Database Application

Project Documentation

ISTE-432 Database Application Development

Music Shop Project

Document (filename): Music Shop Documentation

Team members: Marko Pancirov, Ante Hakstok, Antonio Labriola

Date (last change): 29 November 2017

Version (last version): 5.0

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Description of Change* | *Author(s)* | *Date* |
| 1.0 | First version | All team members | 2017-10-14 |
| 2.0 | Second revised version | All team members | 2017-10-28 |
| 3.0 | Third revised version | All team members | 2017-11-11 |
| 4.0 | Fourth revise version | All team mebmers | 2017-11-25 |
| 5.0 | Final version | All team members | 2017-11-29 |
| 6.0 | Final revised version | All team mebmers | 2017-12-03 |

Table of contents

[1 Introduction 3](#_gjdgxs)

[1.1 Overview 3](#_30j0zll)

[1.2 Purpose and Scope 3](#_1fob9te)

[1.3 Background 3](#_3znysh7)

[1.4 References 3](#_3whwml4)

[1.5 Document Overview 3](#_2et92p0)

[2 Problem Description and Software Architecture 4](#_tyjcwt)

[2.1 Problem Description 4](#_3dy6vkm)

[2.2 Software Architecture 4](#_1t3h5sf)

[2.3 Database Layer and Database Abstraction Layer Description 4](#_4d34og8)

[2.4 Business Layer Description 4](#_2s8eyo1)

[2.5 Presentation Layer Description 4](#_17dp8vu)

[2.6 Areas of particular concern 4](#_3rdcrjn)

[3 Requirements 5](#_lnxbz9)

[3.1 Context 5](#_35nkun2)

[3.2 Functional Requirements 5](#_1ksv4uv)

[3.3 Other (Non-Functional) Requirements 5](#_44sinio)

[4 User Documentation 6](#_2jxsxqh)

[4.1 Graphical User Interface 6](#_z337ya)

[4.2 User Manual 6](#_3j2qqm3)

[5 Installation, Configuration and Acceptance Testing 7](#_1y810tw)

[5.1 Installation 7](#_4i7ojhp)

[5.2 Acceptance Testing 7](#_2xcytpi)

[6 Final Remarks and Conclusion 8](#_1ci93xb)

# Introduction

## Overview

*[Provide an overview of the project and additional information to place application in the context.*

*This serves as a high level description of the domain and the problem and high level description of the desired functionality.]*

The problem today is that all music services lack some feature to make them a complete user experience. We wish to counter this by making the music service that will encompass all the services that other music services offer under one roof. Now you need to use various music providers if you wish to get the full music experience. Our project will offer the users a proper experience of listening to music from a single source rather than having to own multiple accounts on different websites and services. The project will be a music shop desktop app that will provide to the user functionalities for listening and buying products.

## Purpose and Scope

*[Provide an overall description of the document and its purpose as well as intended audience.*

*Discuss the scope of the document.]*

Team members Marko Pancirov, Antonio Labriola and Ante Hakstok are producing the document for the purpose of creating a well defined and clear blueprint for implementation. This document could also be used by possible clients so they can see what the project is about and finally this document is for our Professor Mihaljevic who is a sponsor of this project.

This document covers all areas of the project: documentation, requirements, problem description, software architecture, user manual and installation

## Background and References

*[Describe the project background and who is producing the document and why.*

*List references and other associated documents, including: documentation, standards, policies, deliverables, meeting summaries, white papers, etc.]*

Standards and Policies

1. All changes to this document must be approved by all other team members.
2. Document must be properly formatted and must be consistent.
3. Team meetings are held every other week since the start of the project (2017-10-14) till the end which is planned to be on 2017-12-01
4. All team members must finish their assignments on time

Meeting Summaries

**Metting #1 2017-10-14**

Idea for the project established, assignments given to all team members for the first deliverable, documentation started

**Metting #2 2017-10-28**

Technologies for the project established, assignments given to all team members for the first code deliverable, documentation updated

**Metting #3 2017-11-11**

Discussion about the code, bugs and fixes and discussion about how to proceed, documentation updated, coding assignments assessed and given for the next 2 coding deliverables

**Metting #4 2017-11-25**

Final meeting discussing code, discussion about bugs and fixes, preparation for the final code deliverable, documentation updated

**Metting #5 2017-11-29**

Presentation made and documentation updated, discussion about how to present the demo

Background and References

The team members were thinking about some ideas for the project and the optimal one seemed to be a music shop app. Our app will be similar in some aspects to Spotify because we will also have a music player for songs and the user can download songs and other digital products. Functionalities similar to Beatport are the genre filter and the offer of products for music production like sample packs. Functionality similar to the Google Play (Music) is the ability to buy individual songs.

References:

1. [www.spotify.com](https://www.spotify.com)
2. <https://www.beatport.com/>
3. <https://play.google.com/store/music>

## Document Overview

*[Provide a description of the document organization per chapters.]*

The documents is separated into several sections.

First of we have the Introduction section which covers basic information about the project, how we came up to the idea, and the overview of the documentation. Then comes the Problem Description and Software architecture, which covers what we wish to accomplish with our project. And it covers the information on which covers the software area of the project, mainly which software we’ll use, what databases, technologies… and it will include the problem areas of this project. In the Requirements section the document covers in which cases can be used, what does in need to have to function properly, also if there are some non function related requirements, like does the user need some other technology or knowledge. User Documentation covers the GUI and the User manual for the Users. Installations and Configurations covers covers how to install and configure the project for usage. and Document is concluded with the Final Remarks and Conclusion which gives some insight and thoughts we had while doing this project.

# Problem Description and Software Architecture

## Problem Description

*[Detailed description of the problem that will be solved.]*

The problem with today’s music apps and services is that they lack some features to make them a complete user experience. Some services have certain features but there is not a single service which offers all of those features. Firstly we will offer users not only songs but also other products like programs for making music, sample packs etc. We will include all the popular features from many well known services like Beatport and Spotify like ability to download songs, music player to listen songs and genre filter.

Users will be able to filter products by product type (song, daw, sample pack, plugin), genre (ex. rock, dubstep) and price range. There will also be a pagination system - lazy loading which will display a certain number of products at the time and enable users to navigate through pages. When the user clicks on a certain product a new window will open with all information about that product including product name, price, image cover, genre, product type, list of creators and ratings/comments. There will be a “Add to cart” button which ads product to the cart. On the cart form users can see products in their cart and discount if there is any. Some products will be on discount and users will have discount points which will function in the following way: for every 10$ spent user will get one discount point. One discount point equals 1% discount rate - upper limit is 30% for single purchase.

There will be a button for managing credit cards, button for emptying the cart and a button to buy products in the cart. If the user has enough money on his credit card he will be able to buy products in the cart. Products will be added to the invoice history in the database and an email with the invoice will be sent to the user. Our application will also enable users to have multiple credit cards and manage them (add, update, delete credit card).

Users will be able to see their purchase history and to listen purchased songs and download their digital purchased products (songs, daw, sample pack, plugin).

There will be 2 types of users: general and admin. Admin users will have all the features of the general user with the addition of the ability to administer database tables. Unregistered users can only look at products and only registered users (general users and admin users) can buy products, manage credit cards, see purchase history and listen/download purchased products.

