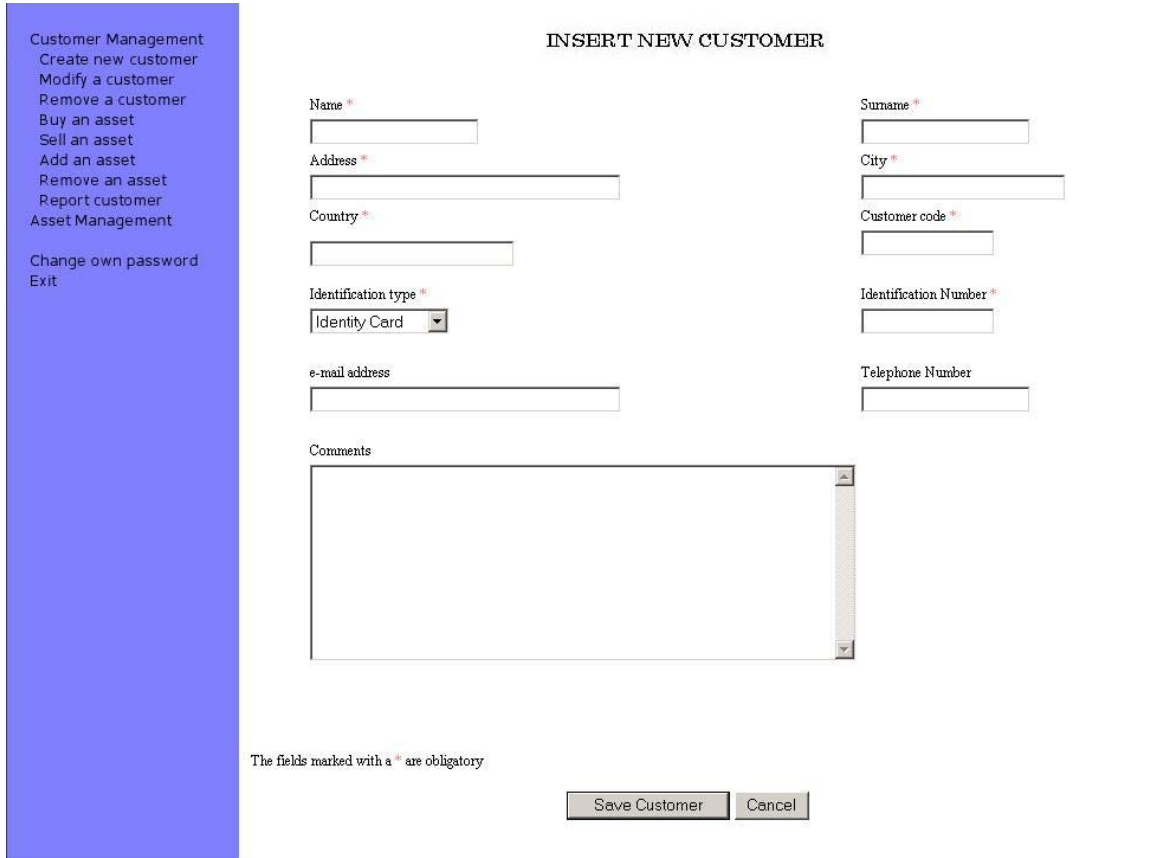


Figure 3-23 Window Customer Management Submenu

Once the user has decided to work with the customer management submenu this new window appears with all the options of this subsystem.



