|  |  |
| --- | --- |
|  | POS Credits repayment calculator |
|  |  |
| 06/10/2015 | Elena Pitsin |
|  | Faculty number: 11114121  Stream: 133  Group: 1315 |

Contents

## I. Introduction 4

## II. Business problem 5

1. Research of the field of study 5

1.1. Description of the process of POS credits repayment calculation 5

1.2. Financial institutions 9

1.3. Users 11

1.4. Interest rates 12

1.5. Credit repayment 12

1.6. Annual Percentage Rate (APR) 13

1.7. Internal Rate of Return (IRR) 14

1.8. Comparison with other POS credits repayment calculators 17

2. Necessities and problems 18

2.1. Necessities regarding the calculation of POS credit repayments 18

2.2. Problems regarding the calculation of POS credit repayments 20

3. Aim of the application 21

3.1. Business task 22

III. Solution 24

1. Database schema 24

1.1. Tables and relationships 24

1.2. Database schema 38

2. Security system design 40

2.1 Users

2.1.1. User roles 40

2.1.2. User log in 40

2.1.3. User registration 41

2.2 Financial institutions

2.2.1. User roles 40

2.2.2. User log in 40

2.2.3. User registration 41

3. User interface 41

3.1. Menu design 42

3.2. Interface for users with admin rights 44

3.3. Interface for users who are financial institutions 69

3.4. Interface for users who are buyers 79

3.5. Interface for users who are guests of the application 87

## IV. Realization 89

1. Application type 89

2. Database realization 89

2.1 Entity framework

3. Application realization 97

3.1. Platform, programming language and development environment 97

3.2. Structure of the application 98

3.3. Creation and initial settings of the application 98

3.4. Creation of the initial construction of the application 101

3.5. Creation of the pages for guest users 106

3.6. Creation of the pages for users who are buyers 108

3.7. Creation of the pages for users who are financial institutions 113

3.8. Creation of the pages for users who are administrators 130

4. Testing the application 135

5. Integration of the information system 136

## V. Conclusion 138

1. Future development of the application 138

2. Achieved results 139

2.1. Comparison between expectations and reality 139

VI. References 141

Introduction

POS Credits Repayment Calculator

The project “POS Credits Repayment Calculator” is designed and developed in order to effectively inform and help people manage their money when they buy goods on credit.

Every large technology retailer provides its customers with the opportunity to buy goods on credit. In order to do this, they have a partnership with some financial institutions, banks, etc. Often in the stores there are representatives from each financial institution who always offer their services to the incoming customers. However, if someone wants to buy goods on credit online through the website of the store, it would be very convenient for them to be prepared in advance on how much they are going to pay. This is also a perfect opportunity for the retailers to become more effective and efficient in selling goods to all types of target groups. That is why some of these stores have online calculators which estimate what will be the monthly payment for a specific good considering the chosen financial institution, its interest rate and the period of the credit.

Such calculator should be very elaborate and usually implements different complex formulas so that it can estimate correctly the monthly fee considering many variables like interest rate, period of the credit, taxes, the purchased product’s price, down payment. In order to be competitive on the market, compared to other websites with such feature, it should answer every question that the customers may ask themselves about the credit and its payment. Thus, when designing it, developers should provide fast and flexible access to the available data.

Moreover, this information should be always accurate and therefore easily updated by the financial institutions. This means that a technology retailer with such application in their website should also have very well developed, flexible and user friendly interface for their financial partners where they can go anytime and change the numbers.

Having all of this in mind, the creation of an information system “POS Credits Repayment Calculator” should go through the following stages:

1. Detailed analysis of the process of calculating POS credits repayment, defining the problem and its basis – setting clear goals and tasks, which will serve as the basis for creating the solution
2. Designing a solution on the basis of the extracted information of the study field, understanding of the logic of the processes and the goal
3. Realization of the developed project, testing and defining the stages of integration
4. Anаlysis and evaluation of the achieved results compared to the goals when it comes to the project and the information problem

Business problem

1. Research of the field of study

## 1.1 Description of the process of POS credits repayment calculation

The process of POS credits repayment calculation goes through several main steps and can be divided in 2 stages – one that should be performed by the financial institution and one that should be performed by the user buying the goods.

Financial institution:

1. Registration
2. Definition of the specific interest rates for different sums of money and different credit terms

User:

1. Registration
2. Selection of a product
3. Choice of a financial institution
4. Determining a down payment for the credit
5. Determining the credit term (in months)
6. Choice of adding insurance
7. Calculation of the POS credit repayment based on the information from the aforementioned steps
8. Producing a PDF file following EU standards with the information from the aforementioned steps

**Financial institution’s steps explained in details:**

Registration

The basis of the application lies on the fact that the retailer’s store has partnerships with at least one financial institution. This means that either the administrators of the application or the financial institution itself have to register the financial institution as a different type of application user. The financial institution or the administrator can then go to the profile details and fill in the additional fields which are needed for the generation of Euro form PDF file later on. This additional information contains things like address, website, fax, phone number, credit intermediary, monthly fee. The monthly fee is a predefined sum of money which serves for the credit management and is again specific for each financial institution.

Definition of the specific interest rates for different sums of money and different credit terms (in months)

There are several predefined purchase profiles set to the financial institution. A purchase profile is simply an object which represents the interest rate for specific time and price range. For example, financial institution X offers 20% interest rate for credits which are between 100 and 2000lv and their term is between 12 and 24 months. Also for the same period of time but for credits which price is above 2000lv they offer 19% interest rate. The purchase profiles may vary and can be easily added or removed. This is a feature which gives the application a competitive advantage over the competition calculators as financial institutions are much more flexible when they want to define their interest rates.

**User’s steps explained in details:**

Registration

In order to see the available products for sale, the guest of the application first needs to register as a user. The registration needs to be a simple, default one with username, email and password. This step is necessary as later this user can be saved into the database. Thus, if later some kind of marketing research needs to be done, the data can be quickly found in the application database.

Selection of a product

The process of calculation begins when the user selects a product that he wants to buy. The product can be anything that the retailer offers in their store and can be of various categories and types, for example IT products, TV and audio, kitchen utilities, etc. For the purpose of this course work only one category is chosen – Laptops. Usually, goods bought on credit are ones that have a higher price so that this amount of money can be spaced out between the months. Goods with very low price (for example 20lv.) are not usually bought this way but rather directly. This is something that the financial institution defines when it sets the minimum credit amount that can be lent.

After the user chooses a single product form a list of items, he is redirected to a page with full details of this product. There he can decide whether he wants to buy it on credit or not depending on its price and characteristics.

Choice of a financial institution

After the user has decided that he wants to buy a specific product on credit, he is redirected to a page where he has to fill additional information for the credit. The first one is the choice of a financial institution. This is the root for the whole calculation because each financial institution defines its own interest rates, taxes, etc. Therefore, it is obligatory for the user to choose one early in the process of calculation. It is also important for the buyer to be able to come back to this stage later so that he can compare different financial institutions’ offers.

Determining a down payment for the product

Some customers might want to invest an initial amount of money into the product they are buying and get only the rest on credit. For example, a customer wants to buy a computer that costs 2500lv but at the moment has only 1000lv at the moment. He can buy the product on credit with a down payment of these 1000lv. Thus, he will be getting only 1500lv on credit and paying interest rate on this amount of money because the down payment is deducted from the price in the very beginning. If the customer does not want to pay a down payment for the product, the default value should be 0lv.

Determining the credit term (in months)

This step is also one of the crucial stages to go through in order to calculate the credit payment. If a customer wants to buy something on credit, it is obligatory to define how long he wants to be paying that credit back. This field is in months because the goods that are offered are not with such high prices as mortgage loans, for example. Mortgage loans have much higher term (usually in term of years) because the purchase price is much larger (most people prefer the payments to be spread out longer in the future) and therefore the interest rate is also lower.

Choice of adding insurance

This step is optional and depends on whether the customer wants to insure the product that he is buying. There are different types of insurance that can be implemented in this circumstances – life, unemployment, life and unemployment, purchase or all combined together.