## Software Architecture

*[Capture important software design decisions. Do not forget to explain the choice of language(s), technologies and frameworks, as well as rationale for this design.*

*Describe and sketch Software Architecture structure – decomposition of the software into layers/tiers/modules/units/components and provide detailed software design description for each part.*

*You can describe Software Architectural Styles that you are going to use in this project.]*

Our application will be a Windows Forms application written in C#. We will use MySQL for the database.

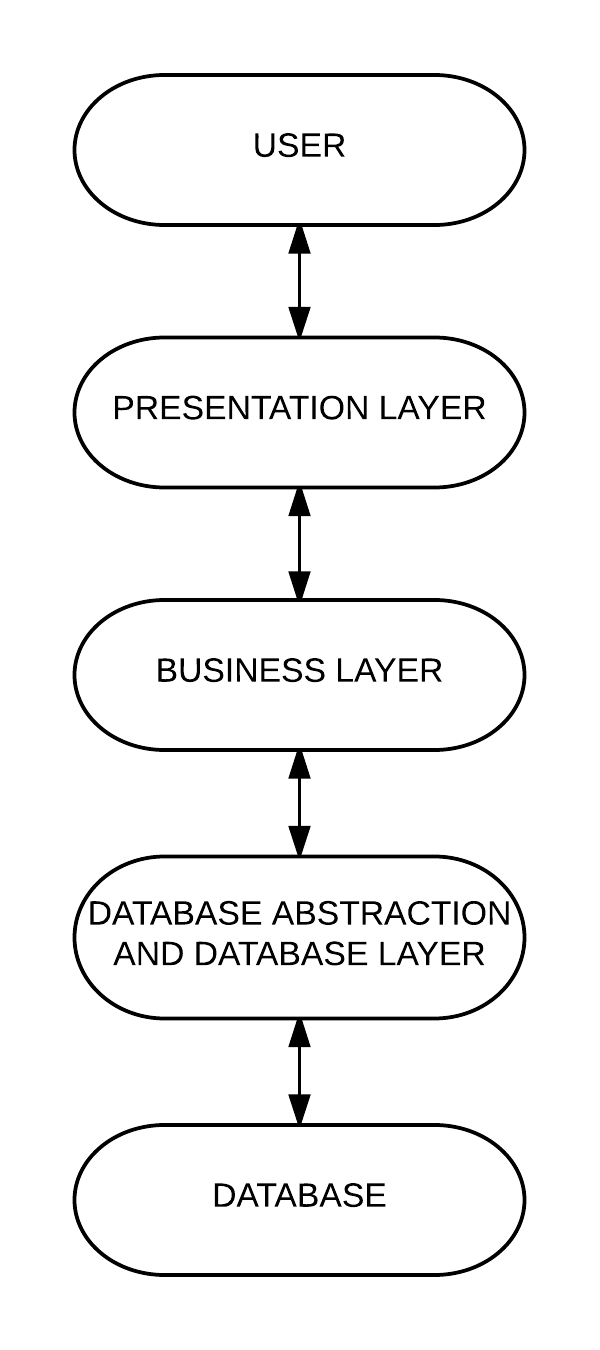
We will use Entity Framework as ORM because it is de facto the default and most used ORM for .NET applications.

So the language is C# which is good for our project since it is an object oriented language and all of the team members know it. The base rationale is the flexibility and versatility of the language. It’s easy to use, both in terms of GUI design (which is very important for our project) and establishing and manipulating connection to the database.

For DBMS we will use MySQL since it is the most convenient to use and we are used to it. It is very easy to install MySQL driver from the NuGet package manager from Visual Studio. It is also very easy to install Entity Framework.

Windows is our targeted platform because most people use Windows and we don’t have enough time and money to make an OSX application.

We chose this technology because it is the most appropriate for our app - we will need many forms for login, registering etc so WIndows Form fits nicely into that paradigm.



We will have 4 layers: database abstraction layer and database layer which will be managed by Entity Framework, business layer which will contain the core business logic of our app and the presentation layer which will communicate with the user.

**Presentation Layer**: User Interface components (buttons, gridviews, labels, textboxes etc.), Presentation Logic components. Example: cart button, login button, add to cart button etc. It will also have event handlers, data/form validation. Example: cart button click event, login button click event, price range validation (must not be empty string or null, “from price” must be smaller than “to price”), genre filter checkbox check event. Presentation layer consist of the following Windows Forms: home page form, product form, login/register forms, cart form, credit card management form, play/download products form, account management form and admin form.

**Business Layer**:

Business layer communicates with the presentation layer and with the database layer. It consists of methods which provide CRUD functionalities on the business/logic level. For example we have a method for getting all the products based on product type, genre and price. There are methods for registering new users, inserting new credit cards, getting genres and product types etc. Business layer calls Entity framework methods for all of the database operations.

**Database Abstraction Layer & Database Layer**: Managed by Entity Framework. It will provide us with methods to access the database and classes which are mapped to database tables. Provides all of the CRUD operations on all of the tables in our database. We used the “database first” approach with Entity Framework -we first created the MySQL database and then we generated the database model in our application with the help of the Entity Framework.

**Other software components and units**

**Exceptions and Logging:**

We catch exceptions in the business layer where the Entity framework methods are called. In some situations we return false if there was an exception so the presentation layer can for example show a user friendly message that the operation failed. We log exceptions to a log.txt file using our custom exception logger (ExceptionLog class) which logs the timestamp, userID of the currently logged in user (if there is one) and the exception message which we customized to provide us with enough detail to fix the problem causing the exception.

We log all database operations to a log table in the database using our custom logger (DBLogger class) which logs currently logged in user (if there is any), timestamp and the database operation that happened.

**Other components:** Other components include: SecurePasswordHasher, customized classes for better data grid preview like the CustomInvoiceDetail and CustomCartDetail class etc.

## Database Layer and Database Abstraction Layer Description

*[Provide database structure with ERD (logical) and database schema (physical), as well as data dictionary explanations of all entities (tables) and their characteristics/attributes/properties (columns).*

*Include all possible information of how the connection to the database will be established and how you intend to use data abstractions (e.g. data access objects, object-relational mapping or similar).]*

**Database Connection**

The connection to the database will be established with the MySql Connector/NET driver for .NET. We are using Entity Framework as ORM. Packages that we needed to install are: EntityFramework by Microsoft, MySql.Data from Oracle and also MySql.Data.Entity. The connection string can be easily added through Visual Studio during development. It can be added through code or changed in the App.config file. We created a form which opens when you start the application. It asks the user to enter server name, user id for the database, password and database name. Our code then creates the connection string and saves it as a static field ConnectionString of the DatabaseConnection class so it can be used throughout the code.

var entityBuilder = new EntityConnectionStringBuilder();

entityBuilder.ProviderConnectionString = "server=" + server + ";user id=" + user + ";password=" + password + ";persistsecurityinfo=True;database=" + database;

entityBuilder.Provider = "MySql.Data.MySqlClient";

entityBuilder.Metadata = @"res://\*/MusicShopModel.csdl|res://\*/MusicShopModel.ssdl|res://\*/MusicShopModel.msl";

ConnectionString = entityBuilder.ConnectionString;

**Object relational mapping**

**Entity Framework**

Entity Framework takes care of object relational mapping - it creates classes for all our tables so we can easily manage the database records by manipulating Entity Framework objects.

musicShopEntities is our DbContext class which we use to perform database operations. Example of usage:

We can get all users with:

var db = new musicShopEntities(DatabaseConnection.ConnectionString);

db.users.ToList().