Customer Management
Create new customer
Modify a customer
Remove a customer
Buy an asset
Sell an asset
Add an asset
Remove an asset
Report customer
Asset Management
Change own password
Exit

INSERT NEW CUSTOMER

Name *

Surname *

Address *

City *

Country *

Customer code *

Identification type *
Identity Card

Identification Number *

e-mail address

Telephone Number

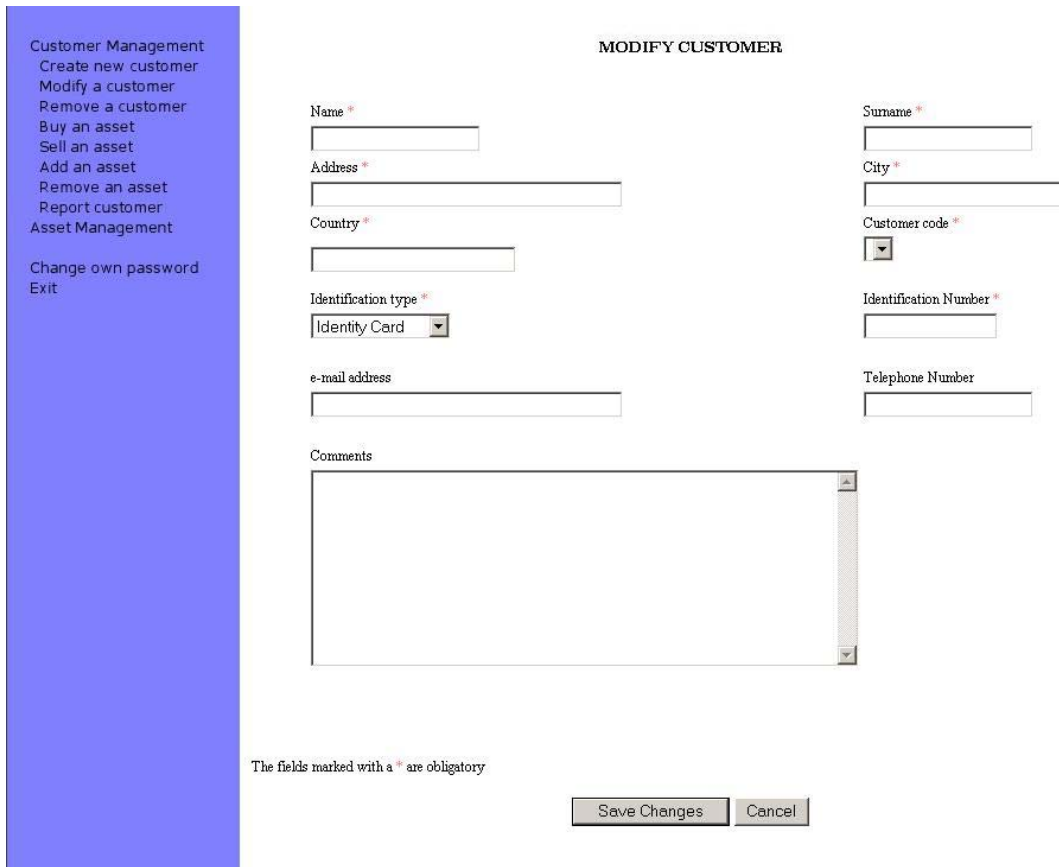
Comments

The fields marked with a * are obligatory

Save Customer Cancel

Figure 3-24 Window Insert New Customer

The user has to fill a number of mandatory fields, they are marked with the red asterisk. The customer code is the code the user will use to search for the customer in the system (other than the internal code, the customer code can be changed while the internal code is the same all the time). All the other fields are optional. The user can add any comments to the customer. Afterwards the user will save the data to the database with the *Save Customer* button.



Customer Management
Create new customer
Modify a customer
Remove a customer
Buy an asset
Sell an asset
Add an asset
Remove an asset
Report customer
Asset Management
Change own password
Exit