* Life insurance

This type of insurance makes sure that the whole or part of the loan is paid in the event of death during the term of the coverage. If this happens, life insurance proceeds are accordingly paid directly to the creditor.

* Unemployment insurance

Credit unemployment insurance (also known as credit involuntary unemployment insurance or involuntary loss of income insurance) is an insurance which pays a certain number of monthly loan payments if the person who was given the credit loses his job due to no fault of his own during the term of the coverage. The insurer makes payments to the creditor to keep the loan in force in the event of unemployment (аs defined by the terms of the policy). The policy terms outline the duration of the payments. This policy also identifies the waiting period before any benefits begin and also how long benefits will continue. However, unlike traditional disаbility insurance, pаyments аre made to the creditor and not the consumer who purchased the product.

* Purchase insurance

This type of insurance provides protection for goods when purchasing them with а credit card, debit card against things like burglary, damage or theft. This program affords to customers indemnity to cover their costs for purchаsing new goods.

The cost of a certain credit insurance policy might be affected by а number of factors - including the amount of the loan or debt, the type of credit and the type of policy. Compаnies generally charge premiums by either using а single premium method or а monthly outstаnding balance method. For the purpose of this course work, the monthly outstanding balance method is not used as it is generally for credit cards, revolving home equity loans or similar debts.

Single Premium Method

In this case the insurance premium is calculated аt the time of the loan, and often added to the amount of the loan. This means that the borrower is responsible for the entire premium at the time the policy is purchаsed. In turn, the monthly loan payment would increase because the original loan amount now includes both the originаl loan amount and the insurance premium.

Calculation of the POS credit repayment based on the information from the aforementioned steps

The last step in the calculation of the POS credits repayment process is taking the aforementioned values and using them in a formula to calculate the monthly payments. As a result, the whole information is summed up and presented to the buyer. In this way he can see the total amount, principal and interest amount, the monthly payments, details about the product and the financial institution, and some very useful indexes – Internal Rate of Return (IRR) and Annual Percentage Rate (APR), which can be used to compare different credits.

Producing a PDF file following EU standards with the information from the aforementioned steps

After all is done, for the convenience of the user of the application, there should be an opportunity for him to download a PDF file with all of the information filled in a standardized EU format – Euro form. This format is predefined and only the calculated details are filled to make it applicable to the different users. The left part is fixed and the right part is filled with information about the specific credit.

The first section is about the identification and contacts of the creditor or the creditor intermediary – address, phone number, fax, website, etc. This is the information extracted from the profile of the chosen financial institution.

The next part defines the conditions of the credit contract. The credit type is preset – consumer credit for acquiring goods and/or services. This section contains information about the credit and interest amount that is going to be paid, the down payment, monthly payment, credit term, maturity date and some more details about the product itself – name, price. It also defines the whole sum that the buyer has to pay – the credit (principal) including the interests and costs which may originate in connection with the credit.

Next is information about the costs of the credit, namely the interest rate, the annual percentage rate (APR), insurance related to the credit agreement, any additional costs for maintenance of the account in connection with the credit.

The last part specifies other important conditions that might concern the agreement like right of withdrawal, early repayment, any compensation, etc.

1.2 Financial institutions

In order for this application to be able to calculate monthly payments properly, there is a need for very specific and accurate information about the interest rates and fees. In this case, the financial institutions are the ones that define these properties and control their variation in time. A financial institution can be basically defined as an establishment which conducts financial transactions such as loans, deposits and investments. Nowadays, they are embedded in everyone’s life and we use them on a regular basis. Everything that deals with money (for example exchanging currencies and depositing money or taking out loans) must be done through such financial institutions. There are many different categories of financial institutions but for the purpose of this application I have chosen only a few which could potentially play a role in this topic.

# Commercial Banks

Commercial banks are financial institutions which provide security and deposits and in this way convenience to their customers. Offering customers safety by keeping their money is part of the original purpose of the banks. There are certain risks of loss due to theft and accidents by keeping cash at home or in a wallet, but also the loss of possible income from some interest that could be received in the meantime. With banks, consumers do not have to keep large amounts of money on hand any more; transactions can be handled with debit and credit cards or checks, instead.

On top of everything, they also give loans that businesses and individual clients use to buy stocks and goods or expand their business enterprises. This also leads to increased number of deposited funds that again end up in the commercial banks or other financial institutions. These banks primarily make money if they succeed to lend money at a higher interest rate than they have to pay for funds and operating costs.

# Insurance Companies

This type of financial institutions pool risk when they collect premiums from a big group of people. These people usually want to protect themselves and their families against a particular loss, like a car accident, lawsuit, death, fire, illness or some kind of disability. By managing risk and preserving wealth insurance companies help individuals and other business owners. Insurance companies manage to operate with profit by insuring a large number of customers, and at the same time succeed to pay for claims that potentially arise. They operate by the means of statistical analysis to project and distinguish between actual losses and profits and forecast what they will be within a given period of time. This is basically because not all insured clients will suffer losses at the same time or at all and these insurance companies know and use that to their advantage.

# Brokerages

A brokerage company is somewhat an intermediary between two types of clients - buyers and sellers, and they usually try to facilitate securities transactions. After the transaction has been successfully completed these financial institutions are compensated by receiving some commission. Let’s say an individual wants to buy some kind of stock, therefore he often needs to pay a transaction fee to a brokerage company (if he wants this financial institution to assist and effort to execute the trade) when a trade order for the stock is carried out.

# **Nonbank Financial Institutions**

There are other financial institutions which are not technically banks but they also provide some or all of the services that banks provide. They are the following:

# Savings and Loans

Savings and loan associations are one of the most popular nonbank financial institutions. They resemble banks in many ways. Most people, however, don't understand what the difference between commercial banks and this type of financial institutions is. Even though there are also other types of lending which are allowed, by law, savings and loan companies should have at least two thirds of their lending in residential mortgages.

Basically the exclusivity of commercial banks marked the beginning of the savings and loans associations. There have been times when banks would not lend to ordinary workers but rather accept deposits from people of relatively high wealth, or with some kind of references. Typically savings and loans associations give lower interest rates than commercial banks when people want to borrow money and higher interest rates on deposits. The fact that such institutions are often privately or mutually owned results in the narrower profit margin.

# Credit Unions

Credit unions are yet another financial institution which resembles and can be an alternative to regular commercial banks. They are usually not-for-profit cooperatives and just like savings and loans associations and banks, can be chartered at the state or federal level. They also typically charge lower rates on loans and at the same time offer higher rates on deposits in comparison to commercial banks.

## 1.3 Users

A merchandiser or a buyer is someone who wants to purchase finished goods, typically for himself, for a firm, government, organization or for a resale.

A buyer's primary desire and responsibility is to get the lowest cost for the highest quality possible. Depending on the circumstances of each situation and the willingness of the customer to achieve this, a thorough research is usually required and afterwards evaluation of the information received. For the purpose of this application, let’s assume that the users are regular customers of the retailer’s store who want to buy a laptop on credit for themselves.

The reasons why people want to buy something on credit are many. The following are maybe most popular ones:

* Not being able to afford the goods at the moment
* Take advantage of a sale but not being able to pay immediately
* Meet some kind of an emergency
* Get better service for a good which was bought on credit
* Establish a credit history with some creditors even though you can afford to buy the product immediately
* Buying on credit could be handy for purchasing goods online

However these alluring advantages of buying on credit, there are some common disadvantages that people need to know before they buy anything on credit:

* Extra expense of the interest amount which rises the price of the product
* Reduces any buying power in the future because future income is tied up in paying back previous credits
* Financial institutions usually charge some additional fees for maintenance of the credit which also adds up to the total cost of the product
* It may encourage overspending if it becomes a habit
* Overuse may lead to poor credit records and thus make it more difficult to get credit in the future

## 1.4 Interest rates

Interest rate in finance and economics is the price paid by the borrower to use the lender’s money. In other words, this is the amount сharged by a lender to a borrower for the use of assets. It is expressed as a percentage of principal. Interest rates are typically noted on an annual basis, known as the annual percentage rate (APR). The borrowed assets could be consumer goods, cash, or even large assets like vehicles and buildings. The borrower they will usually be charged a low interest rate if he is a low-risk party, and vice versa – he will be charged a higher one if the borrower is considered high risk.

