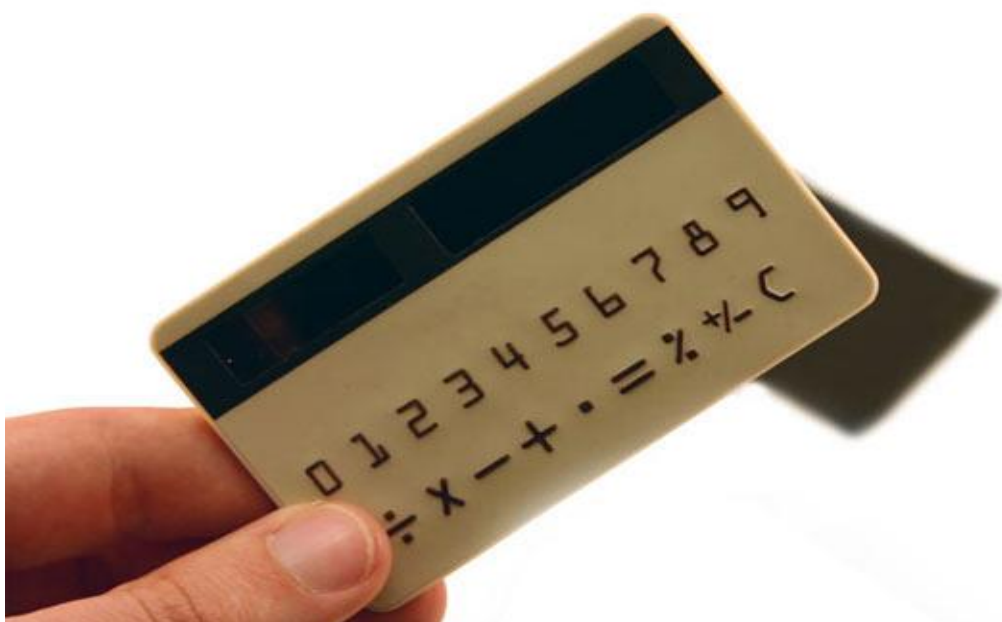




UNIVERSITY OF NATIONAL AND WORLD ECONOMY

DIPLOMA THESIS

TITLE “POS CREDITS REPAYMENT CALCULATOR”



06/10/2015

Elena Georgieva Pitsin

Department “Information Technologies and Communications”

Specialty: Business Informatics

Faculty Number: 11114121

E-mail: epitsin@yahoo.com

Tutor: Monika Tsaneva

Contents

I. Introduction	4
II. Business problem	6
1. Research of the field of study	6
1.1. Description of the process of POS credits repayment calculation	6
1.2. Financial institutions	12
1.3. Users	14
1.4. Interest rates	15
1.5. Credit repayment	16
1.6. Annual Percentage Rate (APR)	18
2. Necessities and problems	21
2.1. Necessities regarding the calculation of POS credit repayments	21
2.2. Problems regarding the calculation of POS credit repayments	22
3. Aim of the application	24
3.1. Business task	24
III. Solution	26
1. Designing the database	26
1.1. Tables and relationships	26
1.2. Database schema	30
2. System access	32
2.1 User roles	32

2.2 User login	33
2.3 User registration	33
3. User interface	35
3.1. Menu design	36
3.2. Interface for users who are guests of the application	36
3.3. Interface for users who are buyers	40
3.4. Interface for users who are financial institutions	45
3.5. Interface for users with admin rights	50
IV. Realization	53
1. Application type	53
1.1. Choice of the platform, programming language and developing environment	53
1.2. Choice of the database management system	54
2. Database realization	56
2.1 Creating the tables and the relationships between them	56
3. Application realization	81
3.1. Structure of the application	81
3.2. Creation and initial settings of the application	86
3.3. Creation of the pages for guest users	87
3.4. Creation of the pages for users who are buyers	92
3.5. Creation of the pages for users who are financial institutions	107
3.6. Creation of the pages for users who are administrators	111
4. Testing the application	114

5. Integration of the information system	116
V. Conclusion	117
1. Future development of the application	117
VI. References	119

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 payment 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. Analysis 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.

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 from 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 (as 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 disability insurance, payments are 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 a credit card, debit card against things like burglary, damage or theft. This program affords to customers indemnity to cover their costs for purchasing new goods.

The cost of a certain credit insurance policy might be affected by a number of factors - including the amount of the loan or debt, the type of credit and the type of policy. Companies generally charge premiums by either using a single premium method or a monthly outstanding 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 at 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 purchased. In turn, the monthly loan payment would increase because the original loan amount now includes both the original 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 or cost of the credit, principal and interest amount, the monthly payments, details about the product and the financial institution, and some very useful indexes – Annual Percentage Rate (APR), which can be used to compare different credits.

The total cost of the credit means all costs on the loan, including interest, commissions, fees, remuneration for credit intermediaries and all other costs directly related to the contract for the consumer credit which are known to the creditor and the consumer has to pay, including the costs of additional services related to the contract for lending of the credit, in particular insurance premiums in cases where the conclusion contract for service is a prerequisite for obtaining the loan, or where the provision of credit is due to the application of commercial terms and conditions. The total cost of credit to the consumer do not include notary fees.

On the other hand, the total amount of credit is the maximum (limit) or the total amount provided under the credit agreement and its interest rate is expressed as a fixed percentage applied on an annual basis to the amount of utilized credit.

Usually purchase credits have a fixed borrowing rate which is the interest rate, primarily laid down in the loan contract, under which the creditor and the consumer agree to a constant interest rate for the entire duration of the credit agreement or negotiated rates for several separate periods of the credit agreement, in which only certain fixed interest rate applies. In the cases when not all of the interest rates on lending to individual periods are defined in the credit agreement, it is assumed that the interest rate on the loan is fixed only for the partial periods in which the interest rate is determined exclusively by means of certain fixed rate agreed at the conclusion of the credit agreement.

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

Before the consumer is bound by a contract or proposal to provide consumer credit, the creditor or credit intermediary provide timely necessary information for the user to compare different offers and to take informed decision to contract consumer credit based on credit terms and conditions offered by the contract.

This information is provided in the form of standard European form to provide information on consumer credit in accordance with Annex № 1.

This form is based on the law and the act to provide financial services and information from a distance. The law states that such form should contain the following:

1. The total amount of credit and the conditions governing the drawdown;
2. The term of the credit agreement;
3. The good or service and its cash price - the loan is in the format system of deferred payment for a specific good or service and linked contracts;

4. The interest rate on the loan, the conditions for its application and each index or reference rate associated with the initial interest rate as the periods, conditions and procedures for changing the borrowing rate; as the case may apply different rates. This information is provided for all the applicable rates;
5. The amount, number and frequency of payments owed by more consumer and, where necessary, the sequence according to which contributions. They will be allocated to the repayment of other outstanding amounts due at different rates in order to repay the loan;
6. The annual percentage rate of charge illustrated by a representative example;
7. The total amount payable by the consumer.

At the request of consumers, the creditor should provide advance and free copy of the draft credit agreement. If, at the moment of the request, the creditor is not willing to proceed with the contract for credit, they may refuse to provide such copy.

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 charged 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:

$$P = \frac{(C + E)r(1+r)^N}{(1+r)^N - 1}$$

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 a single 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):

$$P = \frac{(C+E)r(1+r)^N}{(1+r)^N - 1}$$

The annual percentage rate ($\alpha = A/1200$) is then calculated iteratively by solving the following equation using the [Newton-Raphson method](#):

$$\frac{\alpha(1+\alpha)^N}{(1+\alpha)^N - 1} - \frac{P}{C} = 0$$

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.

$$r = \left(1 + \frac{R/100}{m}\right)^{\frac{m}{q}} - 1$$

Where:

R – interest rate

m – compound frequency

q – payment frequency

When calculating the Annual Percentage Rate on the loan the following costs are not included:

1. The consumer pays for non-fulfillment of its obligations under the contract of consumer credit;
2. Different costs from the purchase price of the goods or service user due upon purchase of a product or service, whether payment is made in cash or by credit;
3. Costs paid to maintain the account in connection with the credit agreement, cost of using a payment instrument capable of making payments related to the acquisition or repayment of the credit, as well as other costs associated with making payments if the account opening is not compulsory and costs associated with the bill clearly stated and separated in the credit agreement or in any other agreement concluded with the consumer.

The law also states that the Annual Percentage Rate cannot be higher than five times the legal interest on arrears in BGN and foreign currency determined by the resolution of the Council of Ministers of the Republic of Bulgaria.

It also specifies that the calculation of the APR should be made taking into account the following:

- Payment by the parties to the contract amounts at different times is not mandatory to be equal and to be paid to the same periods of time
- The starting date is the date of the first drawdown
- Intervals between dates used in the calculations should be expressed in years or parts of the year, for example months

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 be 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 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. Tables and relationships


The first step that should be made is to determine what kind of tables the database should have in order to fulfil the needs of the application and thus those of the customers. In the current project, the tables can be separated in two groups – those that allow the users to enter the application and enable its features, and those which make it possible to calculate some credit payments.

User authorization tables

First and foremost, some roles should be determined based on the access and control over the application. For POS credit repayment calculator there are three roles that should be taken into account – the user, the financial institution and the administrator. All of these roles should have separate tables or at least should be designed in such way that they can be distinguished which is which. Moreover, in order to authenticate themselves, these three types of users should have some identification marks. Here is a solution that can be realized and which satisfies all conditions:

	Column Name	Data Type	Allow Nulls
	Id	nvarchar(128)	<input type="checkbox"/>
	Email	nvarchar(256)	<input checked="" type="checkbox"/>
	EmailConfirmed	bit	<input type="checkbox"/>
	PasswordHash	nvarchar(MAX)	<input checked="" type="checkbox"/>
	SecurityStamp	nvarchar(MAX)	<input checked="" type="checkbox"/>
	PhoneNumber	nvarchar(MAX)	<input checked="" type="checkbox"/>
	PhoneNumberConfirmed	bit	<input type="checkbox"/>
	TwoFactorEnabled	bit	<input type="checkbox"/>
	LockoutEndDateUtc	datetime	<input checked="" type="checkbox"/>
	LockoutEnabled	bit	<input type="checkbox"/>
	AccessFailedCount	int	<input type="checkbox"/>
	UserName	nvarchar(256)	<input type="checkbox"/>
	Address	nvarchar(MAX)	<input checked="" type="checkbox"/>
	ApplicationFee	decimal(18, 2)	<input checked="" type="checkbox"/>
	CreditIntermediary	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Fax	nvarchar(MAX)	<input checked="" type="checkbox"/>
	IsApproved	bit	<input checked="" type="checkbox"/>
	Name	nvarchar(MAX)	<input checked="" type="checkbox"/>
	WebSite	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Discriminator	nvarchar(128)	<input type="checkbox"/>

This table should be connected to another table that will contain all of the possible roles with their IDs and names:

	Column Name	Data Type	Allow Nulls
	Id	nvarchar(128)	<input type="checkbox"/>
	Name	nvarchar(256)	<input type="checkbox"/>


By designing the database in this way all three types of users reside in the table above and they are distinguished by the column Discriminator. It is the one that specifies whether the current user is a regular user/buyer or a financial institution. Of course the regular user has only several of the columns from the above table as its own properties, for example he doesn't have fax or website. In this case these columns can be left empty with default value NULL (that is why they are left nullable).

On the other hand, the admin is recognized from the other table where the name of the role will be the one that is the identification mark.


Credit calculation

The other tables that the database contains should be related to the calculation of the credit repayments and should represent the data in the most sensible and normalized way possible.


As this is a purchase credit, the first thing is to create a table that will contain all the products that are offered by the retailer company. For the calculation the only column that is crucial is the price of the good but afterwards the creation of the PDF in the end needs the name of the products so it is good to fill the name column as well.

	ProductId	int	<input type="checkbox"/>
	Description	nvarchar(MAX)	<input checked="" type="checkbox"/>
	ImageUrl	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Name	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Price	decimal(18, 2)	<input type="checkbox"/>


In order to calculate the credit repayments, there should be some kind of a table that specifies what the interest rate for each good is. The goods usually vary based on their price and the credits vary based on their term and the financial institution that gives them. Therefore, there should be a way to link the price, the term, the interest rate and the financial institution.

	PurchaseProfileId	int	<input type="checkbox"/>
	MonthsMax	int	<input type="checkbox"/>
	MonthsMin	int	<input type="checkbox"/>
	PriceMax	decimal(18, 2)	<input type="checkbox"/>
	PriceMin	decimal(18, 2)	<input type="checkbox"/>

A custom table could be created that will specify the configurations for available purchases. Each of these configurations or purchase profiles will then be linked to the financial institutions so that each of them can specify a certain interest rate for them. This could be accomplished by a middle table that will be the link between the two:


	FinancialInstitutionPurchaseProfileId	int	<input type="checkbox"/>
	FinancialInstitutionId	nvarchar(128)	<input checked="" type="checkbox"/>
	InterestRate	float	<input type="checkbox"/>
	PurchaseProfileId	int	<input type="checkbox"/>

The last table that is necessary for the calculation of the credit is the insurance table. This step of the process is not required but even if it is left empty, there should be a way to reflect this in the database. Such table should contain the insurance rate and a reference to the financial institution that provides this insurance. Insurances also have a type which for purchase credits can be life insurance, unemployment or good protection insurances. This can be accomplished by adding a column with the name of the type of with some nomenclature for easier access in the code:

	InsuranceId	int	<input type="checkbox"/>
	FinancialInstitutionId	int	<input type="checkbox"/>
	PercentageRate	float	<input type="checkbox"/>
	Type	int	<input type="checkbox"/>
	FinancialInstitution_Id	nvarchar(128)	<input checked="" type="checkbox"/>

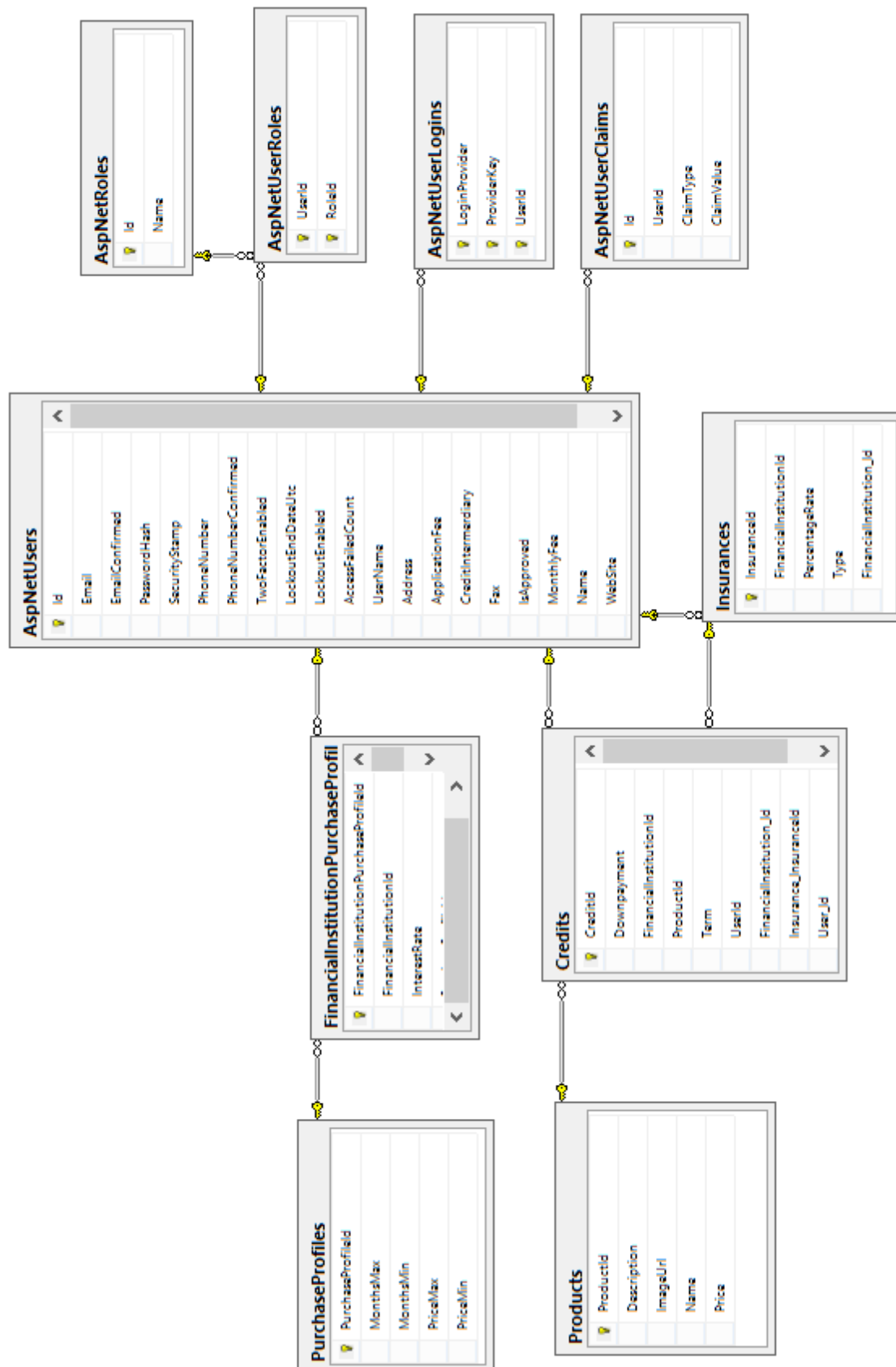
Of course it is a good idea to add a table that will be able to store the calculated indexes and figures of a credit and be used for future statistics and analysis. From this table, the developers or the business people should be able to extract all possible information like the amount of the credit compared to the amount of the insurance. Therefore, it should have references to the financial institution that has lent the credit, the product with its specifications (name, price), the user that has taken the credit and the insurance type and rate (if such exists).

Such table can also be used to speed up the performance when calculating credits. It can be used as an extended table that saves all fields of the credit when it is calculated for each credit. In this way, the next time someone tries to calculate a credit's repayment, a query is going to search the table to see if there isn't such credit with those specifications. If there is one, the data is just going to be mapped to the view model that is displayed to the user. Otherwise, it is going to be calculated and saved for future searches and calculations.