MODIFY CUSTOMER

Name *

Address *

Country *

Identification type *
Identity Card

e-mail address

Comments

Surname *

City *

Customer code *

Identification Number *

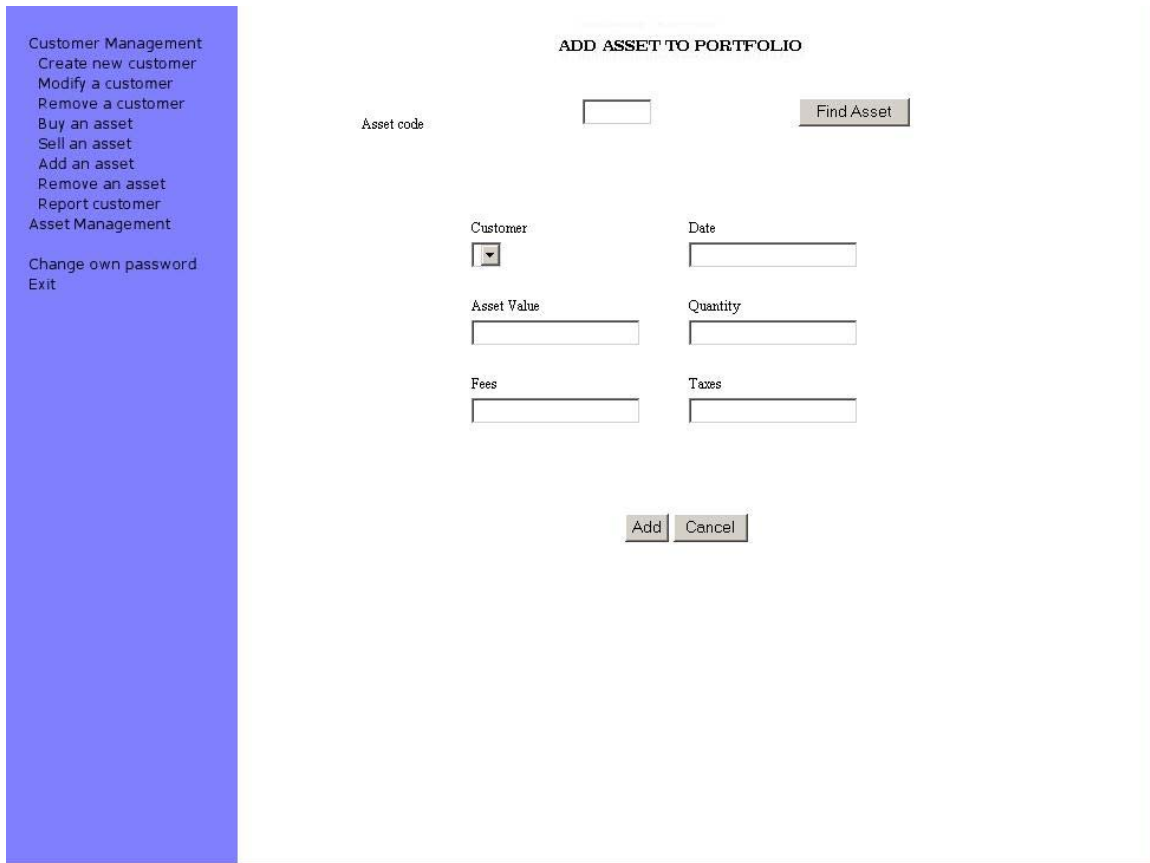
Telephone Number

The fields marked with a * are obligatory

Save Changes Cancel

Figure 3-25 Windows Modify an Existing Customer

If the portfolio manager wants to change the data of a customer he goes to the customer management menu and selects the modify a customer option. In the *Modify Customer* window he selects the customer code, the data fields will be filled by the system and he can change the fields considered necessary.



Customer Management
Create new customer
Modify a customer
Remove a customer
Buy an asset
Sell an asset
Add an asset
Remove an asset
Report customer
Asset Management
Change own password
Exit

ADD ASSET TO PORTFOLIO

Asset code

Customer Date

Asset Value Quantity

Fees Taxes

Figure 3-26 Window Add an Asset to a Portfolio

The user has to select the asset code and fill the fields. After this the user will save the transaction by pressing the add button.