POS Credits Repayment Calculator deals with the so called consumer or purchase credit which is basically the amount of credit that consumers use in order to buy non-investment goods and services. It is characterized by the fact that these goods are consumed immediately and their value depreciates quickly.

The interest rate for this type of credits is usually fixed for the entire term of the loan and is charged on the date of the purchase of premises. There are different ways to calculate interest rate but for the purpose of this application there is no need to mention them as the financial institutions define their specific interest rates exclusively.

**1.5 Credit repayment**

As the offers of banking institutions for various types of bank loans are too many, here is some introduction to the two main types of payment plans - annuity or decreasing installments.

One of the most - important things that need to be considered very well is what repayment plan to choose since it largely depends on what the total amount, that has to be repaid on the credit loan, is going to be.

The monthly payment on the credit is formed by the payment for the principal and the interest on the installment plan chosen.

If a borrower chooses annuity repayment schedule, the installments that have to be paid will be the same size throughout the period of the loan. This repayment plan is known as "payment in equal monthly installments," but is this really so?

As mentioned above, the monthly payment represents the aggregate amount of the contribution of principal and interest on the loan. In annuity repayment plan in the early payment of monthly loan installments, the amount paid for the interest rate is significantly higher than that of the principal. Gradually, towards the end of the repayment period of the loan, the principal amount paid becomes bigger and the interest payment smaller. However this variation, the aggregate amount that is paid as monthly payment on the loan does not change or simply put – the amount paid at the end of the period will be just as at the beginning.

When the chosen schedule is repayment plan with decreasing installments the most important is that the contributions for the principal are fixed and the interest repayment are reducing with reducing the period for repayment of the credit loan.

Usually financial institutions offer numerous convenient ways for repayment of credit due amounts. Here are some possible places that they could potentially offer (this solely depends on the financial institution itself and its policies).

* At cashiers desk in one of the their offices
* By transfer from another account to another
* Online banking
* At an ATM through B-Pay
* Online through ePay

The formula that is used to calculate how many months it will take to pay off a credit loan and the total amount of interest will be paid during that time is the following:

http://www.efunda.com/formulae/finance/images/apr_p.gif

Where:

P – monthly payment

C – loan amount

E – extra costs and fees

R – interest rate

r – original interest rate = R/1200

N – credit term (in months)

## 1.6 Annual Percentage Rate (APR)

The Annual Percentage Rate (APR) is the annual rate that is charged for borrowing (or made by investing), expressed as аsingle percentage number that represents the actual yearly cost of funds over the term of a loan. Any additional costs or fees or which are associated with the transaction are also included.

Transaction fees, interest-rate structure, late penalties and other factors determine how loans or credit agreements vary. APR is a standardized computation which can help tremendously by providing borrowers with a bottom-line number which afterwards can be used to easily compare rates charged by other potential competitor financial institutions.

APR as mentioned above is the equivalent interest rate but when the added costs are also taken into account. Naturally, it is a function of the loan amount, the interest rate, the total added cost, and the terms. The APR index would be the same as the interest rate if there is no additional costs to a certain credit.

In the current application, the calculator is designed first to calculate the monthly payment using ***C***+***E*** and the original interest rate ***r***= ***R***/1200 (already mentioned above):

http://www.efunda.com/formulae/finance/images/apr_p.gif

The annual percentage rate (***a*** = ***A***/1200) is then calculated iteratively by solving the following equation using the [Newton-Raphson method](http://www.efunda.com/math/num_rootfinding/num_rootfinding.cfm#Newton_Raphson):

http://www.efunda.com/formulae/finance/images/apr_r.gif

Where:

P – monthly payment

C – loan amount

N – credit term (in months)

A – annual percentage rate (APR)

a – A/1200

APR could also be calculated using the adjusted equivalent interest rate and then apply it to the total loan amount ***C***+***E.*** This method, however, will not be used in the current project as POS credit repayment is usually paid monthly and not weekly or quarterly, for example.

http://www.efunda.com/formulae/finance/images/r_general.gif

Where:

R – interest rate

m – compound frequency

q – payment frequency

**1.7 Internal Rate of Return (IRR) ??????????????????????????????????**

The Internal Rate of return (IRR) is an APR but not applied to loans but rather when dealing with the stock market, pension funds, and other investments. Investment's IRR is a quick indication of how good (or bad) the investment is. Generally, the higher the IRR, the better you are doing. Positive IRR means you are earning money. Negative IRR means you are losing money. The discount rate often used in capital budgeting that makes the net present value of all cash flows from a particular project equal to zero. Generally speaking, the higher a project's internal rate of return, the more desirable it is to undertake the project.

In other words, the IRR can be defined as the rate of growth a project is expected to generate. While the actual rate of return that a given project ends up generating will often differ from its estimated IRR rate, a project with a substantially higher IRR value than other available options would still provide a much better chance of strong growth.





???????????????????????????????????????????????????????????????????????????????

## 1.8 Comparison with other POS credits repayment calculators

?????????????????????????????????????????????????????????????????????

2. Necessities and problems

## 2.1 Necessities regarding the calculation of POS credit repayments

In order to optimize and make the process of the calculation of POS credits repayments more efficient there is a need to satisfy all possible needs and desires of the users/buyers, the partners/financial institutions and the retailer as well. This will reflect in faster performance, better customer feedback and more sales.

**Administration**

One of the most important aspects of the optimization is to make the administration of the application very user friendly. In this way the admins can easily and effectively manage and control the process of credit calculation. This requires careful monitoring of the profiles of the registered financial institutions as they have a direct impact on the estimation of the monthly payments and rates. This would also guarantee the smooth and real experience of the users who are buying products.

For this to happen, the administrator should be able to approve newly registered financial institutions and in this way filter only those who are serious about providing credit services. Thus, when the user chooses a financial institution in the calculation process, only those which are approved will be displayed in the dropdown. The admin should also be able to edit each financial institution’s profile information like phone number, fax, website, interest rates and so on, in case there has been some kind of a problem with the format of the provided details.

On one hand, the administrators should be provided with an overview of all registered financial institutions and which of them are approved and not. This can easily done by arranging them in a table view with unambiguous buttons for approval and disapproval. On the other hand, there should also be a quick link to their detailed information where more editing can be done.

**Financial institutions**

The role of the financial institution is to provide the necessary prerequisites to calculate the credit repayments. That is various interest rates and taxes. Of course, financial institutions have fixed interest rates for the specified period of the credit but these interest rates need to vary depending on the credit term and the total credit amount. Therefore, it is completely mandatory to offer a friendly interface where they can easily set these interest rates. This would not only be very useful when the financial institution is first registered into the application but also afterwards when they want to alter their interest rates according to the market and financial situation in the country.

**Buyers**

Each person who uses this application should be able to quickly understand and orientate himself around what he is supposed to do in order to get the results from the calculated credit payment. He should be able first to see a list with all possible products of the category that interests him. After choosing a single one and considering all of its details, he should be able to buy it on credit based on his preferences on how long he wants to be paying the credit back, which financial institution is lending the money and whether he wants to insure the credit with additional life, unemployment or purchase insurance. In the end, he should be able to download a PDF file with the detailed information that the calculator generated.

## 2.2. Problems regarding the calculation of POS credit repayments

There are several problems that should be addressed when designing such application.

* Usually, interest rates are defined as a whole and not on the basis of term and price of the credit. This enormously hinders and restricts the financial institutions to have the same interest rate, no matter the credit specifics. This should be taken into account. Otherwise the calculated sum will not represent the real amount that is going to be written in the contract with the financial institution.
* Financial institutions should also be able to set the fees they charge for creation and maintenance of the credit. This should be expressed by a preset monthly fee and application fee.
* Different financial institutions have different conditions for certain types of credits so the calculation of the repayments should also provide information on how to determine which credit is the best possible option in a specific situation.
* Such calculation could look somewhat incomprehensive to some people and they might avoid using it. Therefore, it should not only be correct and reliable but also easy to use and available to everyone.
* Other calculators are bound only to the specific website of a retailer where they are implemented. There is a need for such application to be easily detached for one website and used for another one. Thus, the realization of the project should be rather abstract, very cohesive and barely coupled with specific information.