We can add new user with:

db.users.Add(newUser);

db.SaveChanges();

We can update a user with:

db.Entry(user).State = System.Data.Entity.EntityState.Modified;

We can delete a user with:

db.Entry(user).State = System.Data.Entity.EntityState.Deleted;

For the admin page we bind for example product objects to the data grid view. First we load all products from the database and put them in the grid view with:

db.products.Load();

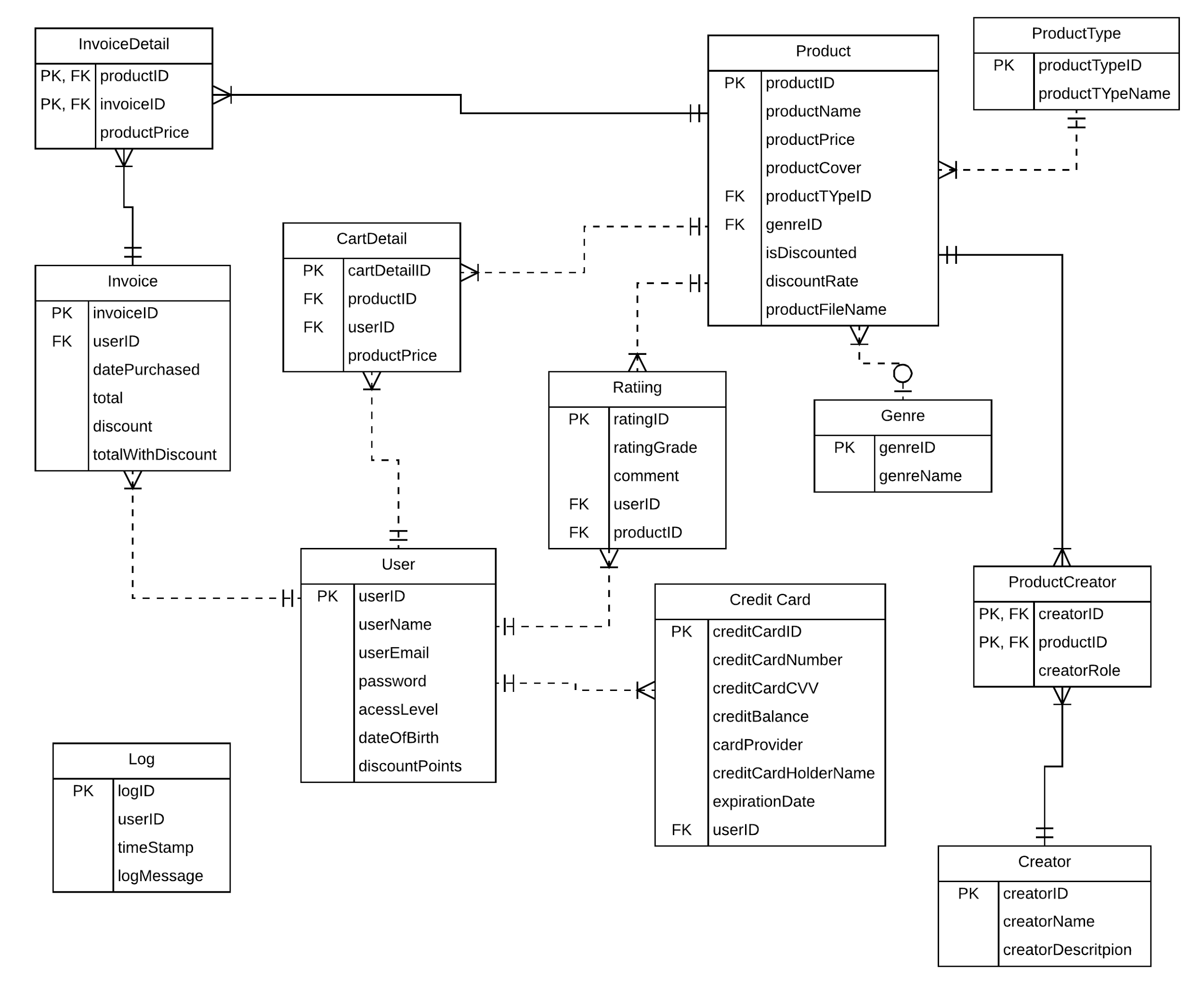
productsDGV.DataSource = db.products.Local.ToBindingList();

Admin can change the products in the grid view - add new product by adding a new row in the grid view, delete a product by deleting a row in the grid view or update a product by updating a certain row in the grid view. When admin clicks “Save” this will save the changes to the product table:

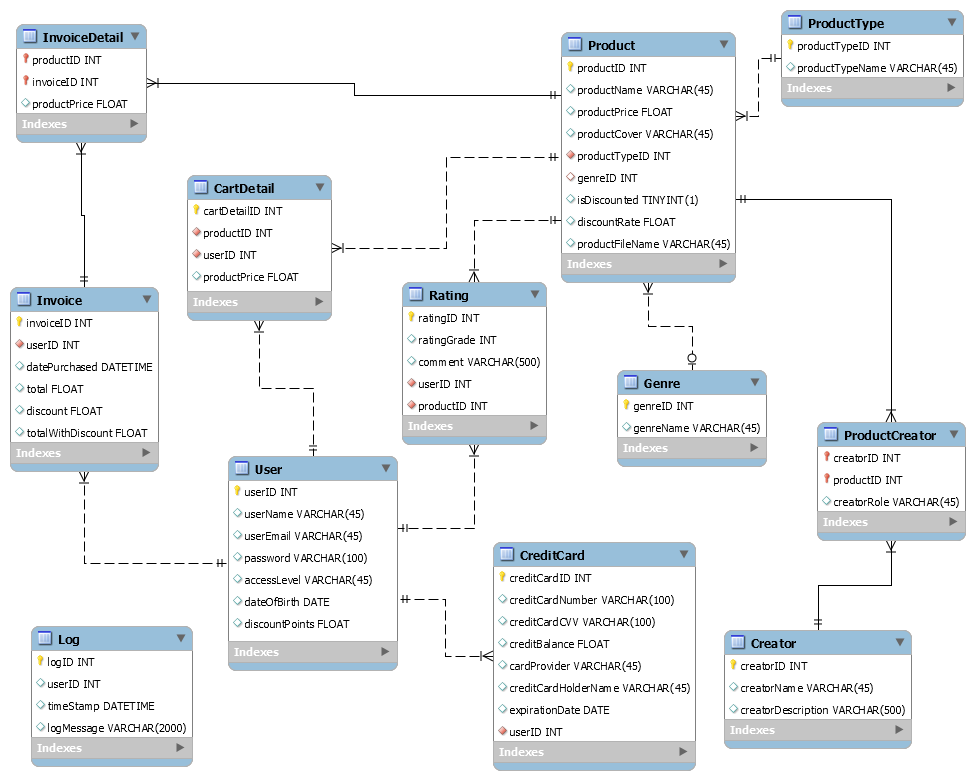
productsDGV.EndEdit();

db.SaveChanges();