Customer Management

- Create new customer
- Modify a customer
- Remove a customer
- Buy an asset
- Sell an asset
- Add an asset
- Remove an asset
- Report customer
- Asset Management

Change own password

Exit

BUY ASSET

Asset code

Customer	Date
<input style="width: 100%;" type="text"/>	<input style="width: 100%;" type="text"/>
Asset Value	Quantity
<input style="width: 100%;" type="text"/>	<input style="width: 100%;" type="text"/>
Fees	Taxes
<input style="width: 100%;" type="text"/>	<input style="width: 100%;" type="text"/>
Current Account	
<input style="width: 100%;" type="text"/>	

Figure 3-27 Window Buy Asset

The *Buy Asset* window is identical as the *Add Asset to a Portfolio* window with the only difference that here appears the field current account.

- Customer Management
 - Create new customer
 - Modify a customer
 - Remove a customer
 - Buy an asset
 - Sell an asset
 - Add an asset
 - Remove an asset
 - Report customer
- Asset Management
- Change own password
- Exit

REMOVE ASSET FROM PORTFOLIO

Asset code

Customer Date

Asset Value Quantity

Fees Taxes

Current Account

Figure 3-28 Window Remove Asset from Portfolio

The reverse to add an asset to a portfolio is to remove it. The fields necessary for this task are identical to the other window.

Customer Management

- Create new customer
- Modify a customer
- Remove a customer
- Buy an asset
- Sell an asset
- Add an asset
- Remove an asset
- Report customer

Asset Management

- Change own password
- Exit

SELL ASSET

Asset code

Customer

Date

Asset Value

Quantity

Fees

Taxes

Current Account

Figure 3-29 Window Sell Asset

Due to the difference between remove an asset from a portfolio and sell an asset, in this windows the field *Current Account* of the customer comes into view.



Figure 3-30 Window Delete an Existing Customer

The user who wants to delete a customer just has to select the customer code and press the *Delete Customer* button.