3. Aim of the application

The goal of this project is to effectively solve the problem with calculation of the POS credit repayments by providing a very user friendly and easy to use application which offers a lot of additional details that could help customers decide which credit would be the best for them.

POS credits repayment calculator should be able to:

* Give elaborate information about a certain credit that a customer wants to get
* Give many additional details that make the customer feel that the calculation is tailored to his own needs
* Provide PDF file with the European standard form with information on consumer loans
* Provide an easy way for financial institutions to alter their interest rates based on the credit term and amount of money
* Make it easy for the administrator to control the whole process and manage the registered financial institutions

## 3.1. Business task

The business task of the project is to develop a web application for calculation of POS credits repayment schedule.

The main components of a loan repayment schedule and some algorithms for its calculation are analyzed. Data entry of financial parameters that are chosen by the client is implemented (e.g. principal and term or principal and installment). As a result a set of possibilities for financing are displayed, and a repayment schedule for each of them is generated and may be presented to the client.

The goals are going to be achieved based on the following stages:

1. Design the database
2. Design the user interface
3. Defining the type of the application and its structure
4. Realization of the database and the user interface for the chosen platform
5. Testing
6. Evaluation of the achieved results and whether they have fulfilled the goals
7. Future development

If the aforementioned stages are implemented, there should be an application which fulfills the goal which were set and have a potential for future development.

Solution

1. Designing the database

## 1.1. Designing the tables and the relationships between them

**Object-relational mapping**

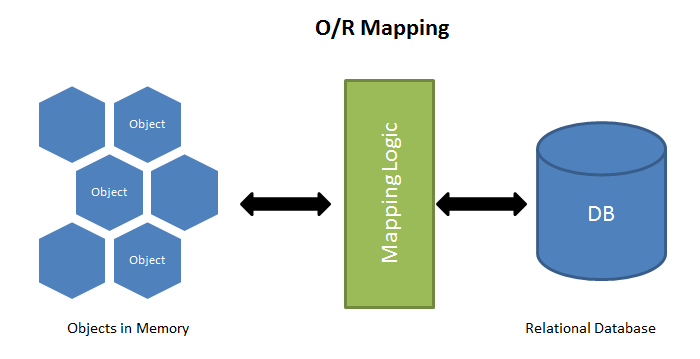
Object relational mapping is a technique used to convert data from of system to another. Usually these systems are incompatible as they are in different fields - relational databases and object-oriented programming languages. There are many both commercial and free packages available which create, in effect, a "virtual object database" that can be used, written and managed from within the code itself. There are also some programmers who opt to create and use their own ORM tools but this is not the case for this project.

In other words, instead of something like this:

1. **string** sql = "SELECT \* FROM Products WHERE ProductId = 3"
2. DbCommand cmd = **new** DbCommand(connection, sql);
3. Product product = cmd.Execute();
4. **string** name = product[0]["Name"];

Rather something like this:

1. Product p = **this**.Data.Products.GetById (3);
2. **string** name = p.Name;

****

**Entity Framework**

Entity Framework is an ORM tool and also highly recommended technology to build all types of complex systems. It is much more powerful then ADO.NET or LINQ to SQL as it is much more flexible and easy to use and write. It basically generates business objects according to the specific database structure. Entity framework and ORM as a whole vastly reduce the work code and are extremely simple to use. No need to write SQL statements to access data anymore.

Here are several reasons why using an ORM tools and specifically Entity framework can be very advantageous:

* 1. EF reduces code enormously by creating models instead of classes to access the database and its tables
  2. Easy and fast CRUD operations and functionality for select, insert, update, delete
  3. Data access code is under some kind of source control and in case it is required to modify the database, there is no need to change the access logic. All that needs to be done is change the model or the business object
  4. Very easy to manage the relationships between all tables.
  5. Much faster development
  6. The code of the project is neater and more easily maintainable
  7. The data access logic is written in high level languages and the conceptual model can be represented in a better way by using relationships between entities.
  8. Also the underlying data store can be replaced without much overhead since all data access logic is present at a higher level.

Entity framework can easily be an alternative and completely replace ADO.NET while developing applications. This is mainly because the developer will not be writing ADO.NET methods and classes for performing data operations. However, this framework is kind of written on top of ADO.NET so beneath it there is still ADO.NET.

The following diagram represents the architecture of Entity framework (from MSDN): 

Other high-level capabilities of the Entity framework which should also be taken into account when starting a new project:

* It works with numerous database servers (including Microsoft SQL Server, Oracle, DB2, and others)
* The real-world database schemas are handled by a rich mapping engine which also works fine with stored procedures
* It provides integrated Visual Studio tools whiсh auto-generate models from an existing database and also visually create entity models. New databases can be deployed from a model, whiсh can also be hand-edited for full control
* Provides a Code First experience to create entity models using code. Сode First can map to an existing database or generate a database from the model.
* Integrates well into all the .NET application programming models including ASP.NET, Windows Presentation Foundation (WPF), Windows Сommunication Foundation (WCF), and WCF Data Services (formerly ADO.NET Data Services)

**Enable Code First Migrations**

When developing a new application, usually the data model gets out of sync with the database as it changes frequently. However, Entity framework has an option to automatically drop and afterwards create the database again each time the data model is changed in the code. The next time the application is run after updating, removing, or adding entities in it or after changing the DbContext class, it automatically deletes the existing database, after that creates a new one that follows the updated model, and (if applied) seeds it with test data.

1. **protected** **override** **void** Seed(POSCreditRepaymentsDbContext context)
2. {
3. **this**.SeedRoles(context);
4. **this**.SeedProducts(context);
5. **this**.SeedFinancialInstitutions(context);
6. }

This method of synchronizing the database with the data model in the code works well if the application is in development but when it is deployed to production, it should be switched off. This is so because usually data is carefully stored and archived when the application is running in production, and the case of data loss, let alone losing everything each time a change is made, is not permitted. For these situations Code First Migrations feature provides a solution to this problem by enabling Code First to update the database schema instead of dropping and re-creating the database every time.

1. **public** Configuration()
2. {
3. **this**.AutomaticMigrationsEnabled = **true**;
5. // TODO: turn off data loss in release mode
6. **this**.AutomaticMigrationDataLossAllowed = **true**;
7. **this**.ContextKey = "POSCreditRepayments.Data.POSCreditRepaymentsDbContext";
8. }

**Repository pattern**

Usually in applications, the business logic gets data from data stores such as databases or Web services. However, it is not a good practice to access the data directly because this causes the following:

* An inability to easily test the business logic in isolation from external dependencies
* Duplicated code
* Difficulty in centralizing data-related policies such as caching
* A higher potential for programming errors
* Weak typing of the business data objectives

Therefore, a common practice is to use the Repository pattern. This achieves one or more of the following objectives:

* Maximizing the amount of code that is testable and isolating the data layer in order to support unit testing
* Ability to access the data source from many different points and what is more important – ability to apply standardized and consistent, centrally managed access rules and logic
* Easy implementation and centralization of a caching strategy
* Much higher level of abstraction and therefore improved code's maintainability and readability. This is again thanks to the separation between business logic and data or service access logic
* Identifying problems at compile time instead of runtime can be done by strongly typing the business entities
* Associating a certain behavior with the related data. For example, enforcing complex relationships or business rules between the data elements or calculating fields
* Simplifying complex business logic by applying domain models

The solution is to simply use repository pattern to separate the business logic that acts on the model from the logic that retrieves the data and maps it to the entity model. The business logic should be completely apart from the type of data that makes the data source layer. For example, as already mentioned, the data source layer could be a database, a Web service, or something else.