**Database Logical ERD**



**Database schema physical**



**Data dictionary**

**Product**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| productID | int | ID of the product, PK, NN | 1;2;3 |
| productName | varchar | Name of the product | “Fear of the dar”, “Ableton” |
| productTypeID | INT | FK referencing producttype, NN | 1;2;3 |
| productPrice | float | Price of the product | 13,99; 1,50; 20.00 |
| productCover | varchar | Image cover of the product | img.png |
| genreID | int | FK referencing genre, nullable | 1;2;3 |
| isDiscounted | tinyint | flag indicating if product is discounted | 1, 0 |
| discountRate | float | discount rate | 24 (%) |
| productFileName | varchar | file name of the product | song.wav |

**ProductType**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| productTypeID | int | ID of the product type, PK, NN | 1; 2; 3 |
| productTypeName | varchar | name of the product type | “Song”, “Daw”, “Plugin”, “Sample Pack” |

**Genre**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| genreID | int | ID of the genre, PK, NN | 1; 2; 3 |
| genreName | varchar | name of the genre | “Rock”, Electronic” |

**ProductCreator**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| creatorID | int | FK referencing creator, PK, NN | 1; 2; 3 |
| productID | int | FK referencing product, PK, NN | 1;2;3 |
| creatorRole | varchar | role of the creator | “Producer”, “Singer” |

**Creator**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| creatorID | int | ID of the creator, PK, NN | 1; 2; 3 |
| creatorName | varchar | name of the creator | “John Smith” |
| creatorDescription | varchar | description of the creator | “John Smith is a music producer...” |

**Rating**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| ratingID | int | ID number of the rating, PK, NN | 1;2;3 |
| ratingGrade | int | rating grade(1-5) | 3 |
| userID | int | FK referencing User, NN | 1;2;3 |
| productID | int | FK referencing Product, NN | 1;2;3 |
| comment | varchar | comment text | “This song is great!” |

**User**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| userID | int | The ID number of the user, PK, NN | 1; 2; 3 |
| userName | varchar | The name of the user | “John Johnson”; “Jake Jackson” |
| userEmail | varchar | The email of the user | “[john.johnson@gmail.com](mailto:john.johnson@gmail.com)” |
| password | varchar | The password of the user | “2342405”; “Y5545ui@3” |
| accessLevel | varchar | The level of access given to the user. | “General”; “Admin”; |
| dateOfBirth | date | The date of birth of the user | 1991/03/03; 1987/12/11 |
| discountPoints | float | points of discount for the user | 24.33 |

**CreditCard**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| creditCardID | int | The ID number of the credit card, PK, NN | 1; 2; 3 |
| creditCardNumber | varchar | credit card number (will be hashed) | “5007 2381 9238 8232” |
| creditCardCVV | varchar | credit card CVV number (will be hashed) | “234” |
| creditBalance | float | credit card balance | 1458.23 |
| cardProvider | varchar | credit card provider | “Visa”; “Master Card” |
| creditCardHolderName | varchar | credit card holder name | “John Smith” |
| expirationDate | date | date of the credit card expiration | 2018-03-01 |
| userID | int | FK referencing user, NN | 1;2;3 |

**CartDetail**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| cartDetailID | int | ID of the cart detail, PK, NN | 1;2;3 |
| productID | int | FK referencing product, NN | 1;2;3 |
| userID | int | FK referencing user, NN | 1;2;3 |
| productPrice | float | product price (when product was put in cart) | 45.23 |

**Invoice**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| invoiceID | int | ID of the invoice, PK, NN | 1;2;3 |
| userID | int | FK referencing user, NN | 1;2;3 |
| datePurchased | datetime | purchase date and time | 2017-11-29 00:38:17 |
| total | float | invoice total before discount | 456.23 |
| discount | float | discount rate for the whole bill | 24.87 (%) |
| totalWithDiscount | float | total minus discount | 342.78 |

**InvoiceDetail**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| productID | int | FK referencing product, PK, NN | 1;2;3 |
| invoiceID | int | FK referencing invoice, PK, NN | 1;2;3 |
| productPrice | float | product price (when product was bought) | 23.67 |

**Log**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Attribute Description** | **Sample Value** |
| logID | int | ID of the log, PK, NN | 1; 2; 3 |
| userID | int | ID of the user | 1;2;3 |
| timeStamp | datetime | time stamp of the log | 2017-11-29 00:38:17 |
| logMessage | varchar | message of the log(db opearation) | “Opened connection at 11/29/2017 12:39:22 AM +01:00” |

## Business Layer Description

*[Provide Business layer design and explanation in details, as well as interfaces to Database layer and Presentation layer.*

*Include all details of how you intend to design business layer, including sublayers, components, and/or external frameworks.]*

The business layer contains the business logic of our application.

Here we have the following methods:

* methods for getting all the products based on product type, genre and price
* methods for getting, updating, adding and deleting credit cards
* method for registering a new user
* methods for getting product types and genres.
* method for getting all users
* method for adding a product into cart
* method for adding new rating
* method for creating the invoice
* method for getting cart total for a specific user
* methods for getting Cart and Invoice details
* method for getting user’s credit cards
* method for filling invoice details
* method for emptying the cart
* method for executing credit card transaction
* method for getting user invoices
* method for getting products that a user purchased

Business layer uses the database layer which is managed by Entity Framework. Entity Framework created for us all the classes which are mappings of database tables. Each method in Business layer first has to create musicShopEntities DbContext object in order to perform database operations. Context gets created and after we perform database operations the context gets disposed because of the using statement. We also log all database operations using our custom logger. Example for adding a new user:

using (var db = new musicShopEntities(DatabaseConnection.ConnectionString))

{

db.Database.Log = s => DBLogger.Log(s);

db.users.Add(newUser);

db.SaveChanges();

return true;

}

For business layer we are not using any external framework. The presentation layer calls methods from the business layer. Each method in the business layer is static which simplifies calling them from the presentation layer. Most methods in the business layer return either true meaning that the operation was successful or they return objects which are passed into the presentation layer and used there.

Each method in the business layer also catches exceptions like DbEntityValidationException, EntityException and DbUpdateException. These are logged using our custom exception logger. Handling of exceptions is mostly done by returning true if the operation was successful or false if the operation failed - this way the presentation layer can know what kind of message to present to the user if something went wrong. Business layer also uses DBUpdateExceptionConstructMessage and ValidationExceptionCustomFormat methods for customizing the exception message that will be logged. It also uses SecurePasswordHasher.Hash method to hash passwords.