	CreditId	int	<input type="checkbox"/>
	Downpayment	decimal(18, 2)	<input type="checkbox"/>
	FinancialInstitutionId	int	<input type="checkbox"/>
	ProductId	int	<input type="checkbox"/>
	Term	int	<input type="checkbox"/>
	UserId	int	<input type="checkbox"/>
	FinancialInstitution_Id	nvarchar(128)	<input checked="" type="checkbox"/>
	Insurance_InsurancId	int	<input checked="" type="checkbox"/>
	User_Id	nvarchar(128)	<input checked="" type="checkbox"/>

1.2. Database schema

Now that all of the tables and the relationships between them are explained, here is the whole database schema of the application POS Credits Repayment Calculator:



2. SYSTEM ACCESS

What helps manage authorization is the role management. It basically specifies the pages, file and all types of resources that users in the application are allowed to access. All that needs to be done is assign users roles such as admin, manager, financial institution when they first register or afterwards when the administrators decide it's necessary and then these groups of users are treated as a unit.

After these roles have been established, some access rules also need to be created – which pages, for example, will be displayed to members only and which will be displayed to everyone. Similarly, there might be pages which will be accessible only to administrators and so on. By using roles, these types of rules stay independent from individual application users.

In order to work with roles, users must be identified first so that after that it is determined whether the user is in a specific role. In this application the roles are three: Users (customers), Financial institutions and Administrators.

2.1 User roles

Users

Users with such role are regular users of the application who want to buy a product on credit and calculate the POS credit repayment. They have the rights to browse through the available products and choose the one that they prefer.

Financial institutions

Financial institutions have access to their profiles where they can edit their details like interest rates, fees and so on. They then get approved by the admins (if their information is valid and trustworthy), and only after that are they added to the dropdown with available institutions when

customers what to buy products on credit. Financial institutions don't have the right to buy products on credit (therefore access that page) as this is not a valid and likely situation.

Administrators

A user defined as an administrator can approve and disapprove registered financial institutions. They can also edit, add or correct information about almost every field in the database connected with the financial institutions' profiles. They first need to inspect all of the details and only after that to approve financial institutions which have proved themselves honest and trustworthy.

2.2 User login

The authorization in the application is pretty simple – it is the default one in ASP.NET where users enter their username and password which are then checked if they exist in the database and if they match with the ones registered. Of course once a user is logged in, he can manage his profile and change the password as many times as he wants. There is also an option for those who have forgotten their password.

For the convenience of the users there is a checkbox that specifies if the users wants the application to remember his username and password, and not to ask him every single time.

The database is designed in a way that all users no matter their roles log in in one and the same way. This is because they are eventually saved in one table called `AspNetUsers`. This is the most appropriate solution in this situation and it is not only simple as user interface but also easy to implement in the code.

In order to stop any security attacks, the controller that manages the account users anti-forgery tokens.

2.3 User registration

The user registration is also a simple one, slightly extended from the default one that ASP.NET offers. This is because there are two different types of registration – user/admin and financial institution. Administrators cannot register as such by themselves. In the beginning, when the application is first deployed, there is one user with admin role that is seeded. If in the future, the retailer using the application decides that they need more than one, this will be easily done by simply adding a new page with a list of all users where the administrator can approve and disapprove them as admins. But for now this feature is not necessary and therefore not implemented. This can be considered as the famous principle YAGNI – You Ain't Gonna Need It. It specifies that developers shouldn't implement additional features if they are not going to be used in the near future.

In the code, the registration method is protected against forgery with the `ValidateAntiForgeryToken`.

3. USER INTERFACE

A hugely successful website or app is solely due to the good user interface design. If not done properly it will fail to make any impact. If potential customers find the application, and in the current context POS credits repayment calculator, to be too complicated to navigate or too busy and confusing, then even the best service or product in the world could be overlooked.

Great user interface design should have a perfect balance of good looks and interactivity with ease of use and simple navigation.

Each page gets access to the data on the user's authorization and on that basis determines whether to allow the access. If the user is a guest and tries without registering or logging in first to use the pages that require authentication first (and these are practically all, except those for login and registration), he must be transferred to the login form. If a user with a specific user role tried to use a page that requires other roles, he must be informed of his mistake.

Depending on the functionalities that they provide (and hence the parameters required), there are two types of pages:

- Pages with direct access - to work they do not require submission of input arguments. If they are fed, they can serve to amend certain part of the form;
- Pages with indirect access - to operate these rules require to be submitted one or more input arguments (the primary key of a record of a table in the database).

The user experience with pages of direct access is through a navigation menu, and those with indirect access - by activating clicking buttons on the pages.

For better user experience Bootstrap is used.

Bootstrap

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive websites and applications. 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 the web development.

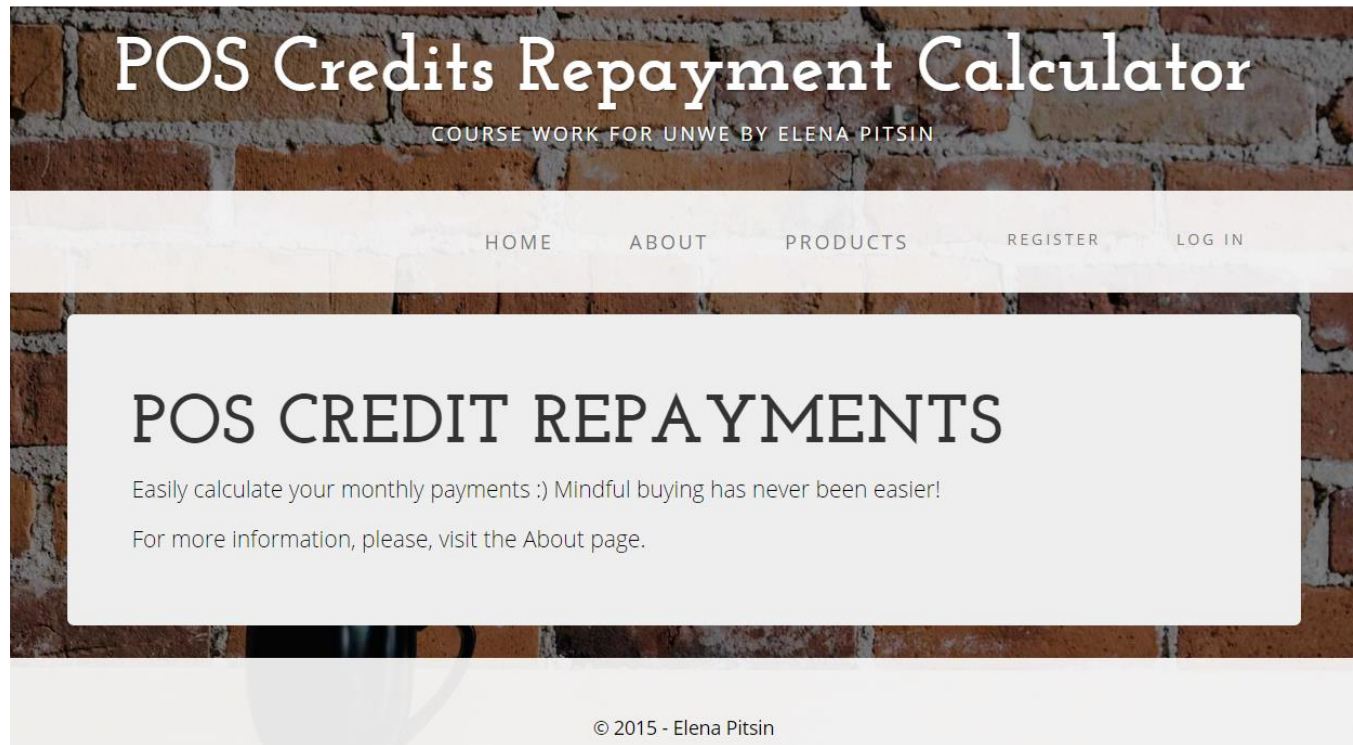
3.1 Menu design

Each user role can see different styles of the menu. Each style has its own functionalities and access rights. Here are all possibilities:

- Home
- About – some information about the application (even guest see that page)
- Products – a list with all products available where additional information can be seen and products can be bought (only for logged in users, only users and admins can buy products on credit)
- Profile – detailed information about the financial institution (only for financial institutions)
- Partners – a list with all financial institution where they can be approved and disapproved, their detailed information can be seen, edited and added (only for administrators)

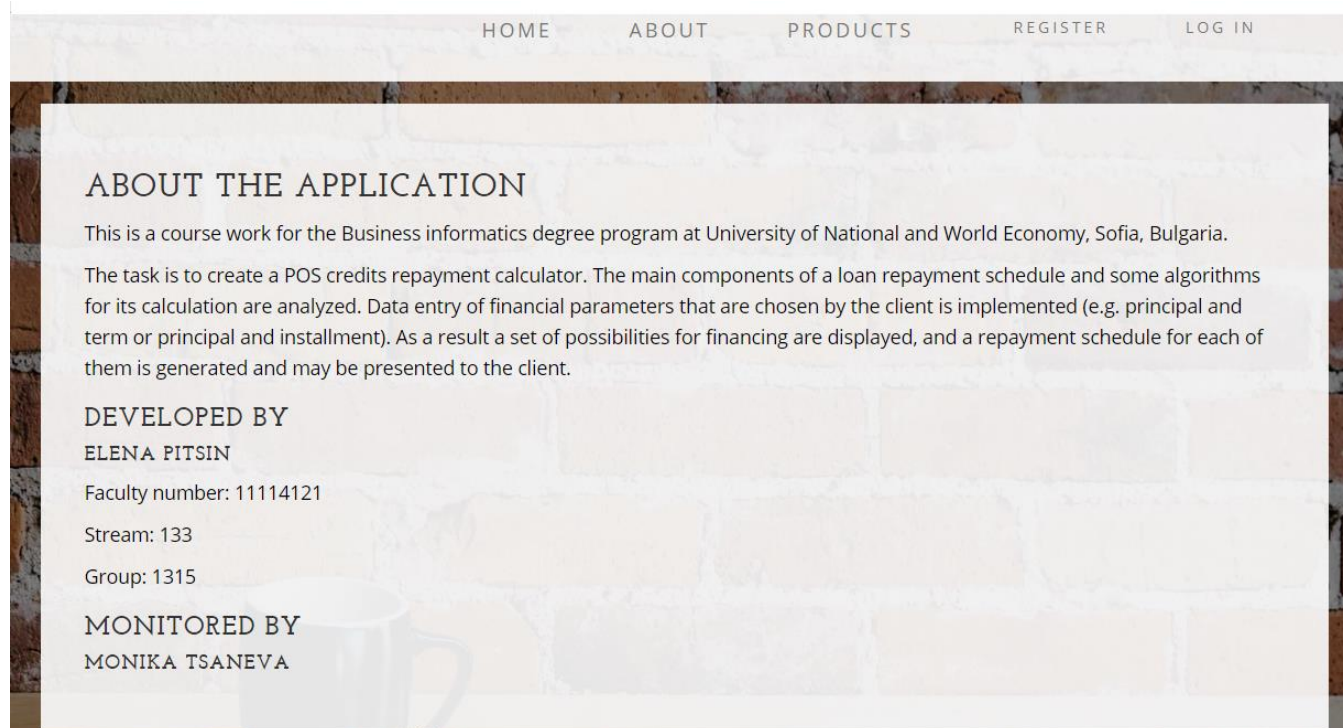
3.2 Interface for users who are guests of the application

Home



This is the Home page with is basically envisaged to be a place where news could be posted. For example the retailer can decide that he wants to display the new products there or some discounted ones, or maybe if they have interesting articles to share with their customers. It is completely up to the manager of the application. This page is left almost empty so that it could be further extended in the future.

About



This page's purpose is to give more information about the retailer's company or in this situation the task of the application and its creator. The About page is usually simple, straightforward, and it communicates the key things of the website or application. In many cases it is one of the most-visited pages on a site because it provides the best orientation of the new users.

Products and other pages

The guest doesn't have access to other pages and in order to take full advantage of the application, he has to register first. This is the page that he is redirected in case he wants to open a forbidden URL:

HOME ABOUT PRODUCTS REGISTER LOG IN

LOG IN.

USE A LOCAL ACCOUNT TO LOG IN.

Username

Password

☐ Remember me?

[Register as a new user](#)

USE ANOTHER SERVICE TO LOG IN.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

© 2015 - Elena Pitsin

In this case the user can either log in with his username and password or go to the register page and join.

REGISTER.

CREATE A NEW ACCOUNT.

Username

Email

Password

Confirm password

Are you a financial institution? ☒

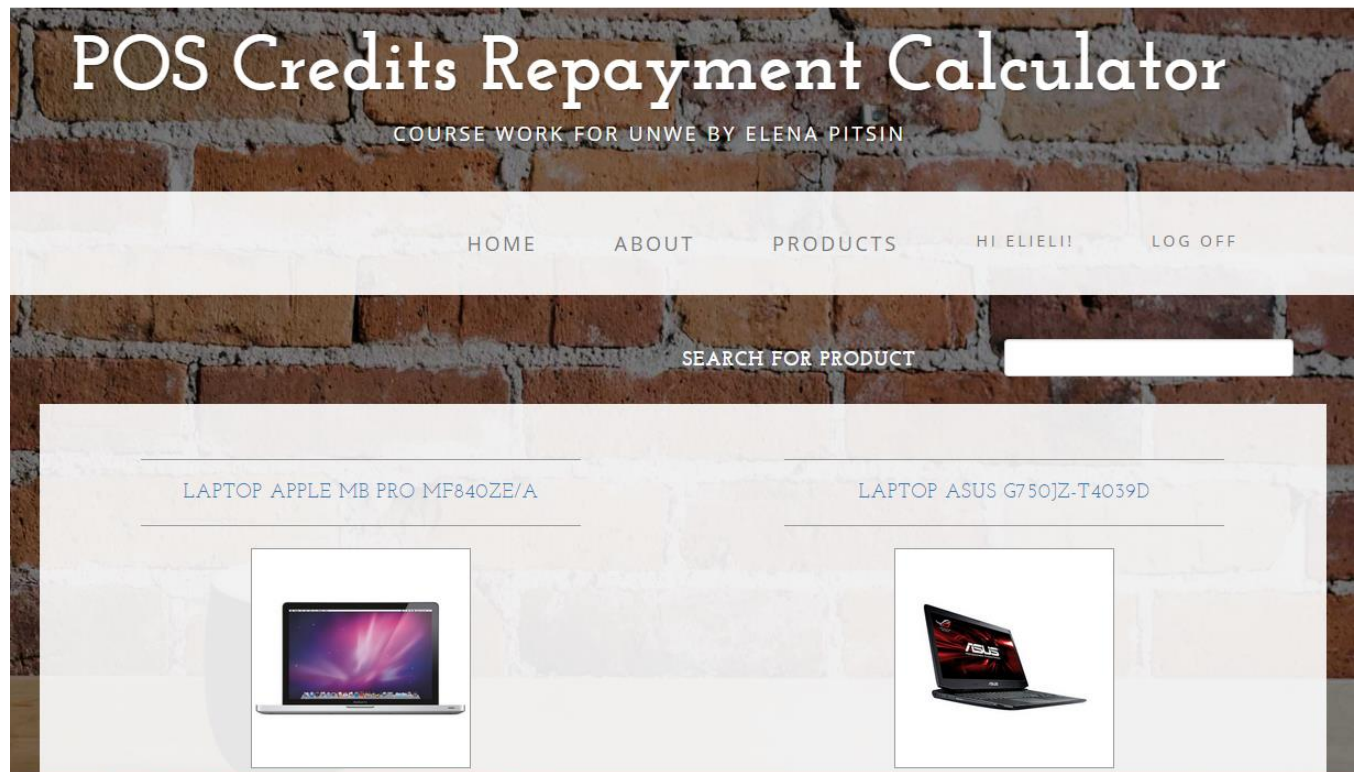
Business Name

As already mentioned before, for financial institutions there is a checkbox which on click shows the field for business name below.

3.3 Interface for users who are buyers

Registered or logged in users have much more flexibility and features to explore.

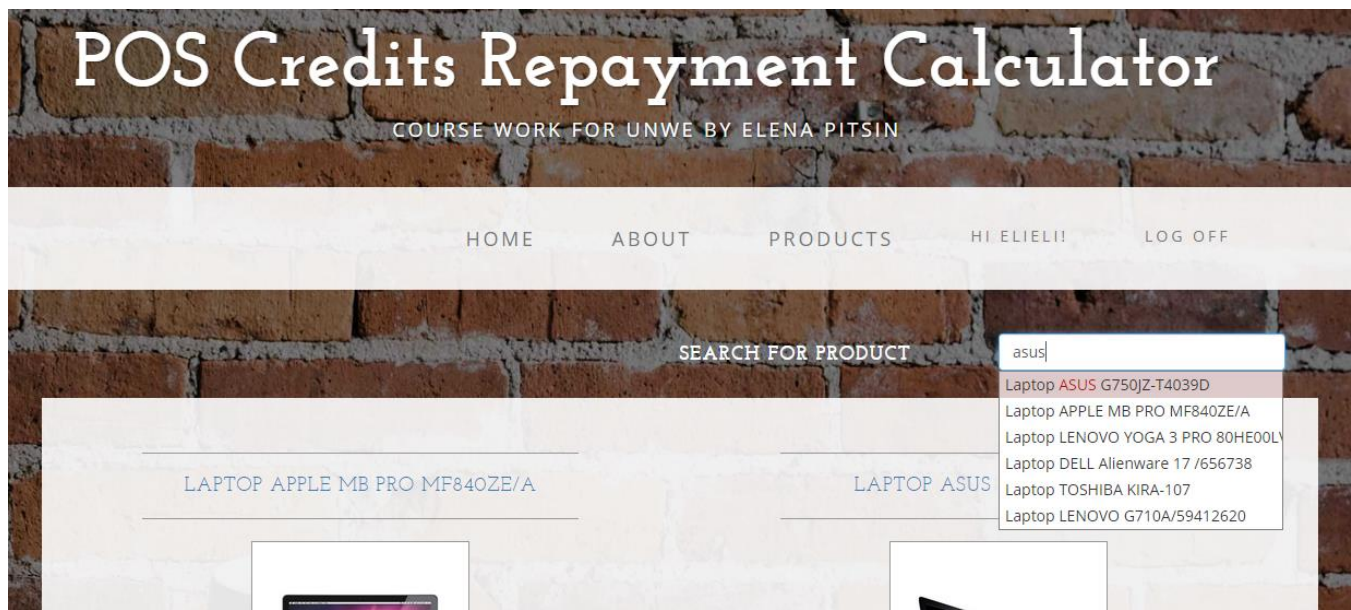
Products



This is the Products page from the menu that users can visit. There is a list with all products available. If the retailer using the application has more than one type of goods that he offers, this page could be separated to different categories and each category to have a list of the products related. However, in this case there is no need for category separation as the only good that this application offers is laptop.