3.6.4 Batch process subsystem

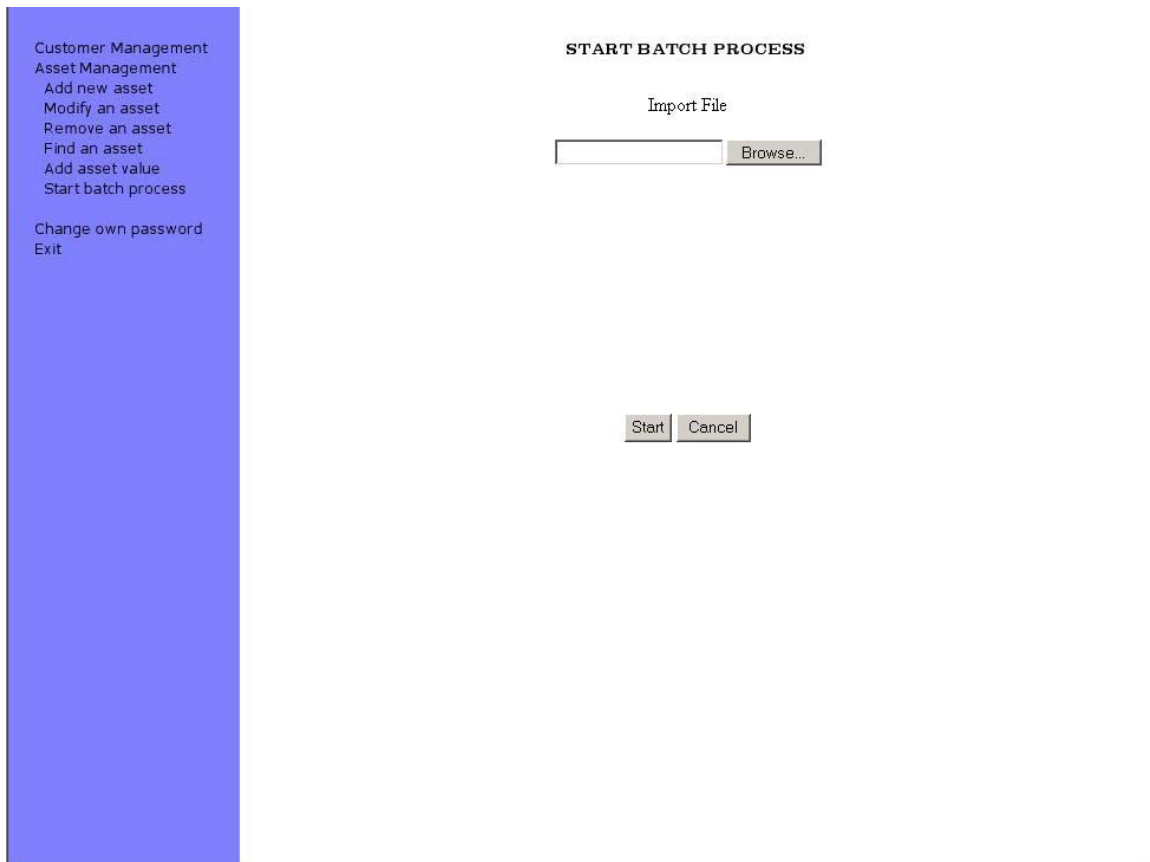


Figure 3-31 Window Start a Batch Process

The user has to select a file containing the information in XML format and then simply click the start button.

4 Implementation

Experience shows that the cascade life cycle of a program is not realistic. The biggest problem is when the analysis and design steps are ended, new requirements and changes can appear. Therefore an incremental and iterative life cycle is the best solution. The application will be developed using the initial specifications, afterwards it will be tested and new specifications will be generated according to the results. Using the new specifications the program will be changed and tested again. And so on until the desired result is reached or until there is no more time for the project.

This project is not different from other projects. That's why it was necessary to make changes to the initial analysis and design.

The difference between the design and the final implementation is perceptible in different points, for instance in the database design. Following the Database diagram from the implemented database in Microsoft SQL Server. The diagram can be compared with the one at the design chapter (Figure 3-5 Database Design).