## Presentation Layer Description

*[Provide Presentation layer graphical user interface design, structure, layout and explanation.*

*Show prototypes and mockups of all important functionalities with details, also include menus and options.]*

Presentation layer will be done with WIndows forms. We have 9 Windows forms - each offering the user a specific set of functionalities. Typical user workflow would be to:

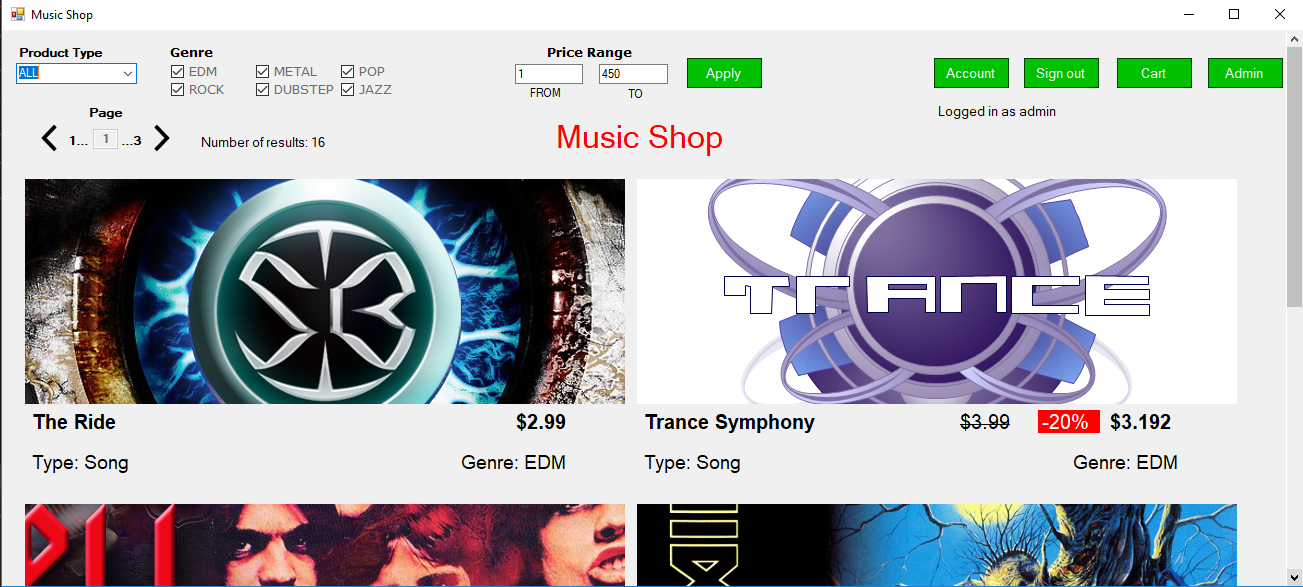
* land on the main Music shop form
* click the login/register button to open the Login or Register forms
* login in or register by inputting data and clicking the Login or Register button
* browse products and click a particular product which will open up the Product form containing information about that product and a list of comments/ratings
* user can post ratings/comments with the Post comment button
* look at the product information and add product to cart with the Add to cart button
* open Cart form immediately after clicking the Add to cart button or from the main page
* buy products in the cart with the Buy all button
* later, user can play/download products on the Play user products form
* user can also open the Account form to see purchase history, open Card management form to manage his credit cards or open the Play user products form to play/download purchased products

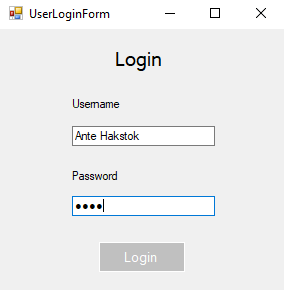
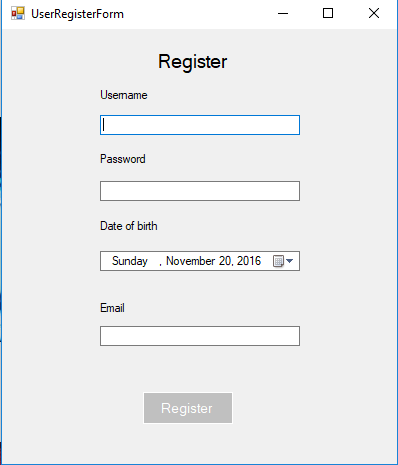
**Music Shop Page Form:**

This is the first page that the user sees. It lists products based on product type(dropdown combo box), genre (checked list box) and price range filter (from - to price). When the user changes the product type using the product type combo box, products in the main form will refresh accordingly showing only the products of the chosen product type. Also, when the user checks or unchecks a genre in the genre checked list box the products will rerfesh showing only the products with the chosen genres. User can input “from” and “to” price text boxes and click the Apply button to apply the price range filter to the products listing which will again refresh the products in the form. Each product name label can be clicked which opens up the Product form.

In the top right corner the main form contains Account, Sign out and Cart buttons. Account button opens up the Account management form. Sign out button signs out the user and the Cart button opens up the Cart form.

Users with different levels of authorization will have different buttons - Admin will have the “Admin” button which opens up a new form where he/she can administer database tables while the general user won’t see the Admin button. Before the user logs in there are only Login and Register buttons in the top right corner. Login button opens up the Login form and Register button opens up the Register form.

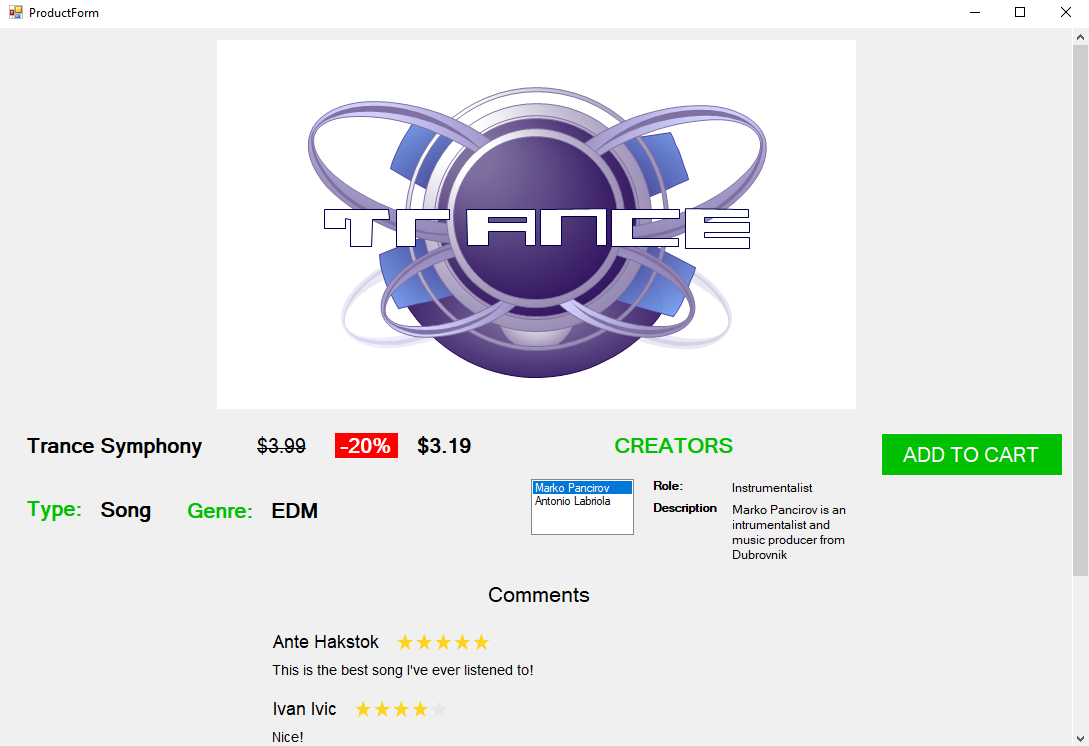




On the Register form user must enter username, password, date of birth and email. The Login form requires username and password. When a new user registers he/she is also automatically logged in.