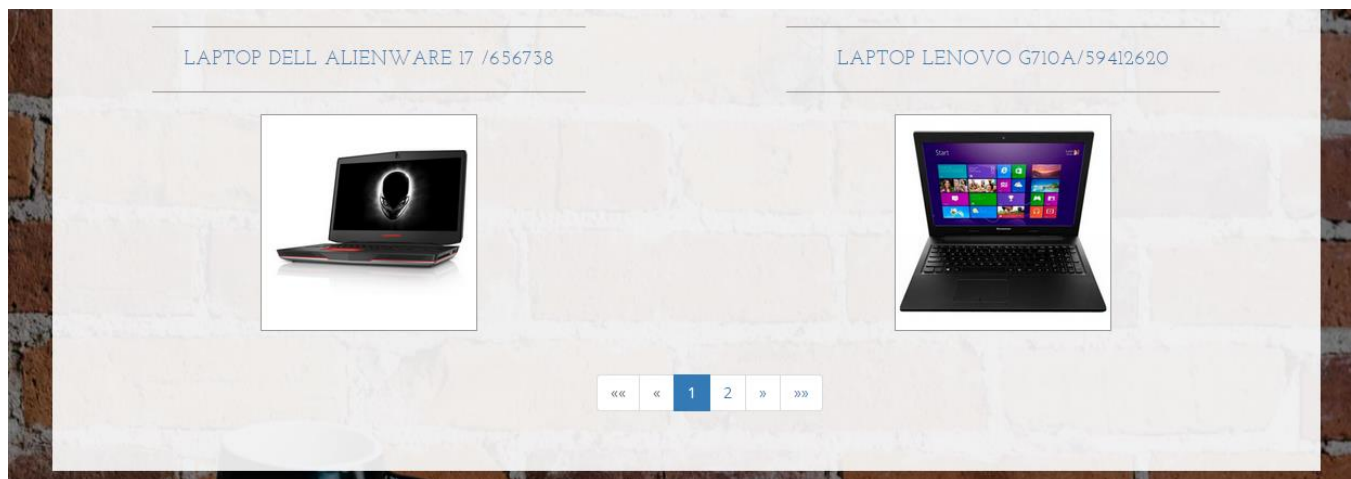
The best user design for such listing of items is to display as little information as possible so that the user doesn't get confused from all the unnecessary details. For this project, only the name and the picture of the laptops are chosen to be displayed as one of the main characteristics.

This doesn't mean that even with only one type of good, there can't be too many and difficult for the users to find the one they are looking for. Therefore, a good solution is to implement a search field which filters the laptops and highlights the one that matches the description.



As shown in the picture, the matching letters are colored in red and when highlighted the whole row changes its background color.

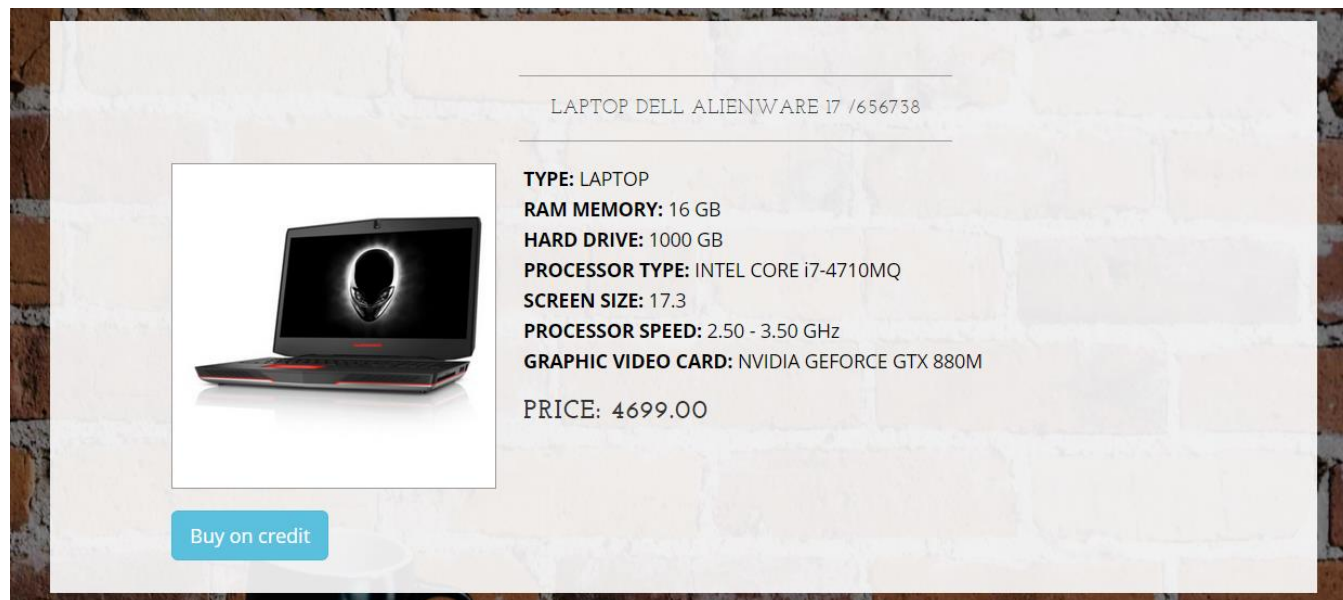
There is one more feature that is implemented in order to make the user experience even more friendly and natural – paging. Instead of scrolling down tens or hundreds of products, it's much more convenient to select pages and go through the products like that.



There are not only numbers representing each page but also Next/Previous and First/Last buttons when products are more. The current number of laptops that a page has is 4 but this can be adjusted to the total number of available products.

Product Details

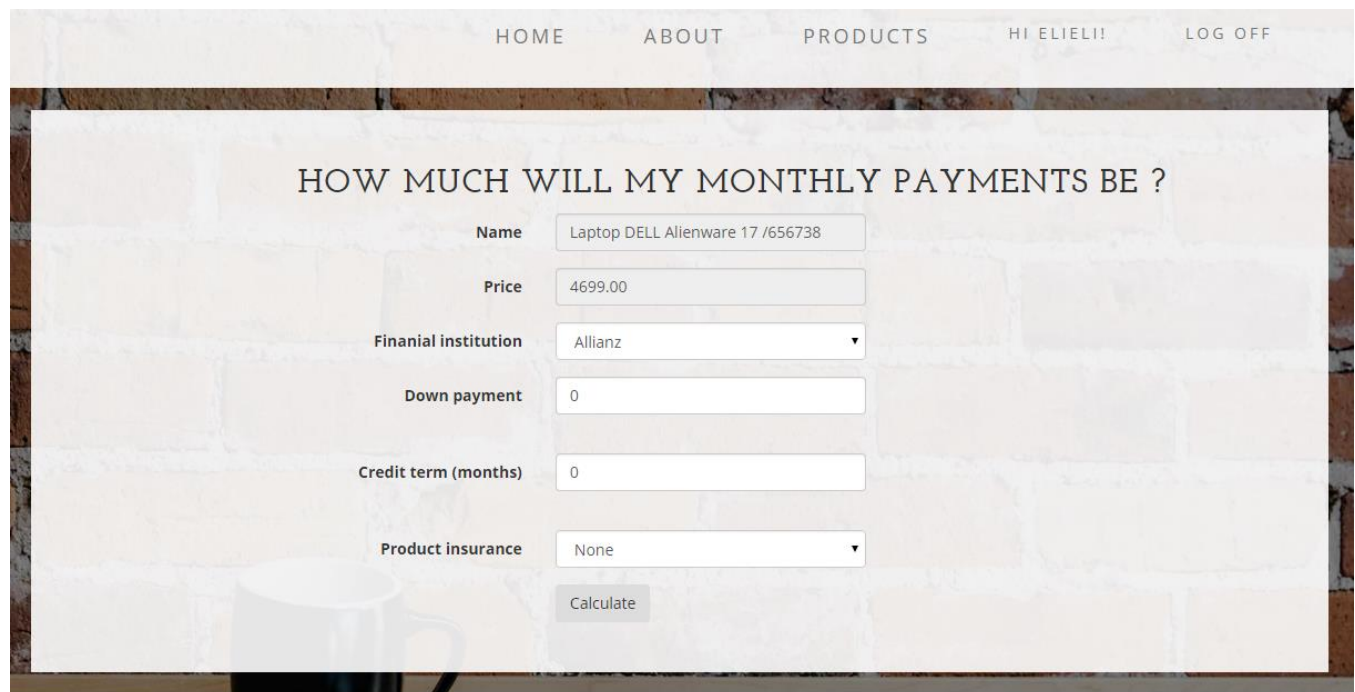
This Products page with direct access also leads to the indirect page Product Details. The user can go there either by clicking on one of the listed laptops (both name and picture are links), or he can search for a laptop and click on the one that matches. This will redirect to this page with details about the laptop:



As shown in the picture, the idea of this page is to display all characteristics of the product so that it can inform the potential customer what exactly he is buying. This page is very likely to be modified a lot especially if there are different categories with products. This is just one simple suggestion with few details of what it might look like.

Buy on credit

If a customer decides that he wants to buy a certain product based on its characteristics and price and of course after he clicks on the Buy on credit button, he is redirected to the very core of the calculation process.



HOME ABOUT PRODUCTS HI ELIELI! LOG OFF

HOW MUCH WILL MY MONTHLY PAYMENTS BE ?

Name

Price

Financial institution

Down payment

Credit term (months)

Product insurance

This is the place where the user can enter the data that is suitable for him and he can try different versions with different financial institutions and terms. All fields are absolutely necessary for the calculation of the repayments. Only the down payment can be 0 if the customer is not willing to pay money up front.

Calculate credit payments

After the customer has entered all the necessary data, the credit repayments are calculated and he is presented with all the details on the credit. The information should be elaborate but also very concise and well organized.

YOUR CALCULATED MONTHLY PAYMENT IS:

Financial institution	UniCredit
Product name	Laptop APPLE MB PRO MF840ZE/A
Product price	3339.00
Total amount to pay (principal + interest)	3444.40
Amount of the credit paid	3020.00
Amount of the interest paid	424.40
Down payment	339
Credit term (months)	12
Interest rate (% per month)	2.08
Interest rate (% per year)	25
Product insurance	None
Insurance amount	0.00
Monthly payment	287.03
Annual Percentage Rate of Charge (APR in %)	28.07

[Download PDF](#)

As clear from the picture above, this page has to provide answers to all questions that the customer may have about the credit such as the amount of the credit or the interest only. This data is very useful when it is displayed in this way because the user can easily compare the current credit with other credits and choose the best option for his needs. It is very important to show the Annual Percentage Rate APR as it is one of the major indexes when dealing with credits. The user interface is very minimalist so that the user cannot get confused with all the percentages and amounts.

Download PDF

If the user is happy with the credit and its terms and conditions, he can go to the next step and download a PDF file with the same data but in a standardized European format. Here is a part of the user interface of how this PDF, filled with information from the previous steps, should look like.

PART II. OTHER IMPORTANT CONDITIONS OF THE CREDIT CONTRACT

1. Credit Type	Consumer credit for acquiring goods and / or services.
2. Credit amount(Maximum amount(limit) or the whole sum, provided on account of the credit contract)	3020.00
3. Credit absorption conditions(How and when is the money received?)	The loan funds are transferred from the creditor to the account of the seller of the goods / insurer chosen by the user.
4. Term of contract	12
5. Indications of the amount, number, terms and dates of the repayments	<p>You must pay the following:</p> <ul style="list-style-type: none"> -downpayment: 339 -amount of each payment: 287.03 -number of payments: 12 -maturity date: 06/07/2016 18:18:25 <p>The interests and / or the costs of the credit are due as follows:</p> <ul style="list-style-type: none"> -by bank account of UniCredit -by pay-desk of UniCredit -by e-pay / b-pay
6. The whole sum that you have to pay(the whole amount of the credit(principal) including the interests and costs, which may originate in connection with your credit)	3444.40
7. When it is applicable for the specified type of credit: When the credit is provided under the form of deferred payment for acquiring a good or a service or is connected with the deliverance of a specific good or the provision of a service:	
Name of the good / service:	Laptop APPLE MB PRO MF840ZE/A
Price of the good / service in cash:	3339.00

3.4 Interface for users who are financial institutions

After registering as a financial institution, such user should go to their profile and add some more information about the company.

Profile

This page has two views – the first one is to simply view the profile details and the other one is to edit them.

Username	<input type="text" value="elieli"/>
Approved by admin	<input type="checkbox"/>
Name	<input type="text" value="eli eli"/>
Address	<input type="text"/>
Phone number	<input type="text"/>
Email	<input type="text" value="eli@eli.eli"/>
Fax	<input type="text"/>
Website	<input type="text"/>
Credit intermediary	<input type="text"/>
Application fee	<input type="text" value="0.00"/>

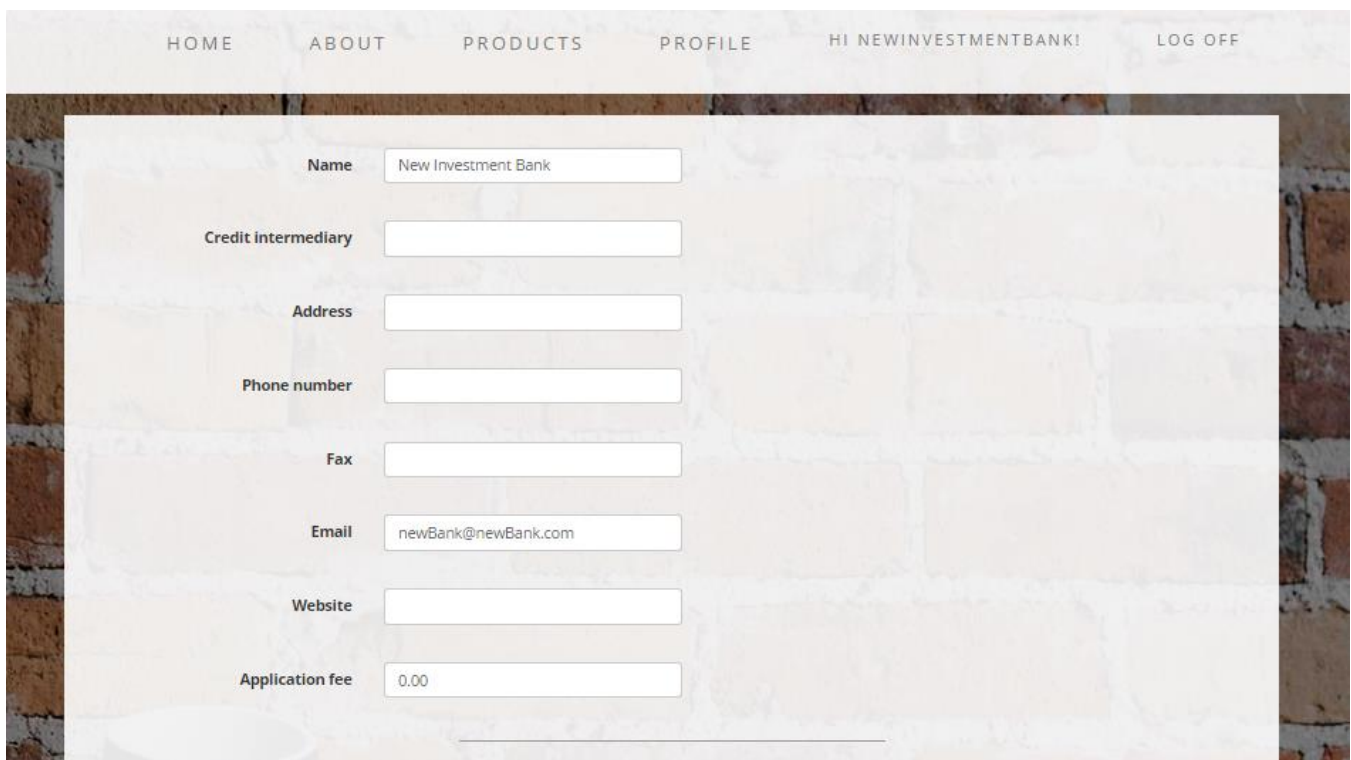
	INTEREST RATES
Credit term: 3 - 12 months Price range: 100.00 - 2000.00	<input type="text" value="0"/>
Credit term: 3 - 12 months Price range: 2000.00 - 100000.00	<input type="text" value="0"/>
Credit term: 13 - 24 months Price range: 100.00 - 2000.00	<input type="text" value="0"/>
Credit term: 13 - 24 months Price range: 2000.00 - 100000.00	<input type="text" value="0"/>
Credit term: 25 - 36 months Price range: 100.00 - 2000.00	<input type="text" value="0"/>
Credit term: 25 - 36 months Price range: 2000.00 - 100000.00	<input type="text" value="0"/>

INSURANCE RATES	
All	0.1
Life	0.03
LifeAndUnemployment	0.05
None	0
Purchase	0.03
Unemployment	0.03

Edit

This is the default profile that financial institutions see when they register in the application. Most of the fields are empty or have default value 0. The approved by admin checkbox is also unclicked as no one has approved this particular financial institution, yet. Fields have been kept to a minimum, to only the ones that are crucial for the calculation of the credit repayments, so that the design is simpler and easier to use. Most times when users see too many fields that need to be filled, they either oversee them or skip most that they feel are irrelevant.

Once they click the edit button, a new view is shown. The first part contains the fields about the institution's contacts and details.



The screenshot shows a web application interface with a navigation bar at the top containing links: HOME, ABOUT, PRODUCTS, PROFILE, HI NEWINVESTMENTBANK!, and LOG OFF. Below the navigation bar is a form with a light beige background and a subtle brick pattern. The form contains the following fields:

Field Label	Value
Name	New Investment Bank
Credit intermediary	
Address	
Phone number	
Fax	
Email	newBank@newBank.com
Website	
Application fee	0.00

All of these details are later going to be used in the Euro form PDF file. Therefore, it's obligatory for the financial institution to fill them in.

The financial institution is also very important to be able to update their interest and insurance rates. Therefore a quick and easy user interface is vital in this situation.