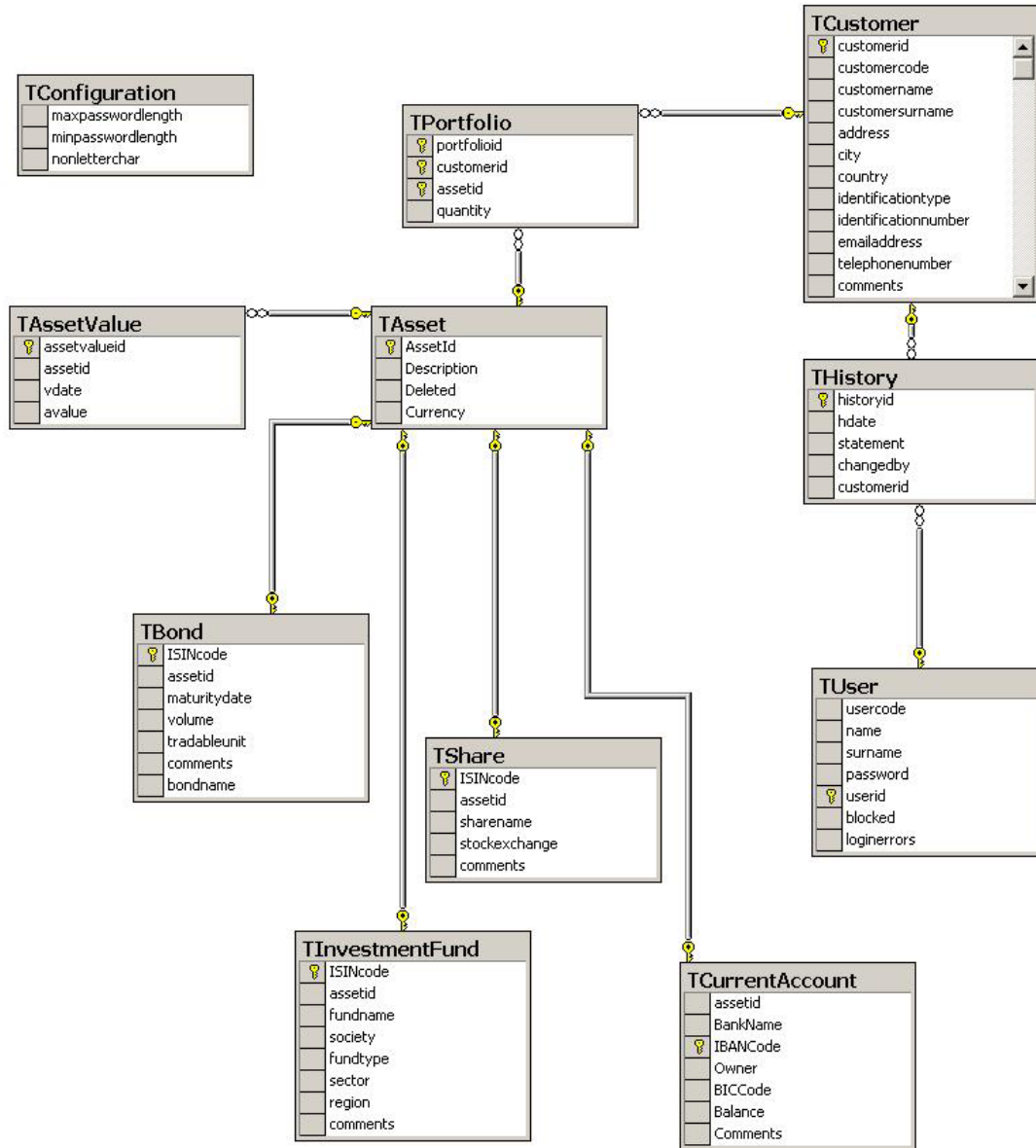


Figure 4-1 Implemented Database Diagram

The differences are not huge, the differences are more in details. But they are easy to notice in the history table. It's easier to store all the SQL statements in the database than to store each changed field. It requires less time and not necessarily more space.

Another good example for the change of the design is the impossibility to read a null value from the database stored in an integer field. For database a fields with data type integer (for example) allowing null values, the system will throw an exception when you try to use the method `SqlDataReader.GetInt32()` when the stored value in the database is null. That this exception can be complicated to handle is preferred the use of a value, that logically can't be used in this field, in substitution to a null value (e.g. a negative price for the asset value). That implies that the specification will be changed. and that instead of the use of a null value, another value will be stored in the database

5 Conclusions and Recommendations

The main objective of the dissertation, learning to use the ASP.NET development environment, is reached. Also how to develop with the Microsoft Visual Studio .NET, the different tools from ASP, the singularities from the programming language C#, the ADO.NET libraries to access the database, the .NET framework, etcetera. These basics were used in the implementation of the application.

But not only understanding of the .NET world was acquired. The work with the MS SQL Server, the work with the Microsoft Reporting service, and the use of XML files are also part of the solution for this dissertation.

As each product, the RDBMS has it own characteristics that differ from other relational databases. To find it, to learn about it and to use it were the principal goals.

How to create the XML files and the definition files were the aims for the use of this technology. Since their use is also integrated in the .NET framework made the decision to work with them more easy.

The final aim was to learn the use of the Microsoft Reporting Services, especially the creation of a report and how to integrate it in the project.

The acquired knowledge from this experience offers an initial view in the technology, and it's a good jumping –off place for future projects.

Due to the complexity of the asset management systems, the system can be extended with thousand of new options. For instance, to create different types of reports (win and lost report, taxes calculation reports, ...), increment the number of asset types, create an interface to a backend system that offers the live values of the assets, internationalize the application (different language support and different tax systems), ... The list of options is endless.