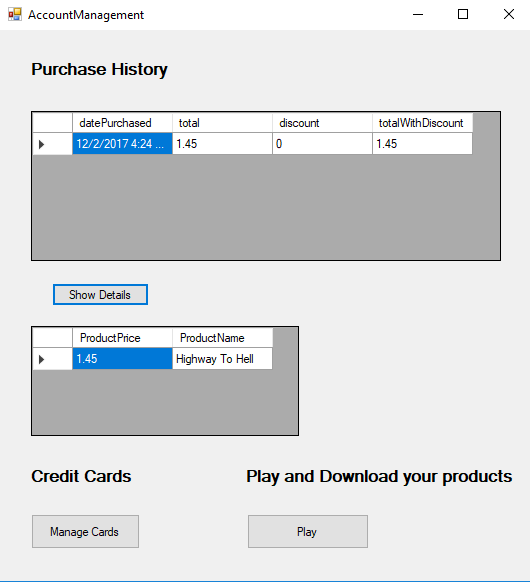
**Product Form**

Product form contains all information about a certain product: image cover, product name, price, discount (if any), list of creators with roles and descriptions, product type, genre (if any), Add to cart button and ratings/comments. User must be logged-in in order to post comments/ratings and to add products to cart.

****

**Account Form**

Account management form contains user’s purchase history, manage credit cards button which opens up the Card management form and the Play button which opens up the form on which users can play songs and download purchased products.



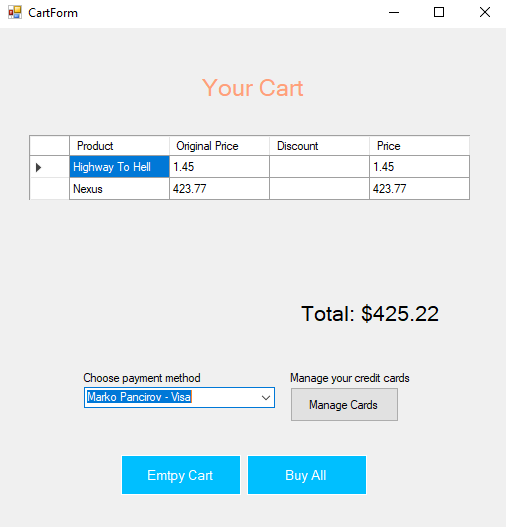
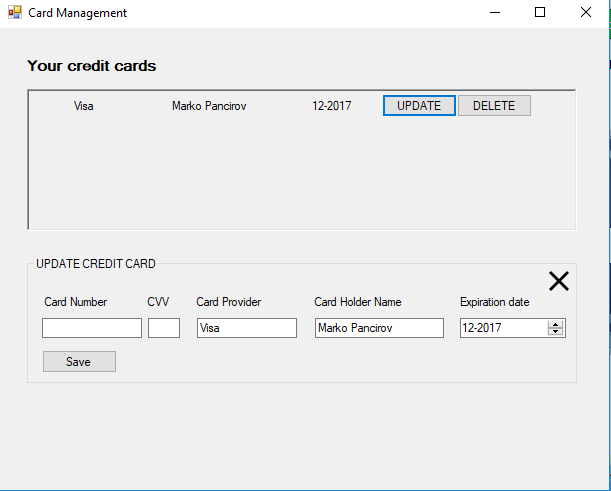
**Cart Form**

Cart form can be opened either from the main page or when user adds a product to the cart (a pop up will ask the user if he/she wants to now open the cart). Cart contains all of the products that the user has put in his/hers cart. Each product has a product name, original price, discount (if the product is discounted) and price with discount (if product is discounted). There is also a total label presenting the user with the total amount in $ of the sum of all prices of products in the cart. If user has discount points then the total price will also be discounted and there will be a label telling the user the price with discount.

In order to buy products in the cart user first has to select the payment method (credit card). To make this more convenient and easy for users there is also a Manage Cards button which opens up the Card management form where user can preview, add, update and delete his/hers credit cards. User can also empty the cart with the Empty Cart button. When the user clicks the Buy All button our application will try to execute the transaction with the credit card that the user selected. If everything goes well a pop will notify the user that the transaction was successful and that an email invoice has been sent to his/hers email.