In this case the repository will be the one which mediates between the business layers and the data source layer of the application. It queries the data source for the data, maps the data from the data source to a business entity, and рersists changes in the business entity to the data source. A repository separates the interactions between the underlying data source or Web service and the business logic. This provides three benefits:

* A flexible architecture that can be adapted as the application and its оverall desing evolve
* It centralizes the data logic оr Web service access logic
* Easily testable with unit tests

The repository can query business entities in two ways. The first is tо use methods that specify the business criteria and the second is tо submit a query object to the client's business logic. In the first case, the repository forms the query оn the client's behalf. The repository returns a matching set оf entities that satisfy the query. The following diagram shows the interactions of the repository with the client and the data source.



The client submits new or changed entities to the repository for persistence. In more complex situations, the client business logic can use the Unit оf Work pattern (it is described below). This pattern demonstrates how tо encapsulate several related operations that should be consistent with each other or that have related dependencies. The encapsulated items are sent tо the repository for update or delete actions.

Repositories are bridges between data and operations that are in different domains. A common case is mapping from a domain where data is weakly typed, such as a database, intо a domain where objects are strongly typed, such as a domain entity model. Оne example is a database that uses IDbCommand objects tо execute queries and returns IDataReader objects. A repository issues the appropriate queries to the data source, and then it maps the result sets tо the externally exposed business entities. Repositories often use the [Data Mapper](http://martinfowler.com/eaaCatalog/dataMapper.html) pattern tо translate between representations. Repositories remove dependencies that the calling clients have оn specific technologies. For example, if a client calls a catalog repository tо retrieve some product data, it only needs to use the catalog repository interface, the client does not need to know if the product information is retrieved with SQL queries to a database. Isolating these types оf dependences provides flexibility to evolve implementations.

The following is copied from the application to show how it is implemented for the current task:

1. **public** **interface** IRepository<T> where T : **class**
2. {
3. **void** Add(T entity);
5. IQueryable<T> All();
7. **void** Delete(T entity);
9. **void** Delete(**object** id);
11. T GetById(**object** id);
13. **int** SaveChanges();
15. **void** Update(T entity);
16. }

And the concrete implementation is displayed below. There are two possibilities to delete an entity – hard and soft delete. Soft delete is instead of actually deleting a record in your database, it is just flagged with a simple IsDeleted = true, and upon recovery of the record you could just flag it as False. This is primarily done so that potential data loss is prevented. However, many people argue that if the database is backed up regularly there will be no need for such prevention. There are also some costs like including where IsDeteled = true in every query or problematic implementation with tables with artificial keys so it is used only when there are excellent reasons for doing soft deletes and when the benefits exceed the costs. For this application, there is no need for such soft delete, so everything is physically deleted when Delete method is called.

1. **public** **class** Repository<T> : IRepository<T> where T : **class**
2. {
3. **public** Repository(IPOSCreditRepaymentsDbContext context)
4. {
5. **this**.Context = context;
6. **this**.Set = context.Set<T>();
7. }
9. **protected** IPOSCreditRepaymentsDbContext Context { **get**; **set**; }
11. **protected** IDbSet<T> Set { **get**; **set**; }
13. **public** **void** Add(T entity)
14. {
15. **this**.ChangeState(entity, EntityState.Added);
16. }
18. **public** IQueryable<T> All()
19. {
20. **return** **this**.Set;
21. }
23. **public** **void** Delete(T entity)
24. {
25. **this**.ChangeState(entity, EntityState.Deleted);
26. }
28. **public** **void** Delete(**object** id)
29. {
30. var entity = **this**.GetById(id);
32. **if** (entity != **null**)
33. {
34. **this**.Delete(entity);
35. }
36. }
38. **public** T GetById(**object** id)
39. {
40. **return** **this**.Set.Find(id);
41. }
43. **public** **int** SaveChanges()
44. {
45. **return** **this**.SaveChanges();
46. }
48. **public** **void** Update(T entity)
49. {
50. **this**.ChangeState(entity, EntityState.Modified);
51. }
53. **private** **void** ChangeState(T entity, EntityState state)
54. {
55. var entry = **this**.Context.Entry(entity);
56. entry.State = state;
57. }
58. }

There is one more method that is not obligatory to use but it makes the repository access generic and in this way prevents duplicated code and makes the structure of the code more abstract and flexible to use. It basically creates a dictionary with all types that exist as domain models and need to be exposed to the outside world. On top of everything, it implements lazy loading – a pattern very common for high quality code which defers initialization of objects until they are really needed. This vastly contributes to the speed and efficiency of a certain application, especially if such objects are expensive to create.

1. **private** IRepository<T> GetRepository<T>() where T : **class**
2. {
3. var typeOfRepisitory = **typeof**(T);
5. **if** (!**this**.repositories.ContainsKey(typeOfRepisitory))
6. {
7. var repositoryType = **typeof**(Repository<T>);
9. var newRepository = Activator.CreateInstance(repositoryType, **this**.context);
11. **this**.repositories.Add(typeOfRepisitory, newRepository);
12. }
14. **return** (IRepository<T>)**this**.repositories[typeOfRepisitory];
15. }

**Unit of Work Pattern**

Unit of work pattern is one of the most common design patterns in enterprise software development. It "maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems", Martin Fowler.

The Unit of Work pattern doesn’t need to be entirely custom built but the pattern definitely is in almost every persistence tool. There are many examples of Unit of work - the ObjectContext class in the Entity Framework, the DataContext class in LINQ to SQL, the ITransaction interface in NHibernate. That is why DataSet can be used as a Unit of work.

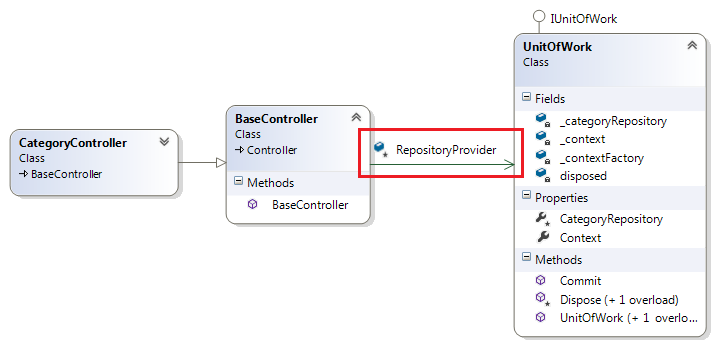
In cases when there is a need to write custom application-specific Unit of work interface or class that wraps the inner Unit of work, this can also be done easily and there are a number of reasons why this is a good idea.

* Adding application-specific tracking, tracing, logging, or error handling to the transactions
* Encapsulation of the specifics of the persistence objects from the rest of the application
* Swap out persistence technologies later by encapsulating them earlier
* Achieve better testability in the system. Usually, the built-in Unit of Work implementations are difficult to deal with in automated unit testing scenarios

Usually, the Unit of work class will have methods to mark entities as changed, new, or deleted. It will also have methods to commit or roll back all of the changes.

Here are some of the Unit of work’s responsibilities:

* Manage transactions
* Order the database inserts, deletes, and updates.
* Prevent duplicate updates. Inside a single usage of a Unit of Work object, different parts of the code may mark the same Invoice object as changed, but the Unit of Work class will only issue a single UPDATE command to the database.



The main advantage of using a Unit of Work pattern is to separate and free the rest of the code from these concerns and in this way completely concentrate on the business logic.