INTEREST RATES

Credit term: 3 - 12 months Price range: 100.00 - 2000.00	<input type="text" value="0"/>
Credit term: 3 - 12 months Price range: 2000.00 - 100000.00	<input type="text" value="0"/>
Credit term: 13 - 24 months Price range: 100.00 - 2000.00	<input type="text" value="0"/>
Credit term: 13 - 24 months Price range: 2000.00 - 100000.00	<input type="text" value="0"/>
Credit term: 25 - 36 months Price range: 100.00 - 2000.00	<input type="text" value="0"/>
Credit term: 25 - 36 months Price range: 2000.00 - 100000.00	<input type="text" value="0"/>

The ability to edit the credit terms and price ranges is available only for the administrators for the moment. It is important, though, in the future to implement this feature for the financial institutions as well. This is something which will bring competitive edge to the calculation service as no other competitor has this kind of feature.

INSURANCE RATES

All	<input type="text" value="0.1"/>
Life	<input type="text" value="0.03"/>
LifeAndUnemployment	<input type="text" value="0.05"/>
None	<input type="text" value="0"/>
Purchase	<input type="text" value="0.03"/>
Unemployment	<input type="text" value="0.03"/>

3.5 Interface for users with admin rights

This part is also vital for the whole application as approving and disapproving financial institutions can make this particular calculator more preferred and trusted compared to other calculators.

Partners

HOME	ABOUT	PRODUCTS	PARTNERS	HI ADMIN!	LOG OFF
PARTNER FINANCIAL INSTITUTIONS					
Name			Approved		
Allianz			<button>Disapprove</button>		
elitoto			<button>Disapprove</button>		
FiBank			<button>Disapprove</button>		
New Investment Bank			<button>Approve</button>		
UniCredit			<button>Disapprove</button>		

This page offers a list of all registered financial institutions. It has one of the most important jobs – to approve and disapprove them. By clicking on the “Approve”/”Disapprove” an Ajax request should respectively change the institution’s information in the database. By doing so the checkbox Approved by admin in the financial institution’s profile will be edited and this is how it is going to know whether they can participate in the calculator’s invertors.

The name of the institutions are links which lead to the profile information (already explained above) where the administrator can also edit the company’s details. The difference between the changes that the financial institution can make to its profile and the administrator can make are that the admins can alter the credit terms and the price ranges of the purchase profiles. Otherwise, the interface is the same.

INTEREST RATES

MonthsMin

3

MonthsMax

12

PriceMin

100.00

PriceMax

2000.00

InterestRate

26

MonthsMin

3

MonthsMax

12

PriceMin

2000.00

PriceMax

100000.00

InterestRate

25

Realization

1. APPLICATION TYPE AND CHARACTERISTICS

Choosing the type of the application is determined largely by the availability of different types of users and the need to provide them with quick and easy access. This condition implies the database and application logic to be implemented on a centralized basis and providing access to them becomes the Internet. Thus, the client portion remains only presentational logic and partial control of the input data. It could be implemented as a desktop application, but the inconvenient distribution and installation of programs from every user, as well as difficulties for use by different computers are significant.

This leads to the choice of creating a web application that communicates with the database to control user input determines output and gives users access to the realized forms through a web browser.

1.1. Choice of the platform, programming language and developing environment

The platform that the POS Credits Repayment Calculator project is designed on should be compatible with the database system and be the most appropriate for developing web applications. This will ensure the quality, security and speed of the application and also help achieve the business goals.

At this moment, probably the best platform that corresponds to the abovementioned criteria is ASP.NET developed by Microsoft. Microsoft is one of the leaders in the software business and constantly updates their products in order to make them better for the final customers and the developers. With ASP.NET they have managed to provide an opportunity for software engineers to build web applications and dynamic websites which require less coding than other alternatives. The applications are also not only fast to develop but also easy to setup and deploy, easy to debug (thanks to the Internet Information Services IIS) and very secure. Before displaying or

saving data from the users, ASP.NET provides excellent data-validation and authentication not only with the client side but also the backend (using inheritance, polymorphism, encapsulation and so on).

Recently, Microsoft's agenda has been focused on persuading the web development community to use ASP.NET MVC. This framework is open source and also is used for development of web applications. Its name comes from the fact that it allows developers to build their applications on the basis of three components – Model, View, Controller.

Here are the reasons why ASP.NET MVC has been chosen for this particular project:

- The framework gives full control over the HTML which is rendered
- Provides clear separation of concerns (SoC)
- Very easily tested and predisposes to use Test Driven Development (TDD)
- Highly extensible (usually used with Inversion of Control (IoC) containers), even view engines can be switched
- Rich UI support and easy integration with JavaScript frameworks (for example jQuery UI)
- RESTful URLs that enable SEO by design, follows the design of stateless nature of the web
- No ViewState andPostBack events
- Supports mitigating antiforgery attacks and XSS vulnerability

Another thing that could be considered as an advantage is that the SQL Server is developed by Microsoft as well and therefore the combination of SQL Server and ASP.NET MVC is very powerful and of course entirely compatible. Visual Studio IDE also contributes to the easy development of the applications using ASP.NET MVC.

1.2. Choice of the database management system

ASP.NET provides many options for storing, retrieving, and displaying data. In order to do that it uses data provider software such as Microsoft SQL Server, MySQL, Oracle, etc.

In the current project, there are no special needs that dictate a different database from the default one which is SQL Server.

Microsoft SQL Server is one of the major and most popular database management systems. As one of the leaders it offers incredible performance and scalability, ability to work with large databases and good security. The databases can be easily backed up or automatically recovered in case of system failure. But the strongest argument in this case is its integration with other Microsoft data access technologies like ASP.NET, Entity Framework and Visual Studio IDE.

Here is an example of how easy it is to connect ASP.NET with SQL Server:

```
1. <connectionStrings>
2.   <add name="DefaultConnection" connectionString="Data Source=(LocalD
   b)\v11.0;Initial Catalog=PosCreditsRepayment;Integrated Security=True"
   providerName="System.Data.SqlClient" />
3. </connectionStrings>
```


2. DATABASE REALIZATION

2.1 Creating 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.

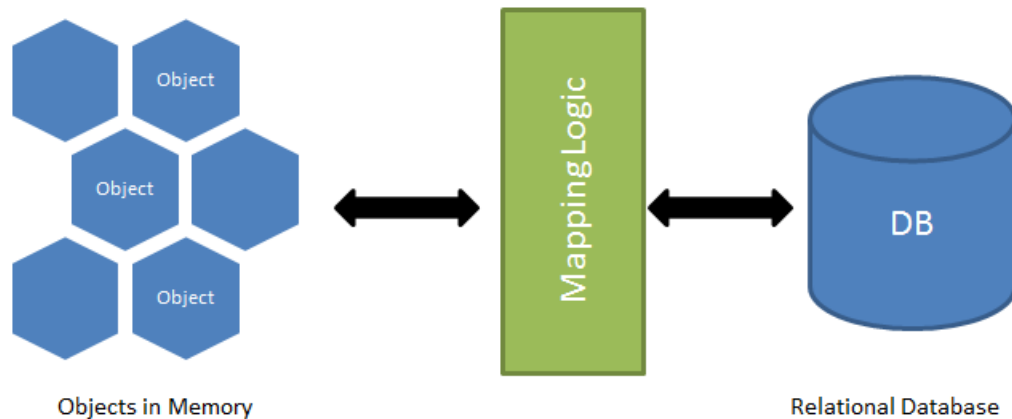
In other words, instead of something like this:

```
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;
```

O/R Mapping



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 than 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.

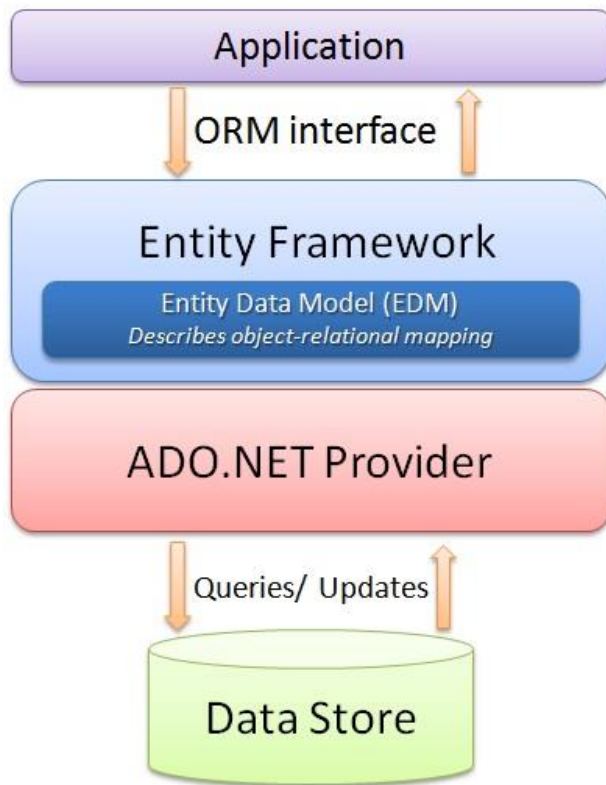
Here are several reasons why using an ORM tool and specifically Entity framework can be very advantageous:

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 which auto-generate models from an existing database and also visually create entity models. New databases can be deployed from a model, which can also be hand-edited for full control
- Provides a Code First experience to create entity models using code. Code 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 Communication 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.         thisAutomaticMigrationsEnabled = true;  
4.  
5.         // TODO: turn off data loss in release mode  
6.         thisAutomaticMigrationDataLossAllowed = true;  
7.         this.ContextKey = "POSCreditRepayments.Data.POSCreditRepay  
    mentsDbContext";  
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

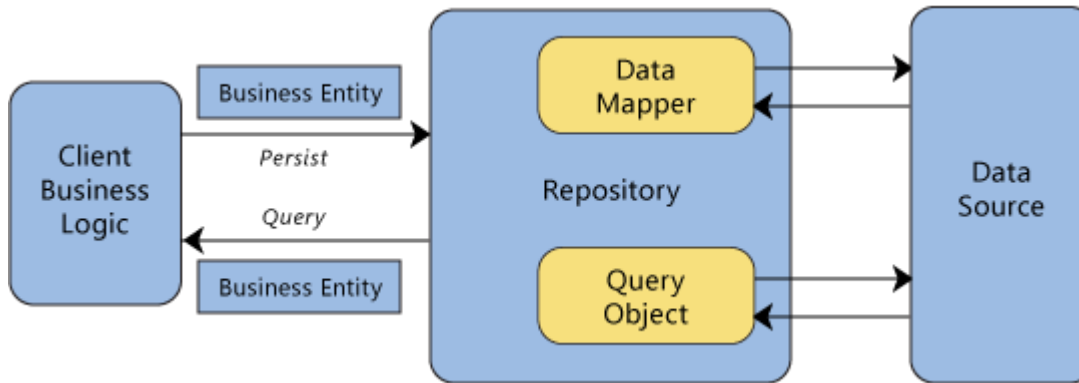
The solution is to simply use repository pattern to separate the business logic that acts on the model from the logic that retrieves the data and maps it to the entity model. The business logic should be completely apart from the type of data that makes the data source layer. For example, as already 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 persists 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 overall design evolve
- It centralizes the data logic or Web service access logic
- Easily testable with unit tests

The repository can query business entities in two ways. The first is to use methods that specify the business criteria and the second is to submit a query object to the client's business logic. In the first case, the repository forms the query on the client's behalf. The repository returns a matching set

of 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 of Work pattern (it is described below). This pattern demonstrates how to encapsulate several related operations that should be consistent with each other or that have related dependencies. The encapsulated items are sent to 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, into a domain where objects are strongly typed, such as a domain entity model. One example is a database that uses `IDbCommand` objects to execute queries and returns `IDataReader` objects. A repository issues the appropriate queries to the data source, and then it maps the result sets to the externally exposed business entities. Repositories often use the Data Mapper pattern to translate between representations. Repositories remove dependencies that the calling clients have on specific technologies. For example, if a client calls a catalog repository to 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 of dependencies 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);
4.
5.         IQueryable<T> All();
6.
7.         void Delete(T entity);
8.
9.         void Delete(object id);
10.
11.         T GetById(object id);
12.
13.         int SaveChanges();
14.
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 `IsDeleted = 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.         }
8.     }
```



```
9.         protected IPOSCreditRepaymentsDbContext Context { get; set; }

10.

11.         protected IDbSet<T> Set { get; set; }

12.

13.         public void Add(T entity)
14.         {
15.             this.ChangeState(entity, EntityState.Added);
16.         }

17.

18.         public IQueryable<T> All()
19.         {
20.             return this.Set;
21.         }

22.

23.         public void Delete(T entity)
24.         {
25.             this.ChangeState(entity, EntityState.Deleted);
26.         }

27.

28.         public void Delete(object id)
29.         {
30.             var entity = this.GetById(id);
31.
32.             if (entity != null)
33.             {
34.                 this.Delete(entity);
35.             }
36.         }

37.

38.         public T GetById(object id)
39.         {
40.             return this.Set.Find(id);
41.         }

42.

43.         public int SaveChanges()
44.         {
45.             return this.SaveChanges();
46.         }

47.
```

```
48.         public void Update(T entity)
49.         {
50.             this.ChangeState(entity, EntityState.Modified);
51.         }
52.
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 typeOfRepository = typeof(T);
4.
5.         if (!this.repositories.ContainsKey(typeOfRepository))
6.         {
7.             var repositoryType = typeof(Repository<T>);
8.
9.             var newRepository = Activator.CreateInstance(repository
    Type, this.context);
10.
11.             this.repositories.Add(typeOfRepository, newRepository);
12.         }
13.
14.         return (IRepository<T>)this.repositories[typeOfRepository];
```

```
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 - theObjectContext 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.

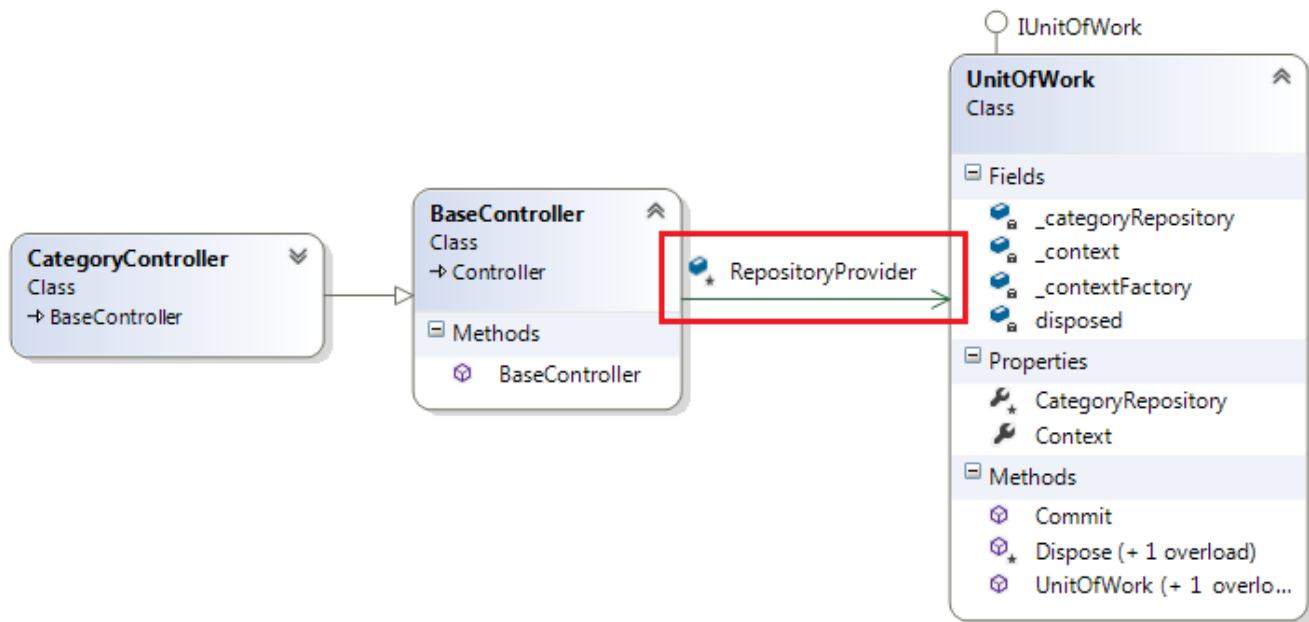
In cases when there is a need to write custom application-specific Unit of work interface or class that wraps the inner Unit of work, this can also be done easily and there are a number of reasons why this is a good idea.

- 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; }
4.
5.     IRepository<FinancialInstitutionPurchaseProfile> FinancialInst
6.     itutionPurchaseProfiles { get; }
7.
8.     IRepository<FinancialInstitution> FinancialInstitutions { get;
9.     }
10.
11.     IRepository<Product> Products { get; }
12.
13.     IRepository<PurchaseProfile> PurchaseProfiles { get; }
14.
15.     IRepository<User> Users { get; }
16.
17.     int SaveChanges();
18. }
  
```

And the concrete implementation:

```
1. public class POSCreditRepaymentsData : IPoSCreditRepaymentsData
2.     {
3.         private readonly IPoSCreditRepaymentsDbContext context;
4.
5.         private readonly IDictionary<Type, object> repositories;
6.
7.         public POSCreditRepaymentsData(IPoSCreditRepaymentsDbContext c
            ontext)
8.         {
9.             this.context = context;
10.            this.repositories = new Dictionary<Type, object>();
11.        }
12.
13.        public IRepository<Credit> Credits
14.        {
15.            get
16.            {
17.                return this.GetRepository<Credit>();
18.            }
19.        }
20.
21.        public IRepository<FinancialInstitutionPurchaseProfile> F
            inancialInstitutionPurchaseProfiles
22.        {
23.            get
24.            {
25.                return this.GetRepository<FinancialInstitutionPur
                    chaseProfile>();
26.            }
27.        }
28.
29.        public IRepository<FinancialInstitution> FinancialInstitu
            tions
30.        {
31.            get
32.            {
33.                return this.GetRepository<FinancialInstitution>()
            ;
34.        }
```

```
34.         }
35.     }
36.
37.     public IRepository<Product> Products
38.     {
39.         get
40.         {
41.             return this.GetRepository<Product>();
42.         }
43.     }
44.
45.     public IRepository<PurchaseProfile> PurchaseProfiles
46.     {
47.         get
48.         {
49.             return this.GetRepository<PurchaseProfile>();
50.         }
51.     }
52.
53.     public IRepository<User> Users
54.     {
55.         get
56.         {
57.             return this.GetRepository<User>();
58.         }
59.     }
60.
61.     public int SaveChanges()
62.     {
63.         return this.context.SaveChanges();
64.     }
65. }
```

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

The Repository and Unit of Work Patterns

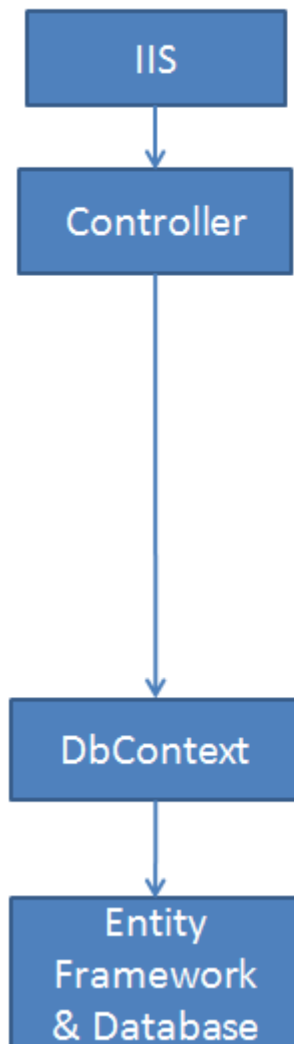
Both of these patterns have the goal to create an abstraction layer between the data access layer and the business logic layer of an application. Implementing Repository and Unit of work

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.

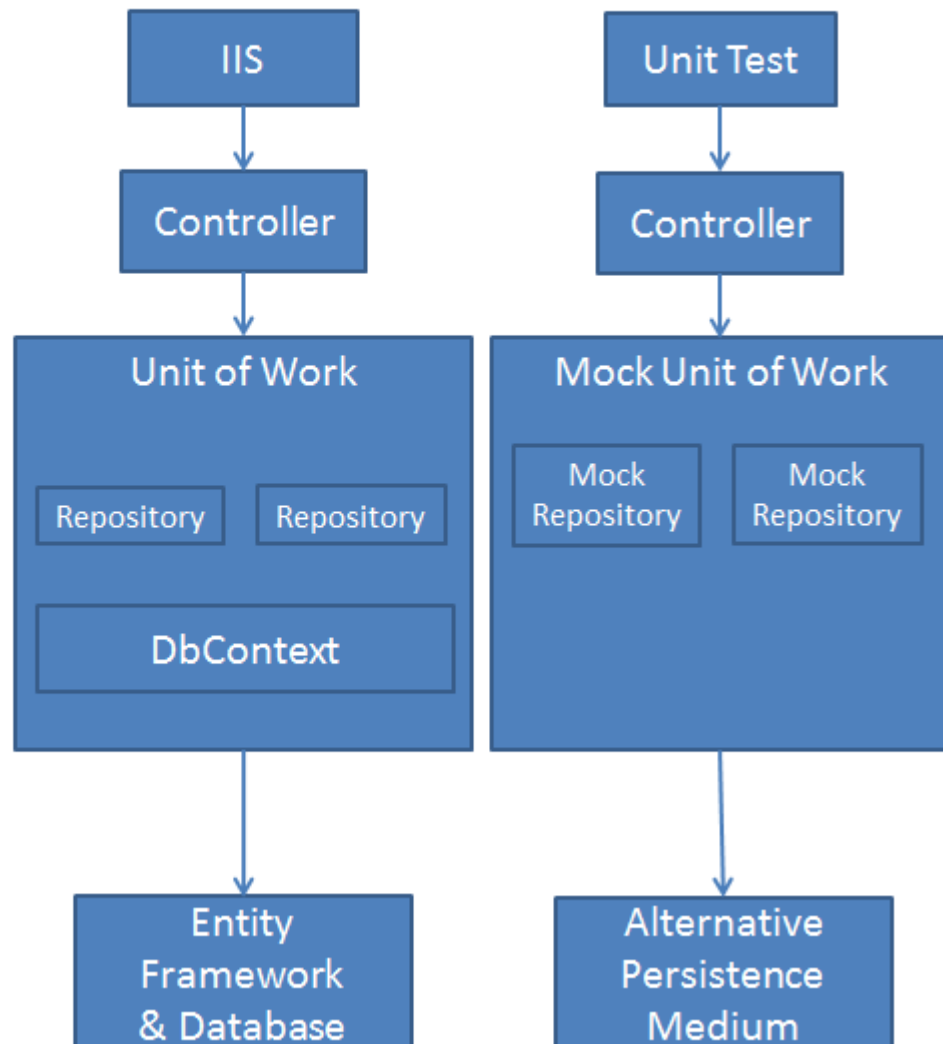
No Repository

Direct access to database context from controller.



With Repository

Abstraction layer between controller and database context. Unit tests can use a custom persistence layer to facilitate testing.



Database models and their relationships

Identity models

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 made 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 gives 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](#)

This package has the Entity Framework implementation of ASP.NET Identity which will persist the ASP.NET Identity data and schema to SQL Server.

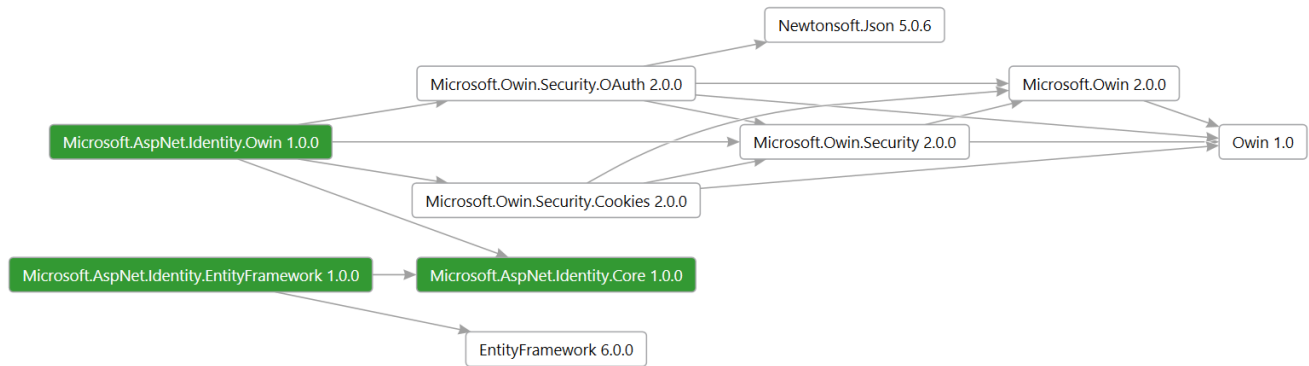
➤ [Microsoft.AspNet.Identity.Core](#)

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.

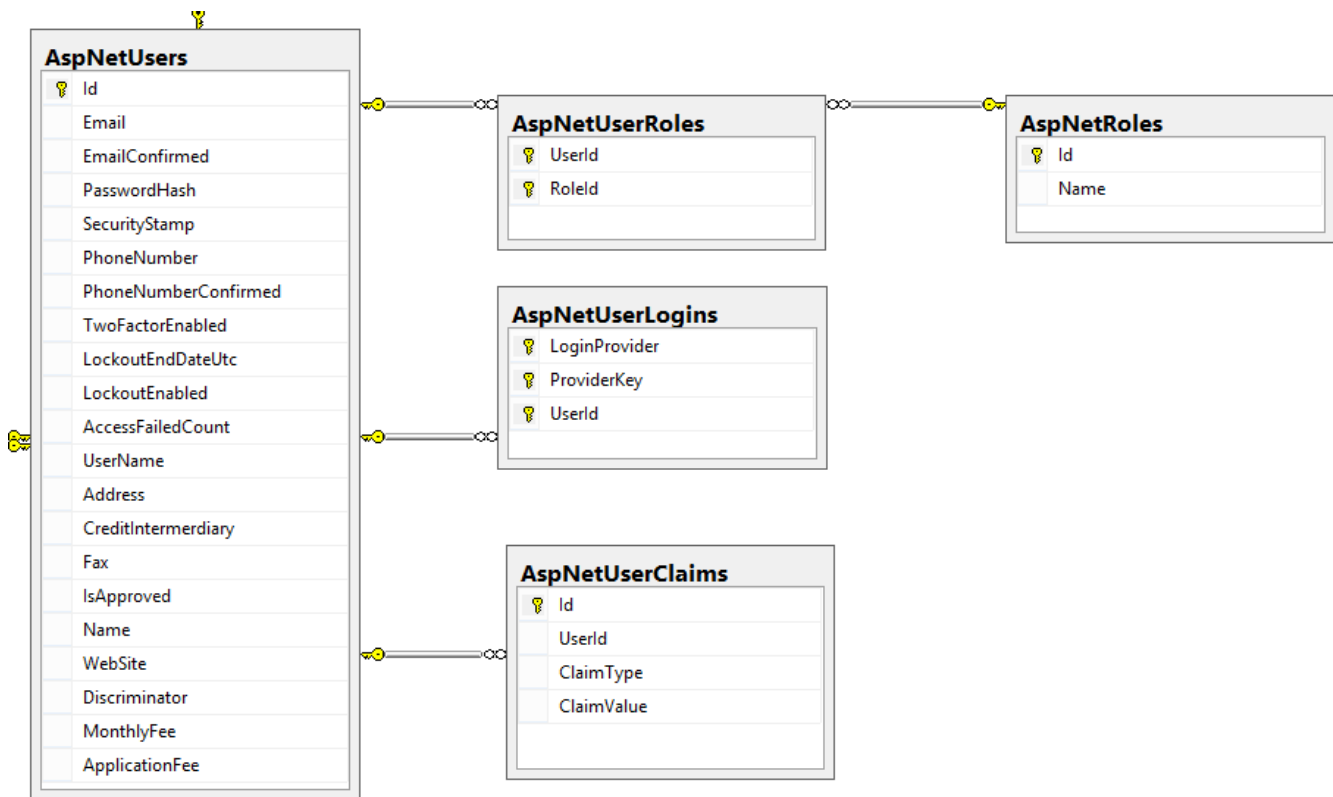
➤ [Microsoft.AspNet.Identity.OWIN](#)

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:

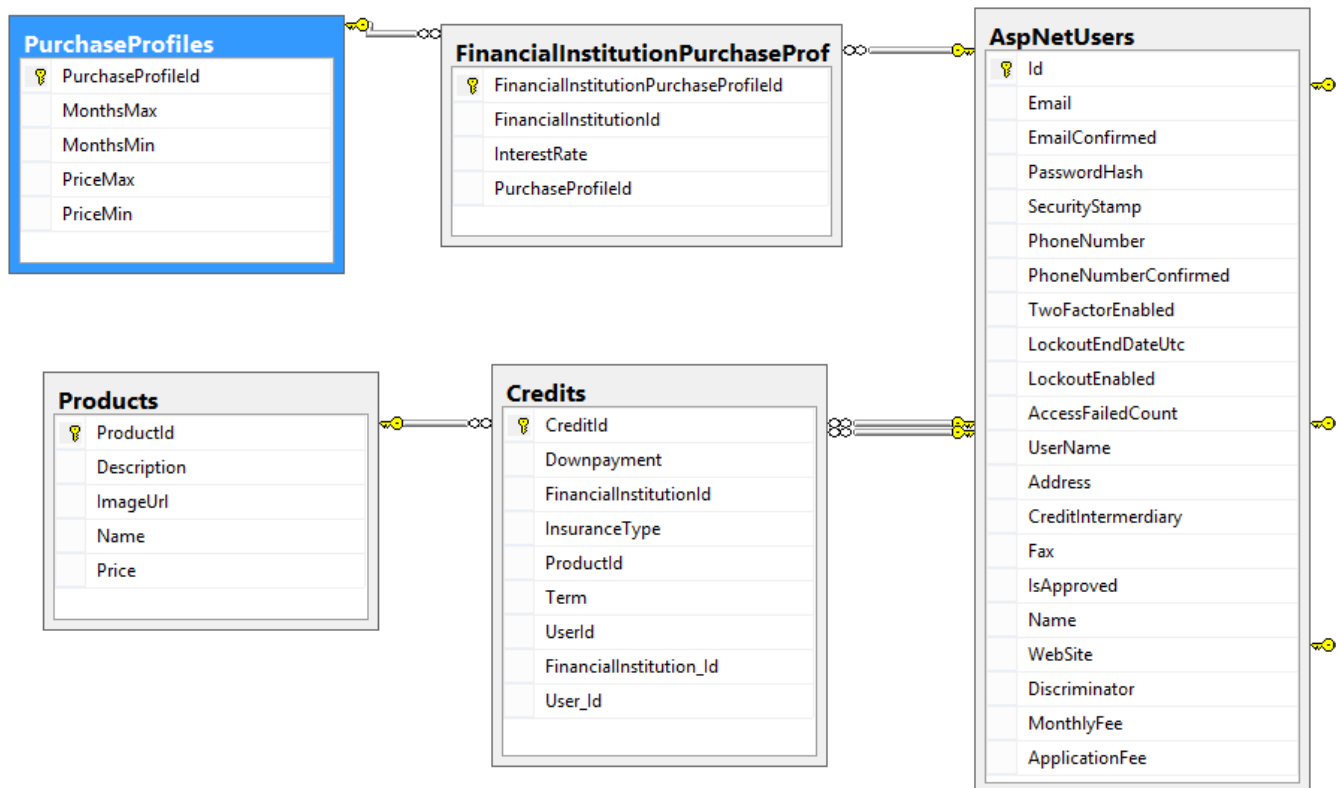


The tables AspNetUserRoles, AspNetRoles, AspNetUserLogins, AspNetUserClaims are the default tables credited by the Identity. AspNetUsers, however, is an extension from the default AspNetUsers. Probably, this is of the best features of the ASP.NET Identity - the user profile and information can be easily 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.

Custom models

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.     }
7.
8.     public virtual ICollection<Credit> Credits { get; set; }
9.
10.    public async Task<ClaimsIdentity> GenerateUserIdentityAsync
    c(UserManager<User> manager)
11.    {
12.        // Note the authenticationType must match the one defi
    ned in CookieAuthenticationOptions.AuthenticationType
13.        var userIdentity = await manager.CreateIdentityAsync(t
    his, 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<Fi
    nancialInstitutionPurchaseProfile>();
7.             this.Insurance = new HashSet<Insurance>();
8.         }
9.
10.        public string Address { get; set; }
```

```
11.
12.         public decimal ApplicationFee { get; set; }
13.
14.         public string CreditIntermerdiary { get; set; }
15.
16.         public virtual ICollection<Credit> Credits { get; set; }
17.
18.         public string Fax { get; set; }
19.
20.         public virtual ICollection<FinancialInstitutionPurchasePr
    ofile> FinancialInstitutionPurchaseProfiles { get; set; }
21.
22.         public virtual ICollection<Insurance> Insurance { get; se
    t; }
23.
24.         public bool IsApproved { get; set; }
25.
26.         public decimal MonthlyFee { get; set; }
27.
28.         public string Name { get; set; }
29.
30.         public string PhoneNumber { get; set; }
31.
32.         public string WebSite { get; set; }
33.     }
```

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<Fi
nancialInstitutionPurchaseProfile>();
6.         }
7.
8.         public virtual ICollection<FinancialInstitutionPurchaseProfile
> FinancialInstitutionPurchaseProfiles { get; set; }
9.
10.        public int PurchaseProfileId { get; set; }
11.
12.        public int MonthsMax { get; set; }
13.
14.        public int MonthsMin { get; set; }
15.
16.        public decimal PriceMax { get; set; }
17.
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; }
4.
5.         public string FinancialInstitutionId { get; set; }
6.
7.         public int FinancialInstitutionPurchaseProfileId { get; set; }
8.

```

```
9.         public double InterestRate { get; set; }
10.
11.         public virtual PurchaseProfile PurchaseProfile { get; set
    ; }
12.
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.

Another data model is the Insurance class:

```
1. public class Insurance
2.     {
3.         public virtual FinancialInstitution FinancialInstitution { get
    ; set; }
4.
5.         public int FinancialInstitutionId { get; set; }
6.
7.         public int InterestRateId { get; set; }
8.
9.         public double PercentageRate { get; set; }
10.
11.         public InsuranceType Type { get; set; }
12.     }
```

It represents the percentage rate that each financial institution sets for a particular insurance type. It was mentioned in the first section that the insurance types applicable to this project are Life, Unemployment, Life and Unemployment, Purchase or all of them combined together.

```
1. public enum InsuranceType
2.     {
3.         None,
4.         Life,
5.         Unemployment,
6.         [Display(Name = "Life and unemployment")]
7.         LifeAndUnemployment,
8.         Purchase,
9.         [Display(Name = "Life, unemployment and purchase")]
10.        All
11.    }
```

Usually, financial institutions have a contract with an insurance company and have a fixed percentage rate for each category above. Therefore, the insurance class is connected with the financial institution with a relationship one-to-many (one financial institution can have many insurance types and rates for them).

Yet another data model is the Credit model which basically sums up the calculated values and the information for the credit in one place and stores each calculated POS credit repayment in the database for any future needs.

```
1. public class Credit
2.     {
3.         public int CreditId { get; set; }
4.
5.         public decimal Downpayment { get; set; }
6.
7.         public virtual FinancialInstitution FinancialInstitution { get
; set; }
8.
9.         public int FinancialInstitutionId { get; set; }
10.
11.        public Insurance Insurance { get; set; }
12.    }
```

```
13.         public virtual Product Product { get; set; }
14.
15.         public int ProductId { get; set; }
16.
17.         public int Term { get; set; }
18.
19.         public virtual User User { get; set; }
20.
21.         public int UserId { get; set; }
22.     }
```

Each credit is unique with its owner (buyer), the financial institution it belongs to and the product it is related to. There is also an option for an insurance of the product so this also exists in the class's properties. All relationships here are one to many – financial institutions can give many credits, buyers also can have many credits, products can be bought from several people again on credit.

And the last data model is the Product model, which is not mandatory for the calculation and it is introduced only for better user experience in the current application. This means that if a company wants to implement this calculator in their retail business and they don't sell products but rather some kind of services that can also be bought on credit, they are not obliged to use it and can easily replace it. However, this is how it looks like in the code:

```
1. public class Product
2. {
3.     public Product()
4.     {
5.         this.Credits = new HashSet<Credit>();
6.     }
7.
8.     public virtual ICollection<Credit> Credits { get; set; }
9.
10.    public string Description { get; set; }
11.
12.    public string ImageUrl { get; set; }
13.
14.    public string Name { get; set; }
15.
16.    public decimal Price { get; set; }
```



```
17.  
18.      public int ProductId { get; set; }  
19.      }
```

It's only reference is to the credit which relationship can easily be broken and replaced with another one.

3. APPLICATION REALIZATION

3.1. Structure of the application

The structure of the application is a common one. It has two layers – the database layer and the web application layer.

In the database layer reside all of the models (as tables) and their values. On the other hand, the web application stores how these models are presented to the user in form of views, templates and partials. The two communicate through LINQ queries translated into SQL queries. The queries search through the tables and retrieve the needed information which is then translated to the models saved in the Data project and then mapped to view models.

The queries (as already explained above) are automatically generated by Entity Framework, so there is no need to further explain them. The code in the web application, however, is very important to be written well and to be optimized to work as efficiently as possible.

Having said that, nowadays, it is more and more important to for developers to white code which is easily maintainable and extensible , with great focus on reusing existing components just so that a cohesive architecture can be formed. However, wiring disparate components can easily become a daunting task especially if the application expands and its complexity increases because together with them dependencies also become more difficult to handle.

The structure of the application is based on the principle of inversion of control design pattern:

Inversion of control

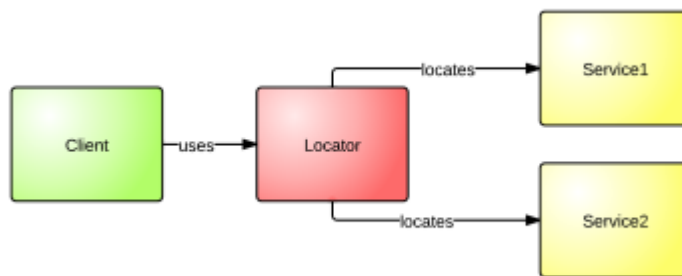
Inversion of control (IoC) is a design structure where custom-written classes receive the flow of control from another generic and most importantly reusable library. The difference between using this design and traditional programming is the following:

- Traditional programming - the custom code calls into reusable libraries to take care of generic tasks
- IoC - it is the reusable code that calls into the custom code.

The idea behind Inversion of Control is the use of the “Hollywood Principle” (another high quality code principle) – “No need to call us, we’ll call you.” IoC is used to increase modularity of the application and make it extensible and has applications in most object-oriented programming languages.

There are two possible implementations of Inversion of Control:

- **Service Locator:** each component is requested by using a locator API and it provides callback functionality and lookup context.



- **Dependency Injection (DI):** in comparison with the service locator there is no need for dependency to a container API. It basically manages the lifecycles (scope, request, thread), lookup and contextual binding and most of the cases is implemented with a DI framework.

For the purpose of this project, DI is chosen as the better implementation and pattern.

Dependency injection

The strongest argument for using Dependency Injection (DI) is that it allows you to inject objects into a class, instead of letting the class create and handle the object itself.

The famous Abstract design pattern implements DI with one of the most common ways to use it – with a factory class. If a component creates an instance of another class within itself, it implements the logic for initializing this class. However, this initialization logic is almost never reusable outside of the creating component, and therefore must be duplicated for any other class that requires an instance of the created class.

Although many developers still perform functions such as object construction and dependency resolution by hand, it is much more time-saving and efficient to automate these monotonous and menial tasks. This resolution of dependencies is simply the resolving of already defined dependencies. Dependency Injection, however, aims to reduce the amount of wiring and unnecessary code that developers have to write.

What's more, DI containers not only make developer's life easier but also provide a layer of abstraction to the project. They reduce the dependency coupling between two classes by providing generic factory classes that instantiate instances of classes instead of letting them do that by themselves. The container then configures these instances, which allows this logic connected with the construction of the dependency to be reused on a broader level.

The library that is used in the current project to implement dependency injection (instead of doing it by hand) is one of the most popular and widely used at the moment – Ninject.

Ninject

Ninject library is a very fast and lightweight dependency injector for .NET applications. Its idea is to try and help, to separate the application as much as possible into very cohesive and at the same time loosely coupled interfaces/classes, and then put them back together again but in a much more flexible way. This makes the code easier to write, extensible and very reuseable, and last but not least, testable.

Another good practice that is worth mentioning in this section as it is directly related to the structure of the application are the view models and how they are displayed/edited with the display and editor templates that ASP.MVC offers.

Why is it important to use View Models?

There are several very important reasons why it is absolutely necessary to use view models and not the database models when writing ASP.NET application.

- Logic in the views

Although using the domain model or the entity model also works fine, after a while some problems start to arise when logic is introduced into the views. This makes the code untestable and breaks the DRY principle – Don't Repeat Yourself.

➤ Security issues

Using view models ensures that the application is more secure – if the database model contains properties that shouldn't be changed, the user can't touch them unless they are displayed via the view model.

➤ Loose coupling

This is probably the worst thing that can happen – to couple the domain models with the presentation layer. If view models are skipped, the moment the entity is changed, all views that use this model should also be changed.

However, some developers might argue that mapping models to view models every single time is time-wasting and inefficient. This is why ASP.NET offers some very nice libraries which do this automatically, with very few modifications and effort.

Automapper

AutoMapper is one such object-to-object mapper, which solves the issues with mapping of the same properties in one object of one type to another object of another type.

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. And this is why this exact library was chosen for the current application.

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 there is a need to customize how data is presented to the user for displaying and for editing. Therefore, display and editor templates are here to save the day.

LabelFor() or TextBoxFor() display 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, it is not appropriate for whole view models or custom data fields. A very simple 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.

Often developers are wandering what is the template that they should use in a specific case – display, editor or partial view. As in this application, all three are present, here are some specifications why each one is used:

Display template or Editor template or Partial view

The difference between the first two is easy - EditorFor or DisplayFor. As the names suggest - the semantics is to generate edit/insert and display/read only views. DisplayFor is used when displaying data, the model's values that are saved in the database. On the other hand, EditorFor is used when editing/inserting data into forms or other ways and the new values are then saved into the database or used in the code to calculate something.

The above methods are model-centric. This means that they will take the model metadata into account (for example the model class can be even annotated 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.

On the other hand Partial is view-centric in that choosing the correct partial view is the main point. 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 there is a need to affect the behavior of this partial in which case an appropriate view model should be passed.

Ultimately the choice depends on what is it that is currently being modeled. Also nothing prevents mixing and matching the abovementioned templates. For example, there could be a partial view that calls the `EditorFor` helper.

3.2. Creation and initial settings of the application

First, in Visual Studio a new project is created of type Class Library where all the models of the database will reside. The project is separated in a folder called Data for better structure.

Then a Web Application in Visual C# is created as a new project where ASP.NET MVC is chosen and the authentication is left Individual authentication. This project is put in folder Web and contains the folders with the controllers, views, CSS styles and JavaScript scripts. Bootstrap takes care of almost all of the styles and necessary scripts but if the develop wants to customize the Bootstrap theme, he can add new files in the respective folders. The configuration of the application is in the `App_Start` folder where for example custom CSS files can be registered or the required password validator can be altered.

Some basic controllers like Home and Account are auto-generated by the ASP.NET together with their views. Each new View and ViewModel is put in another folder with the name of the controller and then the name of the action.

After that the `web.config` is modified to work with the current database:

```
1. <connectionStrings>
```

```
2. <add name="DefaultConnection" connectionString="Data Source=(LocalDb
   )\v11.0;Initial Catalog=PosCreditsRepayment;Integrated Security=True"
   providerName="System.Data.SqlClient" />
3. </connectionStrings>
```

And some additional settings are added, for example for the error handling of the application:

```
1. <system.web>
2.   <authentication mode="None" />
3.   <compilation debug="true" targetFramework="4.5" />
4.   <httpRuntime targetFramework="4.5" />
5.   <customErrors mode="On" defaultRedirect="/Home/Error" />
6. </system.web>
```

This make it possible to show a customized text or picture when there is an error in the application. So instead of showing a white and yellow page with the error message and the stack, it shows a more user friendly message that apologizes for the inconvenience. This usually happens when there is an unexpected server error. If the user is trying to access a page that he has no authorization for, he is redirected to the login page.

3.3. Creation of the pages for guest users

Guests basically can see the Home, About, Login and Register pages. If they try to access the Products page with a list of all products available, they will be redirected to the login page.

The Login and Register pages are the default ones that ASP.NET sets up when a new web application is created.

Login – “~/Account/Login”

Login has two input fields – one for the username and one for the password. They are wrapped up in a BeginForm with validation for security reasons and required fields.

The login will be successful after that input fields are filled and the data is checked with the database. If the username and the password exist and match with the ones stored, the user is

logged in. Of course there is also validation for empty or invalid fields and if the user hasn't logged in already. If the checkbox for remembering the user details is checked than there is no need for authentication every time.

The method that handles the login in the code is the following one:

```
1.      [AllowAnonymous]
2.      [ValidateAntiForgeryToken]
3.      public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
4.      {
5.          if (!ModelState.IsValid)
6.          {
7.              return View(model);
8.          }
9.
10.         var result = await SignInManager.PasswordSignInAsync(
            model.UserName, model.Password, model.RememberMe, shouldLockout: false
        );
11.         switch (result)
12.         {
13.             case SignInStatus.Success:
14.                 return RedirectToLocal(returnUrl);
15.             case SignInStatus.LockedOut:
16.                 return View("Lockout");
17.             case SignInStatus.RequiresVerification:
18.                 return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });
19.             case SignInStatus.Failure:
20.             default:
21.                 ModelState.AddModelError("", "Invalid login attempt.");
22.                 return View(model);
23.         }
24.     }
```

Register – “/Account/Register”

Register page is a little bit different from the default one in the ASP.NET. This is because there are two options for registration – a normal user or a financial institution. Therefore, the UI contains additional checkbox that specifies whether the registering user is a financial institution and if checked, a new input field is shown below with a place to enter a business name.

Just like the Login page, there are many validations that need to be made before a new user enters into the database. There are validations for the username, email address and password – their length (min, max) and so on. There also is a validation if the password and the confirmed password are one and the same.

It also important to mention that when a financial institution is registered there are six default purchase profiles that are assigned to it. These profiles don't have an interest rate (by default it is 0) but have term and price ranges that are most often used. Of course, after being registered the financial institution can go to their profile and change these profiles and their corresponding interest rates.

They also have several default insurances assigned to them. The possible types were already mentioned above – InsuranceType nomenclature. Here is how it is done in the code:

```
1. [HttpPost]
2.     [AllowAnonymous]
3.     [ValidateAntiForgeryToken]
4.     public async Task<ActionResult> Register(RegisterViewModel model)
5.     {
6.         if (ModelState.IsValid)
7.         {
8.             User user;
9.             string role = string.Empty;
10.
11.            if (model.IsFinancialInstitution)
12.            {
13.                PurchaseProfile upToOneYearUpTo2000 = new PurchaseProfile
14.                {
15.                    MonthsMin = 3,
16.                    MonthsMax = 12,
17.                    PriceMin = 100,
18.                    PriceMax = 2000
19.                };
20.
21.                PurchaseProfile upToOneYearAbove2000 = new PurchaseProfile
22.                {
23.                    MonthsMin = 3,
24.                    MonthsMax = 12,
```

```
25.         PriceMin = 2000.001m,
26.         PriceMax = 100000
27.     };
28.     PurchaseProfile upToTwoYearsUpTo2000 = new PurchaseProfile
29.     {
30.         MonthsMin = 13,
31.         MonthsMax = 24,
32.         PriceMin = 100,
33.         PriceMax = 2000
34.     };
35.     PurchaseProfile upToTwoYearsAbove2000 = new PurchaseProfile
36.     {
37.         MonthsMin = 13,
38.         MonthsMax = 24,
39.         PriceMin = 2000.001m,
40.         PriceMax = 100000
41.     };
42.     PurchaseProfile upToThreeYearsUpTo2000 = new PurchaseProfile
43.     {
44.         MonthsMin = 25,
45.         MonthsMax = 36,
46.         PriceMin = 100,
47.         PriceMax = 2000
48.     };
49.     PurchaseProfile upToThreeYearsAbove2000 = new PurchaseProfile
50.     {
51.         MonthsMin = 25,
52.         MonthsMax = 36,
53.         PriceMin = 2000.001m,
54.         PriceMax = 100000
55.     };
56.
57.     FinancialInstitution financialInstitution = new FinancialInstitution
58.     {
59.         UserName = model.UserName,
60.         Email = model.Email,
61.         Name = model.BusinessName
62.     };
63.
64.     financialInstitution.FinancialInstitutionPurchaseProfiles =
65.         new List<FinancialInstitutionPurchaseProfile>()
66.     {
67.         new FinancialInstitutionPurchaseProfile
68.         {
69.             FinancialInstitution = financialInstitution,
70.             PurchaseProfile = upToOneYearUpTo2000
71.         },
72.         new FinancialInstitutionPurchaseProfile
73.         {
74.             FinancialInstitution = financialInstitution,
75.             PurchaseProfile = upToOneYearAbove2000
76.         },
77.         new FinancialInstitutionPurchaseProfile
78.         {
```

```

79.             FinancialInstitution = financialInstitution,
80.             PurchaseProfile = upToTwoYearsUpTo2000
81.         },
82.         new FinancialInstitutionPurchaseProfile
83.     {
84.         FinancialInstitution = financialInstitution,
85.         PurchaseProfile = upToTwoYearsAbove2000
86.     },
87.         new FinancialInstitutionPurchaseProfile
88.     {
89.         FinancialInstitution = financialInstitution,
90.         PurchaseProfile = upToThreeYearsUpTo2000
91.     },
92.         new FinancialInstitutionPurchaseProfile
93.     {
94.         FinancialInstitution = financialInstitution,
95.         PurchaseProfile = upToThreeYearsAbove2000
96.     }
97. };
98.
99.     Insurance insurance1 = new Insurance
100.    {
101.        Type = InsuranceType.All,
102.        PercentageRate = 0.1,
103.        FinancialInstitution = financialInstitution
104.    };
105.
106.     Insurance insurance2 = new Insurance
107.    {
108.        Type = InsuranceType.Life,
109.        PercentageRate = 0.03,
110.        FinancialInstitution = financialInstitution
111.    };
112.     Insurance insurance3 = new Insurance
113.    {
114.        Type = InsuranceType.LifeAndUnemployment,
115.        PercentageRate = 0.05,
116.        FinancialInstitution = financialInstitution
117.    };
118.     Insurance insurance4 = new Insurance
119.    {
120.        Type = InsuranceType.None,
121.        PercentageRate = 0,
122.        FinancialInstitution = financialInstitution
123.    };
124.     Insurance insurance5 = new Insurance
125.    {
126.        Type = InsuranceType.Purchase,
127.        PercentageRate = 0.03,
128.        FinancialInstitution = financialInstitution
129.    };
130.     Insurance insurance6 = new Insurance
131.    {
132.        Type = InsuranceType.Unemployment,

```

```

133.                PercentageRate = 0.03,
134.                FinancialInstitution = financialInstitution
135.            };
136.
137.            user = financialInstitution;
138.            role = GlobalConstants.FinancialInstitutionRole;
139.        }
140.        else
141.        {
142.            user = new User { UserName = model.UserName, Email = model.Email, }
143.            ;
144.            role = GlobalConstants.UserRole;
145.        }
146.        var result = await UserManager.CreateAsync(user, model.Password);
147.        if (result.Succeeded)
148.        {
149.            this.UserManager.AddToRole(user.Id, role);
150.            await SignInManager.SignInAsync(user, isPersistent: false, remember
151.                Browser: false);
152.            return RedirectToAction("Index", "Home");
153.        }
154.
155.        AddErrors(result);
156.    }
157.
158.    return View(model);
159.    }

```

3.4. Creation of the pages for users who are buyers

When guests register or login, they become users with the right to buy products and calculate their POS credit repayment. Users can access the direct page AllProducts and ProductDetails and from there – GetOnCredit and Calculate.

AllProducts – “~/Products/AllProducts”

This page shows a list of all products that are offered by the retailer. They are arranged in a table with pages (the default number of products per page is 4) and can be searched for. The paging is implemented with a library called PagedList, very commonly used for such default paging.

PagedList

The PagedList.Mvc package can be easily installed from the NuGet gallery. It is a library which creates a default view for the paging of a list of elements. It installs a PagedList collection type and extension methods for C# collections like IQueryable and IEnumerable. These extension methods return not all data as IQueryable or IEnumerable, but a single page in the form of a PagedList collection. The package also installs a helper that displays the paging buttons.

```
1. public ActionResult AllProducts(int? page)
2.     {
3.         var allProducts = this.Data.Products
4.             .All()
5.             .Project().To<AllProductsViewModel>()
6.             .OrderBy(u => u.Name);
7.
8.         var pageNumber = page ?? 1;
9.         var onePageOfProducts = allProducts.ToPagedList(pageNumber,
10.             4);
11.         return this.View(onePageOfProducts);
12.     }
```

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.

First, the data is extracted by using the repository Products in the Unit of Work pattern. The return list should be projected to the expected VlewModel from the View –

AllProductsViewModel and then the collection should be ordered. This is absolutely necessary in order for the paging to work properly. 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.

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.

There is one more thing to do – add a partial view for the pager:

```

1. @model IPagedList<POSCreditRepayments.Web.ViewModels.Products.AllProductsViewModel>
2.
3. <div class="row">
4.     <div class="col-md-3 col-md-offset-5">
5.         @Html.PagedListPager(Model, page => Url.Action("AllProducts", new { pagepage
           = page })), new PagedListRenderOptions
6.             {
7.                 DisplayLinkToFirstPage = PagedListDisplayMode.Always,
8.                 DisplayLinkToLastPage = PagedListDisplayMode.Always,
9.                 DisplayLinkToPreviousPage = PagedListDisplayMode.Always,
10.                DisplayLinkToNextPage = PagedListDisplayMode.Always,
11.                MaximumPageNumbersToDisplay = 5
12.            })
13.     </div>
14. </div>

```

These are some settings concerning the interface of the pager that can be edited like whether to show links to the first and last page or just to the previous and next. This is a very easy way to customize the view.