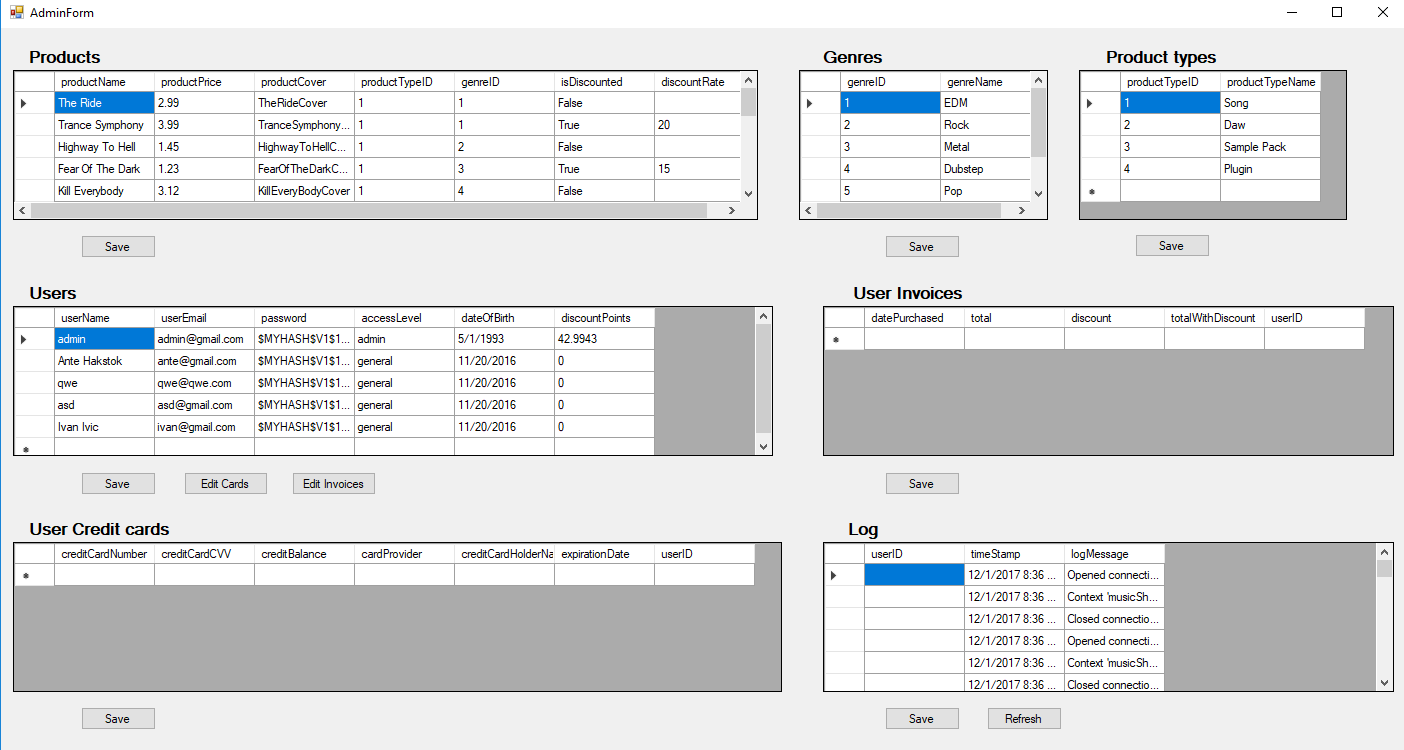
**Card management form**

Card management form lists the user’s credit cards. Each card can be updated or deleted. User can also add a new card - the small windows at the bottom will change accordingly - there is a Add new card button if the bottom window is not opened. The bottom window will open either when the user clicks the Add new card button to allow the user to enter data for the new card or it will open when the user clicks the Update button on an existing card to allow the user to update data for that card.

****

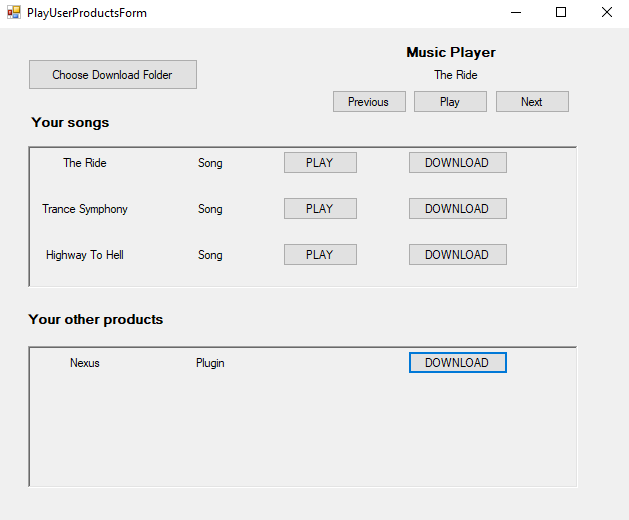
**Admin Form**

Admin form is accessible only to the users with access level of “Admin”. On the admin form user can manage multiple tables. Admin can add a new row by just simply starting to type in a new row, he can update an existing row by just changing data in that row or delete a row by selecting the row and clicking delete button on the keyboard. When the admin is satisfied with the changes he can click the Save button to save changes in the database. If admin wants to edit credit cards or invoices for a particular user he can just select a user and click Edit cards or Edit invoices button. This button click will fill User Credit cards grid view with credit cards of the selected user or the User Invoices grid view with invoices of the selected user. There is also a monitor of the log table. This is useful because admin can see what changes happened in the database and review those changes. Refresh button refreshes the Log grid view. This is especially useful for database administrators.



**Play user products form**

This form enables users to play purchased songs and download purchased products (songs, sample packs, plugins, daws etc.) There is a listing of songs. Each song can be either played or downloaded. In the top right corner there is a music player which can Play/Pause a song or move to the previous or next song. Other products can only be downloaded. To download products user must first selected the download folder where he wants his products to be saved. He can set this folder with the Choose Download Folder button which opens up a folder picker.



## Areas of particular concern

*[Provide Identification of areas of particular note or concerns.]*

Our biggest concern is that in order to use this application the user has to have MySql installed and our database created on his/hers machine so every user will have its own version of the database. In real world, of course, we would make a web service application which would connect to a database (on the server), and a client WIndows Form application (or any other client app) which would just consume that web service. Since this is not subject of this course and not required for the project, we decided that we will not implement it like a web service, but if we wanted to it would be very easy: the client application would retain the presentation layer which would connect to the web service application on the server. The server side would have our business and database layers and could be easily adapted to ASP.NET.

Another concern is that our application can only manage digital products. If in the future someone would want to also sell physical products with our app, significant changes would have to be done in order to facilitate that.

The last concern is performance and security. If millions of users would use our app that could potentially be a big problem since our app is not designed to perform well in such conditions. Security could also be an issue since we have access control only on the application level and not on the database level. If we were making a real world app we would have a web service using https which would encrypt all data between client and server and we would also have database level access control on the server.

# Requirements

## Context

*[Provide a description of the application in the context (users, partners, other systems and applications), with explanations as applicable.*

*The context of an application refers to the connections, interfaces, and relationships between the application and its environment.]*

Our application doesn’t depend on any particular third party application, system or partner.

Our app is used by two types of users: general users and admins. General user can be anyone who registered. Admin must be approved and only then he/she can become administrators. All other users - ones not registered - can only browse through products.

This application is a desktop application that will work on Windows. It connects to a local database. It is a Windows Forms application. It requires .NET and MySQL. It uses MySQL Connector for .NET as a driver to connect to the database. Our application should work on Windows 7, 8 and 10.

Our application can work offline and doesn’t connect to any web service, as we explained in the previous chapter (2.6), so the context and environment of our application is the .NET framework running on Windows operating system.

## Functional Requirements

*[List, name (use some name coding/numbering standard) and explain all key functionalities. You can group functionalities in groups or sets, or create some kind of hierarchy.*

*For all important functionalities you can use UML use-case diagrams with use-case descriptions including use-case name, actors, events flow, exceptions and special requirements.*

*Include user requirements if necessary (users, roles, privileges) and associate with specific functional requirements.]*

**Functional requirements list**

1. list all the products based on product type, genre and price
2. manage (get, update, add and delete) credit cards
3. registering a new user
4. get product types and genres.
5. get all users
6. add a product into cart
7. add new rating
8. create the invoice
9. get cart total for a specific user
10. get user’s credit cards
11. fill invoice details
12. empty the cart
13. execute credit card transaction
14. get user invoices
15. administer database (manage users, products etc.)

**User roles, privileges and requirements based on roles**

**Unregistered user:**

* Ability to preview products
* Ability to register

**General user:**

* Ability to login and sign out
* Ability to buy products
* Ability to post a rating (rating grade + comment)
* Ability to access shopping cart
* Ability to see purchase history
* Ability to manage credit cards
* Ability to play songs
* Ability to download purchased products

**Admin:**

* Ability to manage (preview, add, update, delete) users, products etc.

Admin has all of the privileges of the general user + managing the music shop (database). All roles can preview products. Unregistered user only sees the login/register buttons. General user when logged in sees the Account, Sing out and Cart buttons and the admin sees all of that + the Admin button.