6 Annex

6.1 Bibliography

Paul Kimmel. **Advanced C# Programming**. Osborne McGraw-Hill 2002 ISBN 0072224177

Paul Nielsen. **Microsoft SQL Server 2000 Bible**. John Wiley & Sons. December 2002 ISBN 0764549359

Jesse Liberty. **Programming C#**. O'Reilly Media. April 2005 ISBN 0596006993

Rodeney Landrum, Walter J. II Voytek. **Pro SQL Server 2005 Reporting Services Apress** Oct. 2005 ISBN 1590594983

Grady Booch, James Rumbaugh, Ivar Jacobson. **The unified modeling language. User guide**. Addison-Wesley Feb 2004 ISBN 0201571684

Rolf Beike, Johannes Schlütz. **Finanz-Nachrichten lesen – verstehen – nutzen**. Schäffer-Poeschel Verlag Stuttgart. Oct 2001. ISBN 3791019511

6.2 Internet References

The development of this application without the use of information from the Internet would have been impossible. But a complete list of the sites where I got examples, information or simply solutions for different kind of problem would be too long. Therefore here is only a selection of the most important sites.

Information about XML
<http://www.selfhtml.org/>

Code examples
<http://www.codeproject.com/>

ASP Information
<http://www.aspforum.de/>

6.3 Glossary

Asset management

Asset management is the process of managing money for individuals, typically through stocks, bonds and/or cash equivalents. Professional investors manage these assets according to specific stated objectives or investment styles.

Bloomberg

Bloomberg L.P. is a financial news service founded by Michael Bloomberg in 1982. It provides financial news and data to financial companies and organizations in virtually every country in the world through the Bloomberg Terminal, its the core money-generating product. Bloomberg L.P. has grown to include a global news service, including television, radio, the Internet and publications.

Bonds

Bonds are debts issued by companies or governments who guarantee payment of the original investment plus interest by a specified future date.

Current Account

An active account at a bank into which deposits can be paid and from which withdrawals can be made.

DAO

Data access object, describes a model pattern.

Fund

Investment fund or „funds“ for short, are an investment vehicle according to which assets are pooled and jointly managed for investors, invested in securities or real estate. Investors participate by owning shares.

IBAN Code

The IBAN concept was developed by the European Committee for Banking Standards (ECBS) and the International Standards Organisation (ISO) and is an internationally agreed standard ISO 13616: 1997. It was created as a viable and practical international bank identifier, used internationally to uniquely identify the account of a customer at a financial institution, to assist error-free cross-border payments and to improve the potential for payments STP.

Interest Rate

Compensation paid or to be paid for the use of money. Interest is generally expressed as an annual percentage rate.

ISIN Code

International Securities Identification Number. A unique international code which identifies a securities issue. Each country has a national numbering agency which assigns ISIN numbers for securities in that country.

Maturity

The date when the principle amount of a security becomes due and payable.

Natural Person

In jurisprudence, a natural person is a human being perceptible through the senses and subject to physical laws, as opposed to an artificial person, i.e., an organization that the law treats for some purposes as if it were a person distinct from its members or owners.

Portfolio

The entire combination of securities or investments an individual or institution holds. A portfolio can contain a variety of government and company bonds, preferred and common stocks from different businesses and other types of securities and assets.



Reuters

Reuters is a company supplying global financial markets and news media with a range of information products and transactional solutions, including real-time and historical market data, research and analytics, financial trading platforms, investment data and analytics plus news in text, video, graphics and photographs.

Security

Term describes stocks, bonds and other financial instruments.

Share

Certificate evidencing ownership of a fraction of the capital of the company that issued it. Shares can yield dividends and entitle the holder to vote at general meetings. They may be listed on a stock exchange. Also known as a stock or an equity.

Unicode

A 16-bit character encoding scheme allowing characters from Western European, Eastern European, Cyrillic, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, Thai, Urdu, Hindi and all other major world languages, living and dead, to be encoded in a single character set. The Unicode specification also includes standard compression schemes and a wide range of typesetting information required for worldwide locale support.

XML

The Extensible Mark-up Language (XML) is a W3C-recommended general-purpose mark-up language for creating special-purpose mark-up languages. It is a simplified subset of SGML, capable of describing many different kinds of data. Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet.