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

Contents

## I. Introduction 4

## II. Business problem 5

1. Research of the field of study 5

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

1.2. Financial institutions 8

1.3. Interest rate calculation 13

1.4. Comparison with other POS credits repayment calculators 17

2. Necessities and problems 18

2.1. Necessities regarding the calculation of POS credit repayments 18

2.2. Problems regarding the calculation of POS credit repayments 20

3. Aim of the application 21

3.1. Business task 22

III. Solution 24

1. Database schema 24

1.1. Tables and relationships 24

1.2. Database schema 38

2. Security system design 40

2.1 Users

2.1.1. User roles 40

2.1.2. User log in 40

2.1.3. User registration 41

2.2 Financial institutions

2.2.1. User roles 40

2.2.2. User log in 40

2.2.3. User registration 41

3. User interface 41

3.1. Menu design 42

3.2. Interface for users with admin rights 44

3.3. Interface for users who are financial institutions 69

3.4. Interface for users who are buyers 79

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

## IV. Realization 89

1. Application type 89

2. Database realization 89

2.1 Entity framework

3. Application realization 97

3.1. Platform, programming language and development environment 97

3.2. Structure of the application 98

3.3. Creation and initial settings of the application 98

3.4. Creation of the initial construction of the application 101

3.5. Creation of the pages for guest users 106

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

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

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

4. Testing the application 135

5. Integration of the information system 136

## V. Conclusion 138

1. Future development of the application 138

2. Achieved results 139

2.1. Comparison between expectations and reality 139

VI. References 141

Introduction

POS Credits Repayment Calculator

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

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

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

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

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

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

Business problem

1. Research of the field of study

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

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

Financial institution:

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

User:

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

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

Registration

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

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

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

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

Registration

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

Selection of a product

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

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

Choice of a financial institution

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

Determining a down payment for the product

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

Determining the credit term (in months)

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

Choice of adding insurance

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

* Life insurance

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

* Unemployment insurance

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

* Purchase insurance

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

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

Single Premium Method

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

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

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

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

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

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

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

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

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

1.2 Financial institutions

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

# Commercial Banks ------------------------------------------------------------------------

Commercial banks accept deposits and provide security and convenience to their customers. Part of the original purpose of banks was to offer customers safe keeping for their money. By keeping physical cash at home or in a wallet, there are risks of loss due to theft and accidents, not to mention the loss of possible income from interest. With banks, consumers no longer need to keep large amounts of currency on hand; transactions can be handled with checks, debit cards or credit cards, instead.

Commercial banks also make loans that individuals and businesses use to buy goods or expand business operations, which in turn leads to more deposited funds that make their way to banks. If banks can lend money at a higher interest rate than they have to pay for funds and operating costs, they make money.

Banks also serve often under-appreciated roles as payment agents within a country and between nations. Not only do banks issue debit cards that allow account holders to pay for goods with the swipe of a card, they can also arrange wire transfers with other institutions. Banks essentially underwrite financial transactions by lending their reputation and credibility to the transaction; a check is basically just a promissory note between two people, but without a bank's name and information on that note, no merchant would accept it. As payment agents, banks make commercial transactions much more convenient; it is not necessary to carry around large amounts of physical currency when merchants will accept the checks, debit cards or credit cards that banks provide.

# Insurance Companies

Insurance companies pool risk by collecting premiums from a large group of people who want to protect themselves and/or their loved ones against a particular loss, such as a fire, car accident, illness, lawsuit, disability or death. Insurance helps individuals and companies manage risk and preserve wealth. By insuring a large number of people, insurance companies can operate profitably and at the same time pay for claims that may arise. Insurance companies use statistical analysis to project what their actual losses will be within a given class. They know that not all insured individuals will suffer losses at the same time or at all.

# Brokerages

A brokerage acts as an intermediary between buyers and sellers to facilitate securities transactions. Brokerage companies are compensated via commission after the transaction has been successfully completed. For example, when a trade order for a stock is carried out, an individual often pays a transaction fee for the brokerage company's efforts to execute the trade.

# Nonbank Financial Institutions

The following institutions are not technically banks but provide some of the same services as banks.

# Savings and Loans

Savings and loan associations, also known as S&Ls or thrifts, resemble banks in many respects. Most consumers don't know the differences between commercial banks and S&Ls. By law, savings and loan companies must have 65% or more of their lending in residential mortgages, though other types of lending is allowed.

S&Ls emerged largely in response to the exclusivity of commercial banks. There was a time when banks would only accept deposits from people of relatively high wealth, with references, and would not lend to ordinary workers. Savings and loans typically offered lower borrowing rates than commercial banks and higher interest rates on deposits; the narrower profit margin was a byproduct of the fact that such S&Ls were privately or mutually owned.

# Credit Unions

Credit unions are another alternative to regular commercial banks. Credit unions are almost always organized as not-for-profit cooperatives. Like banks and S&Ls, credit unions can be chartered at the federal or state level. Like S&Ls, credit unions typically offer higher rates on deposits and charge lower rates on loans in comparison to commercial banks.

In exchange for a little added freedom, there is one particular restriction on credit unions; membership is not open to the public, but rather restricted to a particular membership group. In the past, this has meant that employees of certain companies, members of certain churches, and so on, were the only ones allowed to join a credit union. In recent years, though, these restrictions have been eased considerably, very much over the objections of banks.

## 1.3 Users

The users are regular customers of the store who want to buy a certain good on credit

## 1.4 Interest rates

This is the amount charged, expressed as a percentage of principal, by a lender to a borrower for the use of assets. Interest rates are typically noted on an annual basis, known as the annual percentage rate (APR). The assets borrowed could include, cash, consumer goods, large assets, such as a vehicle or building. Interest is essentially a rental, or leasing charge to the borrower, for the asset's use. In the case of a large asset, like a vehicle or building, the interest rate is sometimes known as the "lease rate". When the borrower is a low-risk party, they will usually be charged a low interest rate; if the borrower is considered high risk, the interest rate that they are charged will be higher.

The interest rate is fixed for the entire term of the purchase credit and is charged on the date of the purchase of premises.

## 1.5 Calculation of the Internal Rate of Return (IRR)

The first index that is calculated is the so called Internal Rate of Return (IRR). The discount rate often used in capital budgeting that makes the net present value of all cash flows from a particular project equal to zero. Generally speaking, the higher a project's internal rate of return, the more desirable it is to undertake the project.

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

Last but not least, the Annual Percentage Rate (APR) is calculated. This 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. This includes any fees or additional costs associated with the transaction.

Loans or credit agreements can vary in terms of interest-rate structure, transaction fees, late penalties and other factors. A standardized computation such as the APR provides borrowers with a bottom-line number they can easily compare to rates charged by other potential lenders.





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

## 1.6 Calculation of the Annual Percentage Rate (APR)

## 1.7 Calculation of the monthly payments

## 1.8 Comparison with other POS credits repayment calculators

# 2. Necessities and problems

## 2.1 Necessities regarding the calculation of POS credit repayments

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

2.1.1 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 is easily done by arranging them in a table view with unambiguous buttons for approval and disapproval. On the other hand, there should also be a quick link to their detailed information where more editing can be done.

2.1.2 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.

2.1.3 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 repayment. 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 and because otherwise the calculated sum will not represent the real amount that is going to be written in the contract with the financial institution.
* 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 buying on credit because it seems hard for them to
* ????????????????????????????????????????????????????????????????????????

# 3. Aim of the application

## 3.1. Business task

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

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

1. Design the database
2. Design the user interface
3. Solution
4. Designing the database
   1. Designing the tables and the relationships between them

Object-relational mapping (ORM, O/RM, and O/R mapping) in computer software is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

In other words, instead of something like this:

String sql = "SELECT ... FROM persons WHERE id = 10"

DbCommand cmd = new DbCommand(connection, sql);

Result res = cmd.Execute();

String name = res[0]["FIRST\_NAME"];

you do something like this:

Person p = repository.GetPerson(10);

String name = p.FirstName;

Entity Framework is highly recommended technology to build any complex system. Entity Framework is more powerful then ADO.NET and LINQ to SQL. Object Relational Mapping (ORM) is main technology that Entity Framework is used. ORM will generate business objects according to database structure. You can get more information about Entity Framework from our recently published article "What is Entity Framework - Basic of Entity Framework". ORM reduce work code and very simple to use. Developer don't need to write SQL statements to access data. Five reasons for using an ORM tools to get Advantage of ORM.

Here we have discuss several Advantages which give more awareness about Entity Framework.

Advantage of Entity Framework

1. EF reduce code by creating Model instead of create class to access data.

2. Easy and fast Functionality for select, Insert, update ,delete and other CRUD operation.

3. Data access code is under source control. If any Database Modification required, no need to change

Data access logic. You have to just change model or business object.

4. Easy to manage relationship between tables.

5. Faster Development approach then ADO.NET.

6. Code is also usually much neater and more maintainable

7. Conceptual model can be represented in a better way.We can have all data access logic written in higher level languages.

The conceptual model can be represented in a better way by using relationships among entities.

The underlying data store can be replaced without much overhead since all data access logic is present at a higher level.

and finally, the last question that whether it is an alternative to ADO.NET, the answer would be "yes and no". Yes because the developer will not be writing ADO.NET methods and classes for performing data operations and no because this model is actually written on top of ADO.NET, meaning under this framework, we are still using ADO.NET. So let us look at the architecture of Entity Framework (diagram taken from MSDN):



High-level capabilities of the Entity Framework:

Works with a variety of database servers (including Microsoft SQL Server, Oracle, and DB2)

Includes a rich mapping engine that can handle real-world database schemas and works well with stored procedures

Provides integrated Visual Studio tools to visually create entity models and to auto-generate models from an existing database. 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)

The Repository Pattern

Context

In many applications, the business logic accesses data from data stores such as databases, SharePoint lists, or Web services. Directly accessing the data can result in the following:

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

Objectives

Use the Repository pattern to achieve one or more of the following objectives:

* You want to maximize the amount of code that can be tested with automation and to isolate the data layer to support unit testing.
* You access the data source from many locations and want to apply centrally managed, consistent access rules and logic.
* You want to implement and centralize a caching strategy for the data source.
* You want to improve the code's maintainability and readability by separating business logic from data or service access logic.
* You want to use business entities that are strongly typed so that you can identify problems at compile time instead of at run time.
* You want to associate a behavior with the related data. For example, you want to calculate fields or enforce complex relationships or business rules between the data elements within an entity.
* You want to apply a domain model to simplify complex business logic.

Solution

Use a repository to separate the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. The business logic should be agnostic to the type of data that comprises the data source layer. For example, the data source layer can be a database, a SharePoint list, or a Web service.

The repository mediates between the data source layer and the business layers 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 business logic from the interactions with the underlying data source or Web service. The separation between the data and business tiers has three benefits:

* It centralizes the data logic or Web service access logic.
* It provides a substitution point for the unit tests.
* It provides a flexible architecture that can be adapted as the overall design of the application evolves.

There are two ways that the repository can query business entities. It can submit a query object to the client's business logic or it can use methods that specify the business criteria. In the latter 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.

Interactions of the repository



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. 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. This guidance does not include an example of the Unit of Work pattern. For more information, see [Unit of Work](http://martinfowler.com/eaaCatalog/unitOfWork.html) on [Martin Fowler](http://www.martinfowler.com/)'s Web site.

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 or SharePoint list, 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. Another example is SharePoint, which uses **SPQuery** objects to return **SPListItem** collections. 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](http://martinfowler.com/eaaCatalog/dataMapper.html) 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. For example, the client does not need to know if the product information is retrieved with SQL queries to a database or Collaborative Application Markup Language (CAML) queries to a SharePoint list. Isolating these types of dependences provides flexibility to evolve implementations.

**The Unit of Work Pattern**

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

The Unit of Work pattern isn't necessarily something that you will explicitly build yourself, but the pattern shows up in almost every persistence tool that I'm aware of. The ITransaction interface in NHibernate, the DataContext class in LINQ to SQL, and the ObjectContext class in the Entity Framework are all examples of a Unit of Work. For that matter, the venerable DataSet can be used as a Unit of Work.

Other times, you may want to write your own application-specific Unit of Work interface or class that wraps the inner Unit of Work from your persistence tool. You may do this for a number of reasons. You might want to add application-specific logging, tracing, or error handling to transaction management. Perhaps you want to encapsulate the specifics of your persistence tooling from the rest of the application. You might want this extra encapsulation to make it easier to swap out persistence technologies later. Or you might want to promote testability in your system. Many of the built-in Unit of Work implementations from common persistence tools are difficult to deal with in automated unit testing scenarios.

If you were to build your own Unit of Work implementation, it would probably look something like this interface:

public interface IUnitOfWork {

void MarkDirty(object entity);

void MarkNew(object entity);

void MarkDeleted(object entity);

void Commit();

void Rollback();

}

Your Unit of Work class will have methods to mark entities as changed, new, or deleted. (In many implementations the explicit call to MarkDirty would be unnecessary because the Unit of Work itself has some way of automatically determining which entities have been changed.) The Unit of Work will also have methods to commit or roll back all of the changes.

In a way, you can think of the Unit of Work as a place to dump all transaction-handling code. The responsibilities of the Unit of Work are to:

* 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 value of using a Unit of Work pattern is to free the rest of your code from these concerns so that you can otherwise concentrate on business logic.

**Using the Unit of Work**

One of the best ways to use the Unit of Work pattern is to allow disparate classes and services to take part in a single logical transaction. The key point here is that you want the disparate classes and services to remain ignorant of each other while being able to enlist in a single transaction. Traditionally, you've been able to do this by using transaction coordinators like MTS/COM+ or the newer System.Transactions namespace. Personally, I prefer using the Unit of Work pattern to allow unrelated classes and services to take part in a logical transaction because I think it makes the code more explicit, easier to understand, and simpler to unit test.

The Repository and Unit of Work Patterns

The repository and unit of work patterns are intended to create an abstraction layer between the data access layer and the business logic layer of an application. Implementing these patterns can help insulate your application from changes in the data store and can facilitate automated unit testing or test-driven development (TDD).

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



Bootstrap

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

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

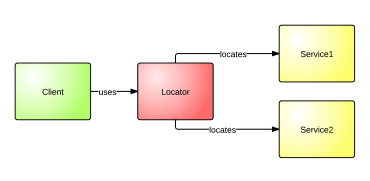
Inversion of control

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

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

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

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



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

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

Dependency injection

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

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

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

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

Ninject

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

Automapper

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

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

PagedList

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

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

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

int pageSize = 3;

int pageNumber = (page ?? 1);

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

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

Rotativa

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

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

1. Realization
2. Conclusion
3. References

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

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

<https://msdn.microsoft.com>

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