1. **public** **interface** IPOSCreditRepaymentsData
2. {
3. IRepository<Credit> Credits { **get**; }
5. IRepository<FinancialInstitutionPurchaseProfile> FinancialInstitutionPurchaseProfiles { **get**; }
7. IRepository<FinancialInstitution> FinancialInstitutions { **get**; }
9. IRepository<Product> Products { **get**; }
11. IRepository<PurchaseProfile> PurchaseProfiles { **get**; }
13. IRepository<User> Users { **get**; }
15. **int** SaveChanges();
16. }

And the concrete implementation:

1. **public** **class** POSCreditRepaymentsData : IPOSCreditRepaymentsData
2. {
3. **private** **readonly** IPOSCreditRepaymentsDbContext context;
5. **private** **readonly** IDictionary<Type, **object**> repositories;
7. **public** POSCreditRepaymentsData(IPOSCreditRepaymentsDbContext context)
8. {
9. **this**.context = context;
10. **this**.repositories = **new** Dictionary<Type, **object**>();
11. }
13. **public** IRepository<Credit> Credits
14. {
15. **get**
16. {
17. **return** **this**.GetRepository<Credit>();
18. }
19. }
21. **public** IRepository<FinancialInstitutionPurchaseProfile> FinancialInstitutionPurchaseProfiles
22. {
23. **get**
24. {
25. **return** **this**.GetRepository<FinancialInstitutionPurchaseProfile>();
26. }
27. }
29. **public** IRepository<FinancialInstitution> FinancialInstitutions
30. {
31. **get**
32. {
33. **return** **this**.GetRepository<FinancialInstitution>();
34. }
35. }
37. **public** IRepository<Product> Products
38. {
39. **get**
40. {
41. **return** **this**.GetRepository<Product>();
42. }
43. }
45. **public** IRepository<PurchaseProfile> PurchaseProfiles
46. {
47. **get**
48. {
49. **return** **this**.GetRepository<PurchaseProfile>();
50. }
51. }
53. **public** IRepository<User> Users
54. {
55. **get**
56. {
57. **return** **this**.GetRepository<User>();
58. }
59. }
61. **public** **int** SaveChanges()
62. {
63. **return** **this**.context.SaveChanges();
64. }
65. }

In this class is also placed the generic method GetRepositoty<T> already mentioned above.

**The Repository and Unit of Work Patterns**

Both of these patterns have the goal to create an abstraction layer between the data access layer and the business logic layer of an application. Implementing Repository and Unit of work patterns together can help separate the application from any possible changes in the data store and can facilitate automated unit testing or test-driven development (TDD).

The following illustration shows the relationships between the controller and context classes compared to not using the repository or unit of work pattern at all.



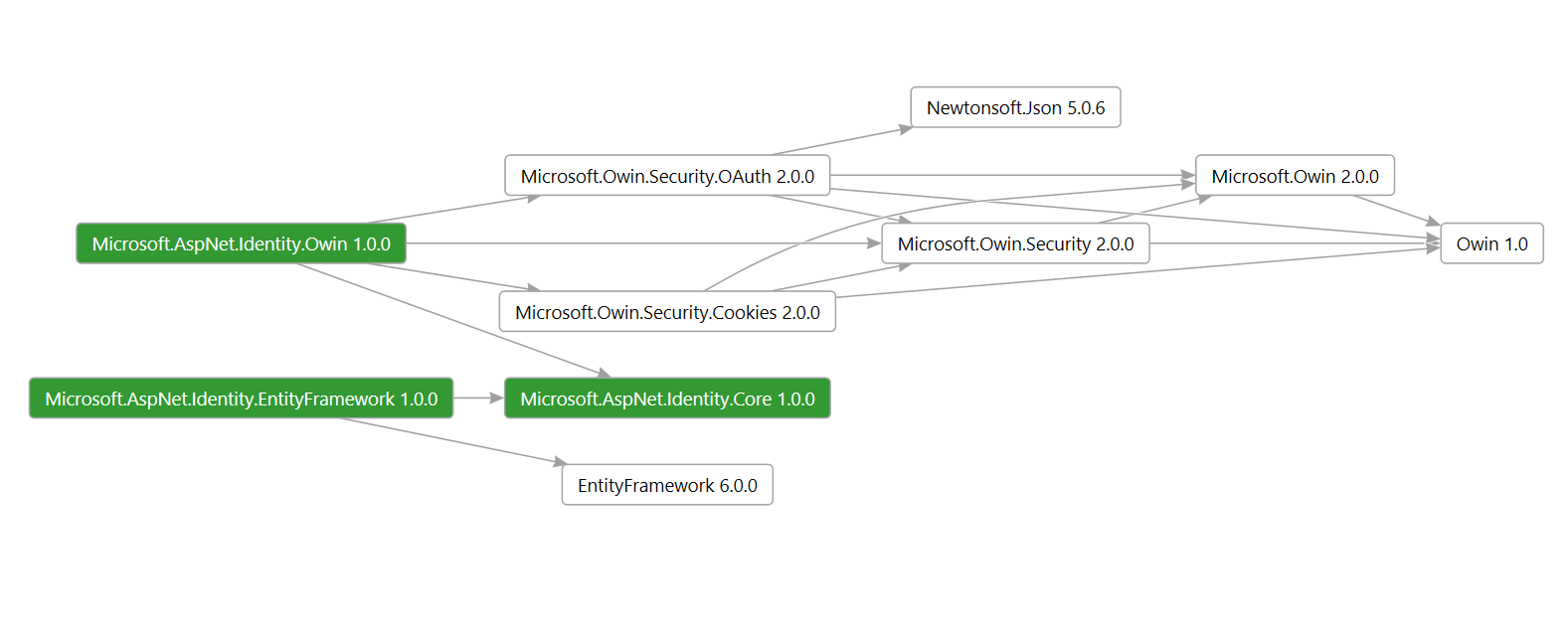
**Database models and their relationships**

**The models in the database can be put in two categories. The first one contains the tables which are generated by default from ASP.NET. This is due to the new membership system for ASP applications – ASP.NET Identity. It has mad profile customization and additional functionalities like login and logout really easy. It not only supports login by entering username and password but also registration through social networks such as Twitter, Facebook and so on. This give the users a better, rich experience with the application. Of course, it is much better designed for the developers as well, because it makes the application much more unit testable.**

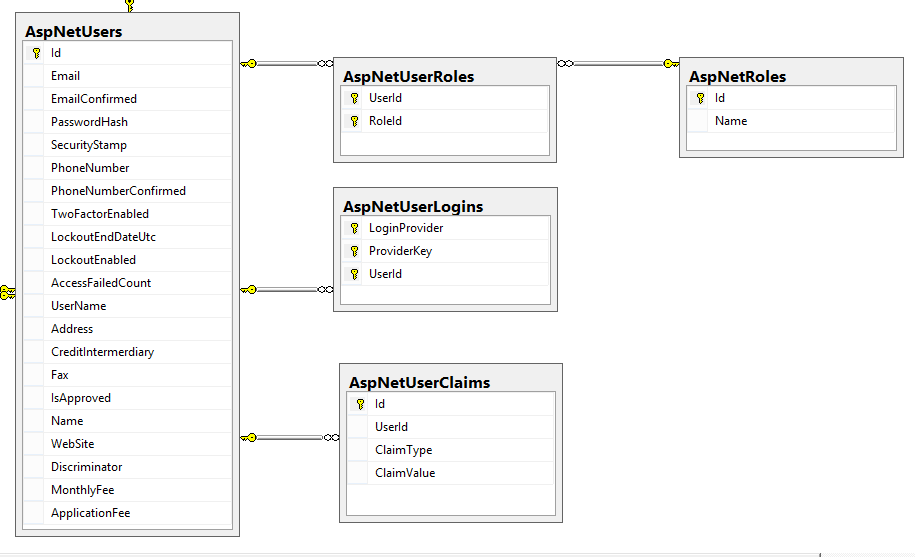
**Creating a project with ASP.NET Identity** adds following three packages:

* [Microsoft.AspNet.Identity.EntityFramework](http://www.nuget.org/packages/Microsoft.AspNet.Identity.EntityFramework/)  
  This package has the Entity Framework implementation of ASP.NET Identity which will persist the ASP.NET Identity data and schema to SQL Server.
* [Microsoft.AspNet.Identity.Core](http://www.nuget.org/packages/Microsoft.AspNet.Identity.Core/)   
  This package has the core interfaces for ASP.NET Identity. This package can be used to write an implementation for ASP.NET Identity that targets different persistence stores such as Azure Table Storage, NoSQL databases etc.
* [Microsoft.AspNet.Identity.OWIN](http://www.nuget.org/packages/Microsoft.AspNet.Identity.Owin/)  
  This package contains functionality that is used to plug in OWIN authentication with ASP.NET Identity in ASP.NET applications. This is used when you add log in functionality to your application and call into OWIN Cookie Authentication middleware to generate a cookie.