The search is another partial that is implemented with JavaScript. There is a single input field where the customer can write the name or part of the name of the product he is looking for. Here is how it is implemented in the view:

```

1. <div class="row">
2.     <div class="col-md-3 col-md-offset-6">
3.         <h4 class="control-label white"><strong>Search for Product</strong></h4>

```

```
4.     </div>
5.     <div class="col-md-3">
6.         <input type="text" id="product-search" class="form-
        control" />
7.     </div>
8. </div>
```

The method in the code that retrieves the products is very simple and it resembles the get all method but works with query string:

```
1. [HttpGet]
2.     public ActionResult SearchForProduct(string query)
3.     {
4.         var suggestedUsers = this.Data.Products
5.                                 .All()
6.                                 .Select(u => new { Id = u.ProductId
7.                                     , Name = u.Name })
8.                                 .ToList();
9.         return this.Json(suggestedUsers, JsonRequestBehavior.AllowGet);
10.    }
```

The filtering is conducted with Ajax for better user experience and better speed of the browsing. When the user has found what he is looking for, each name is a link to the details of the product. This is implemented with the function `getEditUserPath` which works in the same way as the `GetById` method as a server alternative. It also has a function that takes care of the autocomplete of the word if some match is found.

```
1. $(document).ready(function () {
2.     var APPLICATION_NAME = 'POSCreditsRepaymentCalculator';
3.
4.     $('#product-search').devbridgeAutocomplete({
5.         serviceUrl: getServiceUrl('/Products/SearchForProduct'),
6.         transformResult: function (response) {
7.             response = JSON.parse(response);
8.             return {
9.                 suggestions: $.map(response, function (dataItem) {
```



```
10.         return { value: dataItem.Name, data: dataItem
11.             .Id };
12.     });
13. },
14.     onSelect: function (suggestion) {
15.         window.location.href = getEditUserPath(suggestion.data);
16.     },
17. });
18.
19.     function getBaseUrl() {
20.         var hostName = $(location).attr('host');
21.         hostName = 'http://' + hostName;
22.
23.         var appName = $(location).attr('pathname').split('/')[1];
24.
25.         if (appName !== APPLICATION_NAME) {
26.             appName = '';
27.         } else {
28.             appName = '/' + appName;
29.         }
30.
31.         var baseUrl = hostName + appName;
32.
33.         return baseUrl;
34.     }
35.
36.     function getServiceUrl(serviceName) {
37.         var appName = getBaseUrl();
38.         var finalUrl = appName + serviceName;
39.
40.         return finalUrl;
41.     }
42.
43.     function getEditUserPath(id) {
44.         var baseUrl = getBaseUrl();
45.         var whatsLeft = '/Products/ProductDetails/' + id;
46.         var final = baseUrl + whatsLeft;
```

```

47.         return final;
48.     }
49. });

```

ProductDetails – “~/Products/ProductDetails/{id}”

If a customer wants to see additional characteristics of a product in the AllProducts list, he needs to click on either its name or the image. This redirects to a page with the full information about the product.

```

1. <div class="row">
2.     <div class="col-lg-11 col-lg-offset-1">
3.         <hr>
4.         <h2 class="intro-text text-center">
5.             @Model.Name
6.         </h2>
7.         <hr>
8.         
10.        <hr class="visible-xs">
11.            <p>@Html.Raw(HttpUtility.HtmlDecode(Model.Descrip
12.                tion))</p>
13.            <hr class="visible-xs">
14.            <p><h3>Price: @Model.Price</h3></p>
15.        </div>
16.    </div>
17.    <br />
18.    <div class="row">
19.        <div class="col-lg-11 col-lg-offset-1">
20.            @Html.ActionLink("Buy on credit", "GetOnCredit",
                "Credit", new { id = Model.ProductId }, new { @class = "btn btn-
                lg btn-info" })
21.        </div>
22.    </div>

```

This represents the whole structure of the details of a product. It should be highlighted that the model’s description is HTML decoded. In order to look more structured and organized, the

description could contain HTML tags like `<p>` or ``, etc. If such text is not decoded, it is going to be displayed with the tags as plain words.

The method that retrieves the specific product is the following:

```
1. [HttpGet]
2.     public ActionResult ProductDetails(int id)
3.     {
4.         var article = this.Data.Products
5.             .All()
6.             .Where(x => x.ProductId == id)
7.             .Project()
8.             .To<ProductDetailsViewModel>()
9.             .FirstOrDefault();
10.        if (article == null)
11.        {
12.            return this.HttpNotFound();
13.        }
14.
15.        return this.View(article);
16.    }
```

After the repository is called with the defined method `All()`, the wanted ID is extracted and then the domain model is projected/mapped to the view model that is going to be displayed in the view (using Automapper). If such product is not found, an error is return as this is not a likely or wanted situation.

GetOnCredit – “/Credit/GetOnCredit/{id}”

If the customer decides to buy a product and clicks the “Buy on credit” button, he is redirected to a page where he can define the specifics of the credit. In order to send the data that the customer is going to fill, the fields should be in a `BeginForm`. This is a good practice because the fields can also be validated in case of any forgery attempts for example to send script through an input field which is then going to be executed in the database.

```
1. <h2 class="text-center">How much will my monthly payments be ?</h2>
2.     <section id="userInformation" class="top-bottom-space">
```

```

3.         @using (Html.BeginForm("Calculate",
4.             "Credit",
5.             FormMethod.Post,
6.             new { @class = "form-horizontal", role = "form" }))
7.         {
8.             @Html.ValidationSummary(true, string.Empty, new {
9.                 @class = "text-danger" })
10.            @Html.AntiForgeryToken()
11.            @Html.EditorForModel()
12.
13.            <div class="form-group">
14.                <div class="col-md-offset-4 col-md-8">
15.                    <input type="submit" class="btn send_
16.                        btn" value="Calculate" />
17.                </div>
18.            </div>
19.        }
20.    </section>

```

The view also includes the `EditorForModel()` template (already discussed above) for the view model that is passed. As shown below, this template contains all fields of the model and the way that they should be displayed to the users.

```

1. @model POSCreditRepayments.Web.ViewModels.Credits.CreditEditorTemplateViewModel
2.
3. <div class="form-group">
4.     @Html.LabelFor(m => m.Product.Name, new { @class = "col-md-4 control-label" })
5.     <div class="col-md-8">
6.         @Html.TextBoxFor(m => m.Product.Name, new { @readonly = "readonly", @class = "form-
7.             control" })
8.     </div>
9. </div>
10. <div class="form-group">
11.     @Html.LabelFor(m => m.Product.Price, new { @class = "col-md-4 control-label" })
12.     <div class="col-md-8">
13.         @Html.TextBoxFor(m => m.Product.Price, new { @readonly = "readonly", @class = "form-
14.             control" })
15.     </div>
16. </div>
17. <div class="form-group">
18.     @Html.LabelFor(m => m.FinancialInstitutions, new { @class = "col-md-4 control-label" })
19.     <div class="col-md-8">
20.         @Html.DropDownListFor(m => m.SelectedFinancialInstitutions, Model.FinancialInstitution
21.             s, new { @class = "form-control" })

```

```

19.     </div>
20. </div>
21. <div class="form-group">
22.     @Html.LabelFor(m => m.Downpayment, new { @class = "col-md-4 control-label" })
23.     <div class="col-md-8">
24.         @Html.TextBoxFor(m => m.Downpayment, new { @class = "form-
control", type = "number", id = "downpayment" })
25.     </div>
26.     @Html.ValidationMessageFor(m => m.Downpayment, string.Empty, new { @class = "text-
danger col-md-offset-1" })
27. </div>
28. <div class="form-group">
29.     @Html.LabelFor(m => m.Term, new { @class = "col-md-4 control-label" })
30.     <div class="col-md-8">
31.         @Html.TextBoxFor(m => m.Term, new { @class = "form-control", type = "number" })
32.     </div>
33.     @Html.ValidationMessageFor(m => m.Term, string.Empty, new { @class = "text-danger col-md-
offset-1" })
34. </div>
35. <div class="form-group">
36.     @Html.LabelFor(m => m.InsuranceType, new { @class = "col-md-4 control-label" })
37.     <div class="col-md-4">
38.         @Html.EnumDropDownListFor(m => m.InsuranceType, new { @class = "form-
control", @onchange = "calculateInsurance(this.value, " + Model.Product.Price + ")" })
39.     </div>
40. </div>

```

And this is the ViewModel itself:

```

1. public class CreditEditorTemplateViewModel
2. {
3.     [DisplayName("Down payment")]
4.     [Required]
5.     public decimal Downpayment { get; set; }
6.
7.     [DisplayName("Finanial institution")]
8.     [Required]
9.     public IEnumerable<SelectListItem> FinancialInstitutions { get; set; }
10.
11.     [DisplayName("Product insurance")]
12.     public InsuranceType InsuranceType { get; set; }
13.
14.     public Product Product { get; set; }
15.
16.     [Required]
17.     public IEnumerable<string> SelectedFinancialInstitutions { get; set; }
18.
19.     [DisplayName("Credit term (months)")]
20.     [Range(3, 36)]
21.     [Required]
22.     public int Term { get; set; }
23. }

```

It is very clear with this example why using such view models is very beneficial. Here the view model has many custom defined restrictions and annotations which can be different for the different pages where the domain model is displayed. Moreover, in the editor template each property is explicitly set how it is going to be presented and if in the future this needs to be changed, this is only one place that is going to be altered. It also has a `ValidationMessageFor` under each property so that the user can immediately understand if he is making a mistake with a certain field.

The first two text fields – product name and price, have `readonly` class because the user shouldn't be able to edit them. Apart from that, all other fields are open for modification. The down payment and the credit term both have type number to their inputs so that the user cannot enter anything else but numbers there.

In the code, the action that handles rendering this page is the `GetOnCredit` method:

```
1. public ActionResult GetOnCredit(int id)
2. {
3.     CreditEditorTemplateViewModel model = new CreditEditorTemplateViewModel()
4.     ;
5.     model.FinancialInstitutions = this.populator.GetFinancialInstitutions();
6.
7.     Product product = this.Data.Products.GetById(id);
8.     model.Product = product;
9.
10.    return this.View(model);
11. }
```

It uses a helper interface `IDropDownListPopulator` which as its name suggests populates the dropdown with the financial institutions that give credits. This is an interface which can provide different methods for populating financial institutions as it is the case here or it can be easily extended to other objects. In the project, the implementation of the interface is the class `DropDownListPopulator` which handles the extraction of the institutions with the following method:

```
1. public IEnumerable<SelectListItem> GetFinancialInstitutions()
2. {
3.     var institutions = this.cache.Get<IEnumerable<SelectListItem>>("Fi
4.     nancialInstitutions",
5.     () =>
```

```
5.      {
6.          return this.data.FinancialInstitutions
7.              .All()
8.              .Where(c => c.IsApproved)
9.              .Select(c => new SelectListItem
10.                  {
11.                      Value = c.Id.ToString(),
12.                      Text = c.Name
13.                  })
14.              .ToList();
15.      });
16.
17.      return institutions;
18.  }
```

In a real situation, this dropdown will rarely change its values because the administrators have to approve the registered institutions first. Therefore, to optimize the speed of the application, these values should be cached.

The implementation of the `ICacheService` interface in the current project is `InMemoryCache` which caches the data in the `HttpContext.Current.Cache`. This is a class which basically provides caching of any kind of serializable objects.

`HttpContext.Current.Cache` is usually used to cache data that comes from the database servers. This saves having to continually ask the database for data that changes little (as in the case with the financial institutions). This is entirely server-side and does not affect the client.

```
1. public T Get<T>(string cacheID, Func<T> getItemCallback) where T : class
2. {
3.     T item = HttpContext.Current.Cache.Get(cacheID) as T;
4.     if (item == null)
5.     {
6.         item = getItemCallback();
7.         HttpContext.Current.Cache.Insert(cacheID, item, null, DateTime
            .Now.AddSeconds(10), Cache.NoSlidingExpiration);
8.     }
9.     return item;
10. }
```

For the purpose of presentation, the duration of the cache has been made 10 seconds. However, in a real situation it can easily be made 15 minutes or even more. The method is even generic so that it can work with different types – string, int, or objects.

Calculate – “/Credit/Calculate/{model}”

After the user has entered the specifications of the credit that he prefers, a POST request is made to the CreditController to calculate the credit amounts having in mind the user's input.

```

1. public ActionResult Calculate(CreditEditorTemplateViewModel viewModel)
2. {
3.     string institutionId = viewModel.SelectedFinancialInstitutions.FirstOrDefault();
4.     FinancialInstitution institution = this.Data.FinancialInstitutions.GetById(institutionId);
5.
6.     decimal insuranceAmountPerMonth = (decimal)institution.Insurances
7.                                     .FirstOrDefault(x => x.Type == viewModel.InsuranceType)
8.                                     .PercentageRate * viewModel.Product.Price;
9.     decimal creditAmount = viewModel.Product.Price +
10.                            institution.ApplicationFee +
11.                            insuranceAmountPerMonth * viewModel.Term -
12.                            viewModel.Downpayment;
13.     double interestRate = institution.FinancialInstitutionPurchaseProfiles
14.                            .Where(x => x.PurchaseProfile.MonthsMin <= viewModel.Term &&
15.                                    x.PurchaseProfile.MonthsMax >= viewModel.Term &&
16.                                    x.PurchaseProfile.PriceMin <= creditAmount &&
17.                                    x.PurchaseProfile.PriceMax >= creditAmount)
18.                            .FirstOrDefault()
19.                            .InterestRate;
20.     double interestRatePerMonth = interestRate / 1200;
21.     double interestRateForTerm = Math.Pow(1 + interestRatePerMonth, viewModel.Term);
22.     decimal monthlyPayment = ((decimal)interestRatePerMonth * creditAmount * (decimal)interestRateForTerm) / (decimal)(interestRateForTerm - 1);
23.     decimal totalAmount = monthlyPayment * viewModel.Term;
24.     double apr = (Math.Pow(1 + interestRatePerMonth, 12) - 1) * 100;
25.
26.     CreditDisplayTemplateViewModel model = new CreditDisplayTemplateViewModel
27.     {
28.         CreditAmount = Math.Round(creditAmount, 2),
29.         Downpayment = viewModel.Downpayment,
30.         FinancialInstitutionName = institution.Name,

```



```
31.         Insurance = viewModel.InsuranceType.ToString(),
32.         InterestRatePerMonth = Math.Round(interestRate / 12, 2),
33.         InterestRatePerYear = Math.Round(interestRate, 2),
34.         TotalAmount = Math.Round(totalAmount, 2),
35.         InterestAmount = Math.Round(totalAmount - creditAmount, 2),
36.         Term = viewModel.Term,
37.         MonthlyPayment = Math.Round(monthlyPayment, 2),
38.         Apr = Math.Round(apr, 2),
39.         ProductName = viewModel.Product.Name,
40.         ProductPrice = viewModel.Product.Price,
41.         InsuranceAmount = insuranceAmountPerMonth
42.     };
43.
44.     return this.View(model);
45. }
```

The method first finds the chosen financial institution as pointed in the dropdown. It is the pivot point for getting the rates for the interest and insurance. Then the insurance amount is calculated as it is added to the price of the product in order to form the whole credit amount that is going to be lent to the customer. On the basis of that amount and the term of the credit, the interest rate is chosen from the purchase profiles that the financial institution has. And after that the monthly payments and the Annual Percentage Rate (APR) are calculated using the formulas mentioned in the first section of the document.

After all amounts and indexes are found, the credit view model can be created and the credit can be saved in the database for future reference.

Generating PDF with the standardized Euro form

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.

For the generation of the PDF document the following library is used:

Rotativa

In ASP.NET, usually, generating PDF for any kind of document or business report that can also be printable is a bit time-consuming and cumbersome. But with Rotativa this is accomplished very easily and quickly. All that should be done is to download the library which is available in Nuget package. It provides the flexibility to create PDFs directly from Views which is exactly what is done in the current project. The following is the first part of the view that creates the Euro form as the law states it should look like:

```
1. @model POSCreditRepayments.Web.ViewModels.EuroFormTemplate.EuroFormTemplateVi
   ewModel
2.
3. <div class="container">
4.     <div class="box">
5.         <h2 class="text-
           center">European standard form to provide information on consumer loans</h2>
6.         <h3>
7.             Part I. Identification and contact information of the creditor /
           creditor intermediary
```

```

8.      </h3>
9.      <table class="table table-bordered text-center">
10.         <tr>
11.             <td class="col-xs-6">
12.                 1. Creditor
13.             </td>
14.             <td class="col-xs-6">
15.                 @Model.FinancialInstitution.Name
16.             </td>
17.         </tr>
18.         <tr>
19.             <td class="col-xs-6">
20.                 2. Address
21.             </td>
22.             <td class="col-xs-6">
23.                 @Model.FinancialInstitution.Address
24.             </td>
25.         </tr>
26.         <tr>
27.             <td class="col-xs-6">
28.                 3. Phone Number
29.             </td>
30.             <td class="col-xs-6">
31.                 @Model.FinancialInstitution.PhoneNumber
32.             </td>
33.         </tr>
34.         <tr>
35.             <td class="col-xs-6">
36.                 5. Fax
37.             </td>
38.             <td class="col-xs-6">
39.                 @Model.FinancialInstitution.Fax
40.             </td>
41.         </tr>
42.         <tr>
43.             <td class="col-xs-6">
44.                 6. Web Site
45.             </td>
46.             <td class="col-xs-6">
47.                 @Model.FinancialInstitution.WebSite
48.             </td>
49.         </tr>
50.         <tr>
51.             <td class="col-xs-6">

```

```

52.                7. Credit Intermediary
53.                </td>
54.                <td class="col-xs-6">
55.                    @Model.FinancialInstitution.CreditIntermerdiary
56.                </td>
57.            </tr>
58.        </table>

```

The whole PDF file can be found in the attached files at the end of the document.

The method that takes care of the generation of the PDF is very simple thanks to the library.

```

1. public ActionResult GeneratePdfEuroFormTemplate(CreditDisplayTemplateViewMode
   1 model)
2.     {
3.         EuroFormTemplateViewModel template = new EuroFormTemplateViewModel
4.         ();
5.         template.Credit = model;
6.         template.FinancialInstitution = this.Data.FinancialInstitutions
7.         .All()
8.         .Where(fi => fi.Name == model.
   FinancialInstitutionName)
9.         .Project()
10.        .To<EditFinancialInstitutionPr
   ofileViewModel>()
11.        .FirstOrDefault();
12.        return new Rotativa.ViewAsPdf("EuroFormTemplate", template)
13.        { FileName = "EuroFormTemplate.pdf" };

```

First, the financial institution is extracted with a query, the credit is passed as a variable from the previous view and then they are mapped to the view model that represents the Euro form. After that the Rotativa library takes care of the rest by transforming the view into a PDF file.