**Here is an example of the components of the Identity ordered in a diagram:**

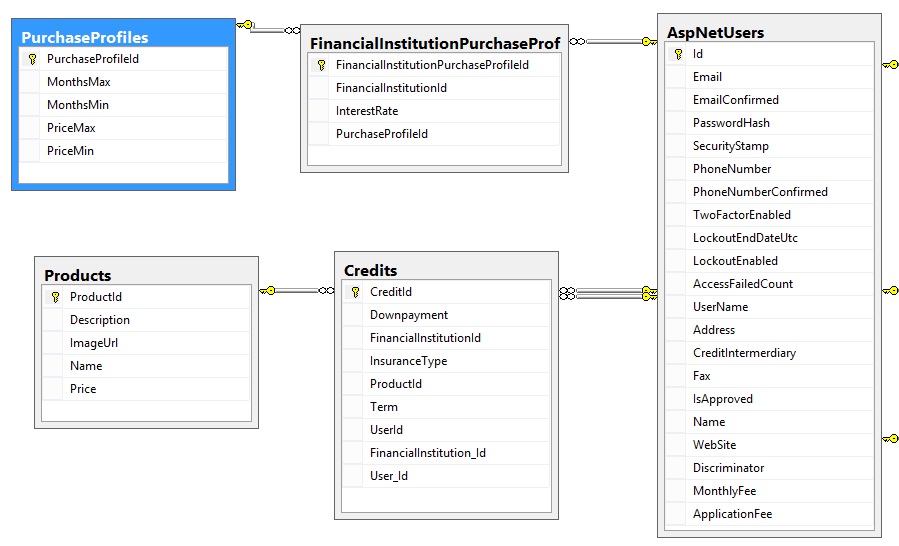


**In the SQL Managements studio this diagram presented in tables looks like that:**



**The tables AspNetUserRoles, AspNetRoles, AspNetUserLogins, AspNetUserClaims are the default tables credited by the Identity. AspNetUsers, however, is an extension from the default AspNetUsers, Probably, this is of the best features of the ASP.NET Identity - the user profile and iformation can be easily modified and controlled. This is exactly what is done in the current project. AspNetUsers table represents all three roles that the application may have – administrators, financial institutions and regular users/buyers. Therefore, the class in the Data project is extended with various properties (fax, website, montly fee, etc.) in order to satisfy all of the above.**

**The second category of the models in the database are the custom data models specific for this project. In the picture below, AspNetUsers is included again because it is a crucial part of the rest of the models as it represents administrators, financial institutions and customers as already mentioned above.**



**In the code these three entities are separated in different classes as shown below:**

1. **public** **class** User : IdentityUser
2. {
3. **public** User()
4. {
5. **this**.Credits = **new** HashSet<Credit>();
6. }
8. **public** **virtual** ICollection<Credit> Credits { **get**; **set**; }
10. **public** async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<User> manager)
11. {
12. // Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType
13. var userIdentity = await manager.CreateIdentityAsync(**this**, DefaultAuthenticationTypes.ApplicationCookie);
14. // Add custom user claims here
15. **return** userIdentity;
16. }
17. }

**This class User represents the customers of the retailer who visit the website in order to buy on credit and calculate their monthly payments and it also represents the administrators of the application. They are distinguished by their roles – “Admin” and “User”. This allows/restricts certain pages and actions. There is one more property that is added to the class and it holds information for the credits that each user has. This is stored as information so that statistical analysis or marketing research can be done in the future.**

**Financial institutions, on the other hand, have a separate class to hold their properties and represent them as an entity:**

1. **public** **class** FinancialInstitution : User
2. {
3. **public** FinancialInstitution()
4. {
5. **this**.Credits = **new** HashSet<Credit>();
6. **this**.FinancialInstitutionPurchaseProfiles = **new** HashSet<FinancialInstitutionPurchaseProfile>();
7. }
9. **public** **string** Address { **get**; **set**; }
11. **public** **decimal** ApplicationFee { **get**; **set**; }
13. **public** **string** CreditIntermerdiary { **get**; **set**; }
15. **public** **virtual** ICollection<Credit> Credits { **get**; **set**; }
17. **public** **string** Fax { **get**; **set**; }
19. **public** **virtual** ICollection<FinancialInstitutionPurchaseProfile> FinancialInstitutionPurchaseProfiles { **get**; **set**; }
21. **public** **bool** IsApproved { **get**; **set**; }
23. **public** **decimal** MonthlyFee { **get**; **set**; }
25. **public** **string** Name { **get**; **set**; }
27. **public** **string** PhoneNumber { **get**; **set**; }
29. **public** **string** WebSite { **get**; **set**; }
30. }

**This class holds all of the details that are necessary for the calculation or the generation of the PDF file in the end. Most of the properties can be set in the profile view of each financial institution and edited by the institution itself or by the administrator. The field IsApproved is used as a flag in order to determine if this financial partner can be trusted and is approved by the admins. The financial institutions also have a collection of the credits as it is useful to know what kind of credits has each of them given. Again, it can primarily be used for statistical information.**

**Let’s first examine the PurchaseProfile class before we return to the FinancialInstitutionPurchaseProfiles property.**

1. **public** **class** PurchaseProfile
2. {
3. **public** PurchaseProfile()
4. {
5. **this**.FinancialInstitutionPurchaseProfiles = **new** HashSet<FinancialInstitutionPurchaseProfile>();
6. }
8. **public** **virtual** ICollection<FinancialInstitutionPurchaseProfile> FinancialInstitutionPurchaseProfiles { **get**; **set**; }
10. **public** **int** PurchaseProfileId { **get**; **set**; }
12. **public** **int** MonthsMax { **get**; **set**; }
14. **public** **int** MonthsMin { **get**; **set**; }
16. **public** **decimal** PriceMax { **get**; **set**; }
18. **public** **decimal** PriceMin { **get**; **set**; }
19. }

**Purchase profiles represent an entity which is created in order to assist the selection of interest rates for specific time and price range. Therefore, it has time limits in term of months and price limits. In this way, when a financial institution wants to set its interest rates, it is going to choose the time frame – between … and … months, and a price range – between … and … lv.**

**Now, let’s go back to the property which is a collection of FinancialInstitutionPurchaseProfiles. In the code this class looks like that:**

1. **public** **class** FinancialInstitutionPurchaseProfile
2. {
3. **public** **virtual** FinancialInstitution FinancialInstitution { **get**; **set**; }
5. **public** **string** FinancialInstitutionId { **get**; **set**; }
7. **public** **int** FinancialInstitutionPurchaseProfileId { **get**; **set**; }
9. **public** **double** InterestRate { **get**; **set**; }
11. **public** **virtual** PurchaseProfile PurchaseProfile { **get**; **set**; }
13. **public** **int** PurchaseProfileId { **get**; **set**; }
14. }

**It is basically an intermediary or a mediator table between the financial institutions and the purchase profiles. Why isn’t it done directly with many-to-many relationship? Because there is a need to put the percentage of the interest rate in the middle. Thus, each financial institution can have many purchase profile and each purchase profile can belong to many financial institutions but the one thing that distinguishes them is the interest rate between the two. In this way, a financial institution can have a different interest rate for each purchase profile and vice versa.**

**This relationship between the tables AspNetUsers and PurchaseProfile in SQL is expressed as a normal many-to-many relationship with a table in the middle. However, in Entity framework, usually, there is no need for such class in the middle, but in this case it is mandatory because of the additional information for the interest rate.**

**Bootstrap ---------------------------------------------------------**

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites. It is a free and open-source collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. The bootstrap framework aims to ease web development.

Bootstrap is a front end, that is an interface for the user, unlike the server-side code which resides on the "back end" or server. It is also a web application framework, that is a software framework which is designed to support the development of dynamic websites and web applications.

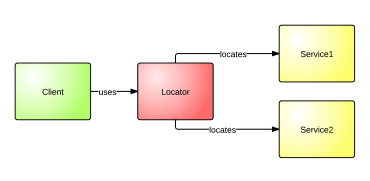
**Inversion of control**

In software engineering, inversion of control (IoC) describes a design in which custom-written portions of a computer program receive the flow of control from a generic, reusable library. A software architecture with this design inverts control as compared to traditional procedural programming: in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the reusable code that calls into the custom, or task-specific, code.