3.6. Creation of the pages for users who are financial institutions

When financial institutions login with their username and password, they can go to their profiles and update the information there according to their policies.

FinancialInstitutionProfile – “/FinancialInstitution/FinancialInstitutionProfile”

The method that is used here is HTTP GET request by the ID of the financial institution. Again it is a query which first gets the all financial institutions and projects them to the needed view model and then selects the one that matches the ID.

```

1. [HttpGet]
2.     public ActionResult FinancialInstitutionProfile(string id)
3.     {
4.         if (id == null)
5.         {
6.             id = this.CurrentUser.Id;
7.         }
8.
9.         FinancialInstitutionProfileViewModel institution = this.Data.FinancialInstitutions
10.
11.                                     .All()
12.                                     .Project()
13.                                     .To<FinancialInstitutionPro
fileViewModel>()
14.
15.                                     .FirstOrDefault(x => x.Id =
= id);
16.
17.         if (institution != null)
18.         {
19.             return this.View(institution);
20.         }
21.         return this.RedirectToAction("Error", "Home");
22.     }

```

The view that is rendered is just like the ones above where the properties of the model are displayed with their name as a label and a text box for the value which is disabled so that it cannot be edited. The only difference is the cycle that goes through all interest rates with their purchase profiles and the insurance rates:

```

1. <div class="row">
2.     <h3 class="text-center">Interest rates</h3>
3. </div>
4. @foreach (var purchaseProfile in Model.FinancialInstitutionPurchaseProfiles)
5. {
6.     <div class="form-group padding">
7.         <div class="col-md-4 control-label">
8.             <label>
9.                 Credit term: @purchaseProfile.PurchaseProfile.MonthsMin - @purchaseProfile
.PurchaseProfile.MonthsMax months
10.            <br />
11.            Price range: @purchaseProfile.PurchaseProfile.PriceMin - @purchaseProfile.
PurchaseProfile.PriceMax
12.        </label>
13.    </div>
14.    <div class="col-md-8">
15.        @Html.TextBoxFor(m => purchaseProfile.InterestRate, new { @readonly = "readonl
y", @class = "form-control" })
16.    </div>
17. </div>

```

```

18.     }
19.     <hr />
20.     <div class="row">
21.         <h3 class="text-center">Insurance rates</h3>
22.     </div>
23.     @foreach (var insurance in Model.Insurances)
24.     {
25.         <div class="form-group padding">
26.             <div class="col-md-4 control-label">
27.                 <label>
28.                     @insurance.Type
29.                 </label>
30.             </div>
31.             <div class="col-md-8">
32.                 @Html.TextBoxFor(m => insurance.PercentageRate, new { @readonly = "readonly",
33.                     @class = "form-control", type = "number" })
34.             </div>
35.         }

```

There is no validation for this view as there is no input that comes from the users.

EditFinancialInstitutionProfile –

“/FinancialInstitution/EditFinancialInstitutionProfile/{id}”

This part has both HTTP GET and POST methods. The first is to get the view that enables editing of the fields and the second one is the action that handles the view model passed with the altered values.

The GET method is the same as everywhere else where the data is filtered by the given ID and mapped to the necessary view model. If no institution is found with the given ID then the user is redirected to the Home page.

The POST method, however, is more complex as there are several steps that need to be done in order to update the financial institution’s profile information. First, the financial institution should be found by its ID. If it exists, the direct properties are set. Then, the interest rates are updated by finding the purchase profile with the given term and price, and if no such profile is found, a new one is created. If a new purchase profile is to be added, a simple validation that the price and term minimum is less than the maximum. Because this data is very sensitive to mistakes, the only users that can edit it are the administrators. In the end, the data changes are saved and the details are updated.

1. [HttpPost]
2. [ValidateAntiForgeryToken]

```

3. public ActionResult EditFinancialInstitutionProfile(EditFinancialInstitutionProfileViewModel m
odel)
4. {
5.     if (this.ModelState.IsValid)
6.     {
7.         FinancialInstitution institution = this.Data.FinancialInstitutions.GetById(model.Id);
8.
9.         if (institution != null)
10.        {
11.            institution.Address = model.Address;
12.            institution.CreditIntermediary = model.CreditIntermediary;
13.            institution.Email = model.Email;
14.            institution.Fax = model.Fax;
15.            institution.Phone = model.Phone;
16.            institution.WebSite = model.WebSite;
17.            institution.ApplicationFee = model.ApplicationFee;
18.
19.            foreach (var viewModel in model.FinancialInstitutionPurchaseProfileViewModels)
20.            {
21.                FinancialInstitutionPurchaseProfile financialInstitutionPurchaseProfile =
22.                    this.Data.FinancialInstitutionPurchaseProfiles
23.                        .All()
24.                        .FirstOrDefault(p => p.PurchaseProfile.MonthsMin == viewModel.MonthsMin &&
25.
26.                            p.PurchaseProfile.MonthsMax == viewModel.MonthsMax &&
27.                            p.PurchaseProfile.PriceMin == viewModel.PriceMin &&
28.                            p.PurchaseProfile.PriceMax == viewModel.PriceMax &&
29.                            p.FinancialInstitutionId == institution.Id);
30.
31.                if (financialInstitutionPurchaseProfile == null)
32.                {
33.                    PurchaseProfile purchaseProfile = this.Data.PurchaseProfiles
34.                        .All()
35.                        .FirstOrDefault(p => p.MonthsMin ==
36.                            viewModel.MonthsMin &&
37.                            p.MonthsMax ==
38.                            viewModel.MonthsMax &&
39.                            p.PriceMin == v
40.                            iewModel.PriceMin &&
41.                            p.PriceMax == v
42.                            iewModel.PriceMax);
43.
44.                    if (purchaseProfile == null)
45.                    {
46.                        if (viewModel.PriceMin >= viewModel.PriceMax || viewModel.MonthsMin >=
47.                            viewModel.MonthsMax)
48.                        {
49.                            return this.RedirectToAction("Error", "Home");
50.                        }
51.                        else
52.                        {
53.                            purchaseProfile = new PurchaseProfile
54.                            {
55.                                MonthsMin = viewModel.MonthsMin,
56.                                MonthsMax = viewModel.MonthsMax,
57.                                PriceMin = viewModel.PriceMin,
58.                                PriceMax = viewModel.PriceMax
59.                            };
60.                            this.Data.PurchaseProfiles.Add(purchaseProfile);

```

```

56.         }
57.     }
58.
59.     financialInstitutionPurchaseProfile = new FinancialInstitutionPurchaseProf
    ile
60.     {
61.         PurchaseProfile = purchaseProfile,
62.         FinancialInstitution = institution
63.     };
64.
65.     this.Data.FinancialInstitutionPurchaseProfiles.Add(financialInstitutionPur
    chaseProfile);
66.     }
67.
68.     financialInstitutionPurchaseProfile.InterestRate = viewModel.InterestRate;
69. }
70.
71.     this.Data.SaveChanges();
72.
73.     return this.RedirectToAction("FinancialInstitutionProfile", "FinancialInstitution"
    , model.Id);
74.     }
75. }
76.
77.     return this.RedirectToAction("Error", "Home");
78. }
79.

```

3.7. Creation of the pages for users who are administrators

When a user is an administrator, he has all possible rights and can access all features of the application. Apart from buying products on credit as regular users, admins can approve and disapprove financial institutions and edit their profile information.

AllFinancialInsitutions – “/Admin/AllFinancialInstitutions”

This page shows a grid with all registered financial institutions where the admin has to decide whether they should be approved and participate in the calculation process or not.

After the database is queried and the list that is returned is ordered by name, the Ajax that toggles between approved and disapproved institutions is the following:

```

1. @foreach (var cp in Model)
2.     {
3.         <tr class="row">
4.             <td class="col-md-6">
5.                 @Html.ActionLink(cp.Name, "FinancialInstitutionPro
    file", "FinancialInstitution", new { id = cp.Id }, new { })

```



```

6.         </td>
7.         <td class="col-md-6">
8.             @using (Ajax.BeginForm("ToggleFinancialInstitution
9.                 sStatus", "Admin", new { id = cp.Id },
10.                    new AjaxOptions
11.                    {
12.                        HttpMethod = "Post",
13.                        UpdateTargetId = "institutions",
14.                        InsertionModeInsertionMode = Inse
15.                        rtionMode.Replace
16.                    },
17.                    new { }))
18.            {
19.                @Html.AntiForgeryToken()
20.                if (cp.IsApproved)
21.                {
22.                    <input type="submit" value="Disapprov
23.                    e" class="btn btn-danger" />
24.                }
25.                else
26.                {
27.                    <input type="submit" value="Approve"
28.                    class="btn btn-success" />
29.                }
30.            }
31.        </td>
32.    </tr>
33.    }

```

The first case is when the financial institution is already approved, then the request that can be sent is to disapprove it and the opposite is to approve it. In both cases the request is sent to the `ToggleFinancialInstitutionsStatus` method which on its hand toggles the status in the following way:

```

1. public ActionResult ToggleFinancialInstitutionsStatus(string id)
2. {
3.     FinancialInstitution institution = this.Data.FinancialInstitutions.GetById(id);
4.
5.     if (institution != null)
6.     {
7.         institution.IsApproved = institution.IsApproved == true ? false : true;
8.         this.Data.SaveChanges();
9.     }
10.
11.     IList<AllFinancialInstitutionsViewModel> institutions = this.GetFinancialInstituti
12.     ons();
13.     return this.PartialView("_AllFinancialInstitutionsPartial", institutions);

```

```
14.     }  
15.
```

First, the status is changed to the opposite of its current value and then another method is called to retrieve the list of financial institution again. This method is just like any other GetAll method from the abovementioned but maps the objects to AllFinancialInstitutionsViewModel which is a much smaller version of the AspNetUser and has only Id, name and status.

Apart from that, as already mentioned above the edit the profile of the financial institutions in which case the FinancialInsitutionController is referenced and the actions there are used.

4. TESTING THE APPLICATION

To verify that the developed web application meets the requirements set out in the design solution and satisfies the needs of participants in the process of calculation of POS credit repayment in the most adequate way, it is necessary to test the functionalities developed and matched expectations.

For this purpose it is necessary to draw up a plan of action lines that outline the steps to verify the correct operation of the information system:

- To create users with different roles in the table User
- To add sample data (preferably real) in the main tables as Products, Financial institutions, Purchase profiles, Insurances
- To test the system login and access control forms to the created user accounts and roles and try using inaccessible by definition features
- Use the administrator account login and move along the process of approving/disapproving financial institutions and modifying their interest rates and profile information
- Use the financial institution's account login and move along the process of modifying its interest rates and profile information, also trying to buy a product on credit and being referred to the login page
- Check whether the administrator can alter the credit terms and price ranges of the financial institution and at the same time the financial institution to be forbidden to change them (only the interest rates to be editable)
- Use the user account to browse through the products and see their extended details and information (test paging and searching for products)
- Use the user account or the administrator's account to buy a product on credit and calculate its payments
- To refuse to calculate the credit repayments for users with financial institution's account
- Checking whether the financial institution is shown in the dropdown with financial institutions when calculating a product if it has already been approved (and vice versa)

- A PDF file to be downloaded when the button Download PDF is clicked
- Check whether the information from the last page of the calculation is the same as the information in the PDF file

This testing plan does not affect the purely technical checks on the system, which must also be made, such as response incorrect input; multiple login attempts with incorrect password; modifying the transmitted data and of the forms etc.

5. INTEGRATION OF THE INFORMATION SYSTEM

In order for the application to work in another environment like server or computer, the following things should be done:

- Windows OS
- IIS
- Microsoft SQL Server
- Copy of the web application

As the database is created with Code First and Entity Framework, there is no need for any kind of script or backup of databases. The only thing that needs to be done is to copy the directory of the web application and to create a virtual directory in the IIS which leads to the former path.

Conclusion

1. FUTURE DEVELOPMENT OF THE APPLICATION

The concrete realization of the design solution is fully functional, but it is always necessary to seek to optimize, improve and develop. Possible directions for changes and additions to the implementation are:

- Optimization of application performance through the use of funds of ASP.NET and Microsoft SQL Server for caching query results. It is already used in some places (eg. Dropdown with financial institutions), but could seek expansion so that the application can respond quickly to customers. In order not to get so that consumers see absolutely no actual data is appropriate to use SQL Cache Dependency - for changes in the data cache becomes invalid.
- Adding a graphical representation of statistical information - could implement various kinds of graphs for the portion of the principal vs. interest amount in each payment.
- Improving and expanding the idea with setting the interest rates for different periods and price amounts.
- Find a comfortable interface solutions for the presentation and editing of the profile details/ interest and insurance rates.

Comparison between the expectations and the realized system

The designed and developed applicaiton meets to the maximum possible extent the expectations and the problem of the POS credits repayment calculator:

- Ensures the participants in the process of calculation means to satisfy those indicated for each of them needs. This happens through the implementation of easy and friendly web application, and through the design and implementation tailored to the specificities of the needs of the participants.

- Solves the reviewed problems and thus supports the efficient organization and control of the calculation of POS credit repayment

The approach used in the design and development enables the need to expand the scope, the emergence of new needs or problems the system to be relatively flexible.

References

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

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

<https://msdn.microsoft.com>

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

EUROPEAN STANDARD FORM TO
INFORMATION ON CONSUMERPART I. IDENTIFICATION AND CONTACT
CREDITOR / CREDITOR INTERMEDIARY

1. Creditor	New Investment Bank
2. Address	Bulgaria, Sofia 1000
3. Phone Number	0888 55 66 77
5. Fax	02/888 55 66
6. Web Site	newBank.com
7. Credit Intermediary	Investment Bank

PART II. OTHER IMPORTANT CONDITIONS
CONTRACT

1. Credit Type	Consumer credit for acquiring goods and / or services.
2. Credit amount(Maximum amount(limit) or the whole sum, provided on account of the credit contract)	5721.64
3. Credit absorption conditions(How and when is the money received?)	The loan funds are transferred from the creditor to the account of the seller of the goods / insurer chosen by the user.
4. Term of contract	12
5. Indications of the amount, number, terms and dates of the repayments	<p>You must pay the following:</p> <ul style="list-style-type: none">-downpayment: 699-amount of each payment: 514.29-number of payments: 12-maturity date: 06/09/2016 02:05:05 <p>The interests and / or the costs of the credit are due as follows:</p> <ul style="list-style-type: none">-by bank account of New Investment Bank-by pay-desk of New Investment Bank-by e-pay / b-pay
6. The whole sum that you have to pay(the whole amount of the credit(principal) including the interests and costs, which may originate in connection with your credit)	6171.44
7. When it is applicable for the specified type of credit: When the credit is provided under the form of deferred payment for acquiring a good or a service or is connected with the deliverance of a specific good or the provision of a service:	
Name of the good / service:	Laptop DELL Alienware 17 /656738
Price of the good / service in cash:	4699.00
8. When applicable for the specific credit: Required	Not applicable for the specific credit

securities:	
9. When applicable for the specific credit: Credit repayments, which do not lead to immediate pay off of the whole amount of the credit(principal)	Not applicable for the specific credit

PART III. COSTS OF THE CREDIT

1. The interest rate on the credit and the conditions for its application and, if applicable to the type of credit , the different rates which are relevant for the credit.	20 % FIXED
2. Annual percentage rate (APR) The total cost to the consumer , present or future , expressed as an annual percentage of the total amount of the credit. APR allows you to compare various proposals for the conclusion of the credit agreement .	APR - 21.94 %
3. Required to obtain the credit or to obtain it in particular proposed conditions must contract for:	
- Insurance related to the credit agreement , or	140.9700
- Another contract for additional service.	No
4. Related to the contract costs:	
4.1 . Where applicable for the credit type : Costs of opening and maintenance of one or more accounts in connection with the credit agreement.	Fee for opening an account 30.00 BGN Service fee expense - NA
4.2 . Where applicable for the loan type : Cost of using a payment instrument (eg credit card).	Not applicable for this type of credit
4.3 . Where applicable for the loan type : Any other costs associated with the credit agreement.	Not applicable for this type of credit
4.4 . Where applicable for the loan type : The conditions under which the above-mentioned costs related to the credit agreement can be changed.	Not applicable for this type of credit
4.5 . Where applicable for the loan type : Obligation to pay notarial fees and expenses of notary.	Not applicable for this type of credit
4.6 . Costs payable in arrears Presence of unpaid contributions can lead to serious adverse consequences for you (including enforcement) and obtaining credit more difficult in the future.	In arrears, you have to pay legal interest on all overdue amount for the period of notice - from the due date until the date of effective payment.

PART IV. OTHER IMPORTANT CONDITIONS OF THE CREDIT AGREEMENT :

1. Right of withdrawal You have the right of withdrawal from the credit agreement within 14 calendar days.	You have the right , without compensation or penalty and without giving any reason to cancel the credit agreement within 14 calendar days from the date of signing the contract.
2. Early repayment You have the right at any time to discharge fully or partially your obligations under the credit agreement.	You have the right at any time to discharge fully or partially his obligations under a credit agreement. In these cases, you are entitled to reduce the total cost of credit, such reduction consisting of the interest and the costs for the remaining duration of the contract.
3. Where applicable to the type of credit: The creditor is entitled to compensation for early	In early repayment of credit by consumers, the creditor is entitled to compensation for possible costs directly

repayment of the loan.	linked to early repayment of the loan in the amount of 1 percent of the prepaid amount of the loan , when the remaining contract period is greater than one year and amounted to 0.5 percent of the amount of credit repaid early - when the remainder of the contract is less than one year.
4. Inspection of the Central Credit Register or in another database used in the Republic of Bulgaria to assess the creditworthiness of consumers. The creditor is entitled to compensation for early repayment of the loan.	Where a creditor refuses to give you credit based on inspection in the Central Credit Register or in another database of Art. 16 , he is obliged to inform you immediately and without charge of the outcome of reference. This does not apply to cases where the provision of such information is prohibited by European legislation or is contrary to the requirement to ensure public order and security.
6. Where applicable to the type of loan : The period for which the creditor is bound by the predefined information.	Date: 09.06.2015
Part V. When applicable to the type of loan , additional information in the provision of financial services from a distance	Not applicable for this type of credit
Official : (signature and stamp)	