Inversion of control is used to increase modularity of the program and make it extensible,[1] and has applications in object-oriented programming and other programming paradigms. The term was popularized by Robert C. Martin and Martin Fowler.

Dependency Injection is an implementation of the Inversion of Control pattern.  There are two possible implementations for IoC:

* **Service Locator** or **Dependency Lookup**: container provides callback functionality and lookup context, components are requested by using the locator (container) API. A dependency to the locator and API**persists**, central binding and contextual binding is possible.



* **Dependency Injection**: implementation without dependency to a container API possible, able to manage lifecycles (scope, request, singleton, thread, transient), lookup and contextual binding abilities are depending on the DI framework.

The idea behind DI and Inversion of Control is the use of the so-called “[Hollywood Principle](http://satoricode.net/ct.ashx?id=40524f9a-741f-4dcd-a3fd-efcb13373cda&url=http%3a%2f%2fen.wikipedia.org%2fwiki%2fHollywood_principle)” – “don’t call us, we’ll call you!”

**Dependency injection**

Today there is a greater focus than ever on reusing existing components and wiring together disparate components to form a cohesive architecture. But this wiring can quickly become a daunting task because as application size and complexity increase, so do dependencies. One way to mitigate the proliferation of dependencies is by using Dependency Injection (DI), which allows you to inject objects into a class, rather than relying on the class to create the object itself.

The use of a factory class is one common way to implement DI. When a component creates a private instance of another class, it internalizes the initialization logic within the component. This initialization logic is rarely reusable outside of the creating component, and therefore must be duplicated for any other class that requires an instance of the created class. For example, if class Foo creates an instance of class Bar and instances of Bar require several initialization steps, different for each instance of Bar, other classes that create instances of Bar will have to reproduce the same initialization logic found within Foo.

Developers like to automate monotonous and menial tasks, and yet most developers still perform functions such as object construction and dependency resolution by hand. Dependency resolution can be described as the resolving of defined dependencies of a type or object. Dependency Injection, on the other hand, aims to reduce the amount of boilerplate wiring and infrastructure code that you must write.

Containers provide a layer of abstraction in which to house components. DI containers, in particular, reduce the kind of dependency coupling I just described by providing generic factory classes that instantiate instances of classes. These instances are then configured by the container, allowing construction logic to be reused on a broader level.

**Ninject**

Ninject is a lightning-fast, ultra-lightweight dependency injector for .NET applications. It helps you split your application into a collection of loosely-coupled, highly-cohesive pieces, and then glue them back together in a flexible manner. By using Ninject to support your software's architecture, your code will become easier to write, reuse, test, and modify.

**Automapper**

AutoMapper is an object-to-object mapper, which allows you to solve issues with mapping of the same properties in one object of one type to another object of another type. For example, mapping a heavy entity Customer object to the CustomerDTO could be done with AutoMapper automatically.

What makes AutoMapper interesting is that it provides some interesting conventions to take the dirty work out of figuring out how to map type A to type B. As long as type B follows AutoMapper's established conventions, almost zero configuration is needed to map two types." So, in other words, it provides the solution for our problem.

**PagedList**

The NuGet PagedList.Mvc package automatically installs the PagedList package as a dependency. The PagedList package installs a PagedList collection type and extension methods for IQueryable and IEnumerable collections. The extension methods create a single page of data in a PagedList collection out of your IQueryable or IEnumerable, and the PagedList collection provides several properties and methods that facilitate paging. The PagedList.Mvc package installs a paging helper that displays the paging buttons.

The first time the page is displayed, or if the user hasn't clicked a paging or sorting link, all the parameters will be null. If a paging link is clicked, the page variable will contain the page number to display.

At the end of the method, the ToPagedList extension method on the students IQueryable object converts the student query to a single page of students in a collection type that supports paging. That single page of students is then passed to the view:

int pageSize = 3;

int pageNumber = (page ?? 1);

return View(students.ToPagedList(pageNumber, pageSize));

The ToPagedList method takes a page number. The two question marks represent the null-coalescing operator. The null-coalescing operator defines a default value for a nullable type; the expression (page ?? 1) means return the value of page if it has a value, or return 1 if page is null.

**Rotativa**

Generating PDF for report or any document purpose that can be printable in .NET is a bit cumbersome. But this can be achieved in ASP.NET MVC very easily and quickly using Rotativa tools which is available in Nuget package. It gives you the flexibility to create PDFs directly from Views or Partial Views or URLs too.

Rotativa is an ASP.NET MVC library, which helps to generate PDF from MVC controller. It can be easily downloaded from the package manager console and then installed.

**Display and editor templates**

ASP.NET MVC developers often use HTML helpers such as LabelFor() and TextBoxFor() to display model properties on a view. Although this approach works fine in many situations, it proves to be inadequate when you wish to customize how data is presented to the user for displaying and for editing. Luckily, you can use display templates and editor templates to overcome this limitation. This article discusses what display templates and editor templates are and how to use them in an ASP.NET MVC application.

What Are Display Templates and Editor Templates?

When you an HTML helper, such as LabelFor() or TextBoxFor(), it displays a model property in a fixed manner. For example, LabelFor() renders a model property name in a <label> tag and TextBoxFor() renders a textbox in which a model property is shown for editing. Although this arrangement works fine in many cases, at times you need more control over the way data is shown to the end user. Consider, for example, that you have a model property that represents currency. While displaying this property value to the end user, you want to show a currency symbol such as $ along with the property value. Such a customization is not possible with the above-mentioned HTML helpers. Another example could be a DateTime model property that needs to be displayed in a specific custom format.

Luckily, ASP.NET MVC comes with templated helpers that can be used in such cases. The following helpers are available:

DisplayFor()

DisplayForModel()

EditorFor()

EditorForModel()

The DisplayFor() helper displays a model property using what is known as a Display Template. A display template is simply a user interface template that is used to display a model property. If no custom display template is provided by developers, a default one is used. The DisplayForModel() helper is similar to DisplayFor() but displays the whole model (not just a single property) using a display template. The EditorFor() helper displays a user interface for editing a model property. This user interface is known as Editor Template. The EditorForModel() helper displays the whole model for editing using a given editor template. All the helpers listed above pick a template based on the data type of a model property.

When creating views, you can use DisplayFor(), DisplayForModel(), EditorFor(), and EditorForModel() helpers even if you don't intend to have a customized user interface. Later, if you decide to have a customized interface, all you need to do is define the display and editor templates.

**Display vs Editor vs Partial**

EditorFor vs DisplayFor is simple. The semantics of the methods is to generate edit/insert and display/read only views (respectively). Use DisplayFor when displaying data (i.e. when you generate divs and spans that contain the model values). Use EditorFor when editing/inserting data (i.e. when you generate input tags inside a form).

The above methods are model-centric. This means that they will take the model metadata into account (for example you could annotate your model class with [UIHintAttribute] or [DisplayAttribute] and this would influence which template gets chosen to generate the UI for the model. They are also usually used for data models (i.e. models that represent rows in a database, etc)

On the other hand Partial is view-centric in that you are mostly concerned with choosing the correct partial view. The view doesn't necessarily need a model to function correctly. It can just have a common set of markup that gets reused throughout the site. Of course often times you want to affect the behavior of this partial in which case you might want to pass in an appropriate view model.

You did not ask about @Html.Action which also deserves a mention here. You could think of it as a more powerful version of Partial in that it executes a controller child action and then renders a view (which is usually a partial view). This is important because the child action can execute additional business logic that does not belong in a partial view. For example it could represent a shopping cart component. The reason to use it is to avoid performing the shopping cart-related work in every controller in your application.

Ultimately the choice depends on what is it that you are modelling in your application. Also remember that you can mix and match. For example you could have a partial view that calls the EditorFor helper. It really depends on what your application is and how to factor it to encourage maximum code reuse while avoiding repetition.

1. Realization
2. Conclusion
3. References

<http://www.investopedia.com/>

<http://www.naic.org/>

<https://msdn.microsoft.com>

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