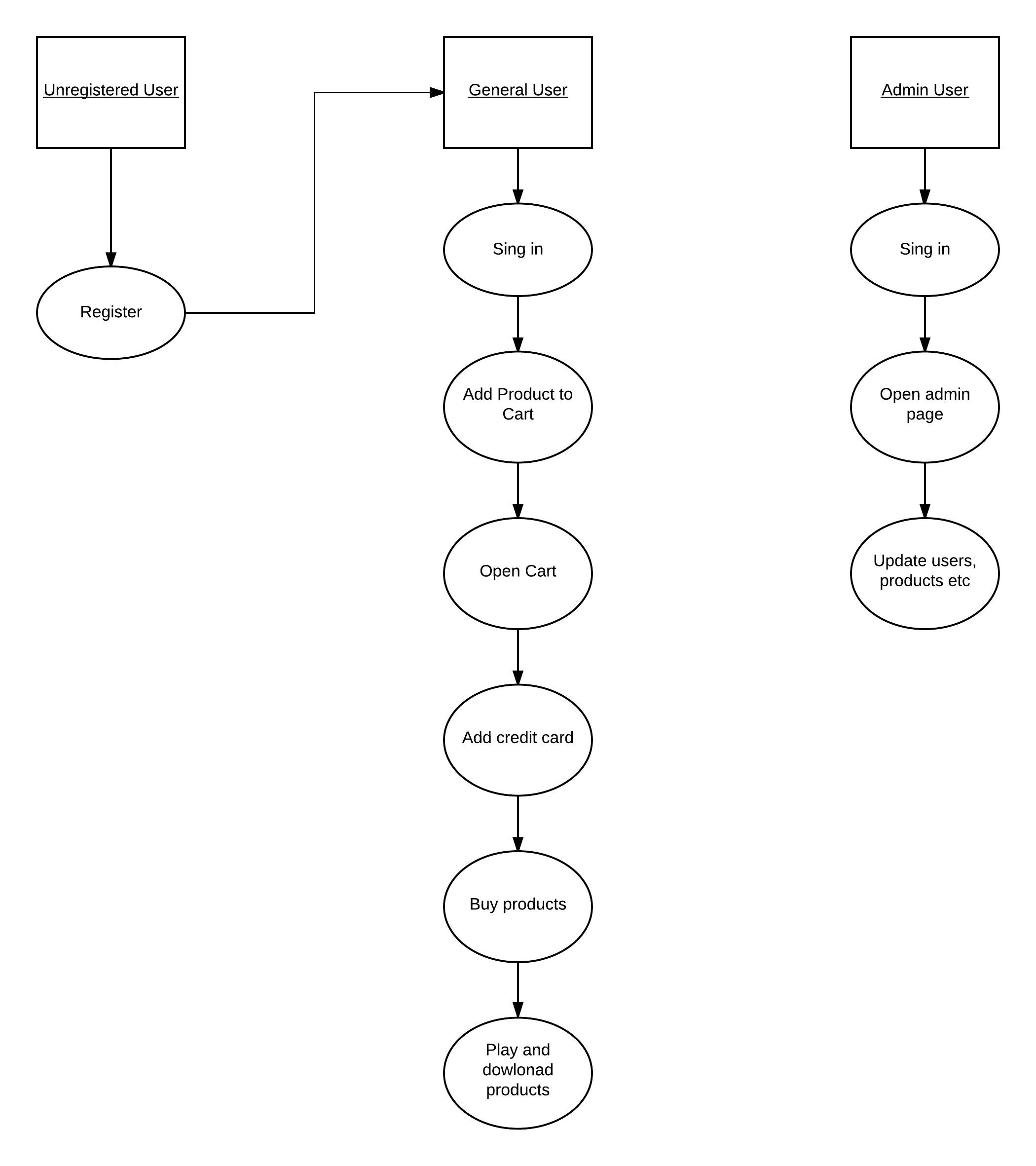
**UML Use Case Diagram**

Here are the 3 most common and typical ways by which a user can use our music shop app.

Unregistered user can simply register to become general user - there are no special requirements besides having an email account.

Registered user becomes the general user. He can add products to cart, manage his credit cards, access shopping cart at any time, buy the products in the cart and listen to purchased songs and download purchased products including songs, daw, sample pack etc.

Admin can do everything that a general user can do + access the admin page after he/she logged in.



## Other (Non-Functional) Requirements

*[Describe the non-behavioral and non-functional requirements, including hardware and software requirements (platforms needed to support the system), interfaces and communication, as well as operational requirements (how the system will run and communicate with operations personnel).*

*You can also provide information about application availability, general performance, capacity, error handling, conventions used, security and similar if necessary.]*

In order for our system to run there are several non-functional requirements:

There needs to be at least one admin person managing the music shop (users, products etc.). There also needs to be at least one technical person for monitoring the exception logs and fixing errors when necessary.

The application is envisioned as a Windows desktop app. The only thing that the user needs to have in terms of hardware and software is a computer with Windows operating system.

The application will have to be simple to increase the accessibility for the users, which also means that the application won’t affect the performance on the system it is used on because it will not use much CPU and it will take less than 50MB of memory on hard drive.

When errors that the system can’t resolve immediately occur a message box will display a user friendly message that the requested operation cannot be performed at the moment.

We explained security and performance concerns in chapter 2.6

# User Documentation

## Graphical User Interface

*[Explain GUI design and user experience considerations, considering how GUI should be used.]*

Our GUI is made up of 9 different screen pages (Windows Forms). We are using standard buttons, combo boxes and text boxes which most users are already very familiar with. Screenshots with explanations are in the 2.5 sections of this document.

We designed our application to be very user friendly and easy to use. On the first main Music Shop page, user can either press the login button to login or the register button to create a new account. Both login and register forms are intuitive to use: just fill in the data and click login/register.

After user logs in, the main page changes login and register buttons to Account, Sign out, Cart and Admin buttons. Admin button will be displayed only if the currently logged in user has access level of “admin”.

Cart form can be opened in 2 ways: from the main page or when adding a product into cart (a pop will ask the user if he wants to open the cart). This is very convenient for the user because he doesn’t have to go back to the main page to open the cart.

Cart form displays products in the user’s cart. User can easily empty the cart or buy products in the cart by clicking the corresponding buttons. If the user hasn’t previously added a credit card he can open the Card management form directly from the Cart form which is very convenient. When the user adds, updates or deletes his cards the payment method dropdown combo box in the Cart form will update so the user can immediately select the card that he wants to use for payment. If the transaction failed a user friendly message will notify the user. If the transaction was successful a user friendly message will notify the user and also tell him that an email invoice was sent to his email. User will receive a nicely formatted email invoice.

Admin page is also very intuitive and easy to use. User can edit the database tables by directly modifying the corresponding grid views on the admin page. We think that our biggest accomplishment regarding the ease of using the user interface is the way in which admin can edit the database tables. At first, our idea was to make a complicated admin page with buttons for each of the operations (UPDATE, INSERT and DELETE), but we scrapped that idea and went for the much more user friendly solution.

Whenever an exception occurs and the application can’t resolve it a user friendly message will appear telling the user that something went wrong.

Whenever the user didn’t satisfy a precondition a user friendly message will also appear. For example if the user clicks the Buy All button in the cart but he didn’t select a payment method. Another example is in the Play user products form. If the user wants to download a product but he didn’t select the download folder a user friendly message will appear telling the user to select the download folder first.

## User Manual

*[Provide a user manual for expected usage of the available functionalities.*

*You can list all functionalities or divide them per user roles, and you have to include screenshots with typical usage descriptions.]*

**Usage of functionalities**

First you will be shown the main Music shop home page

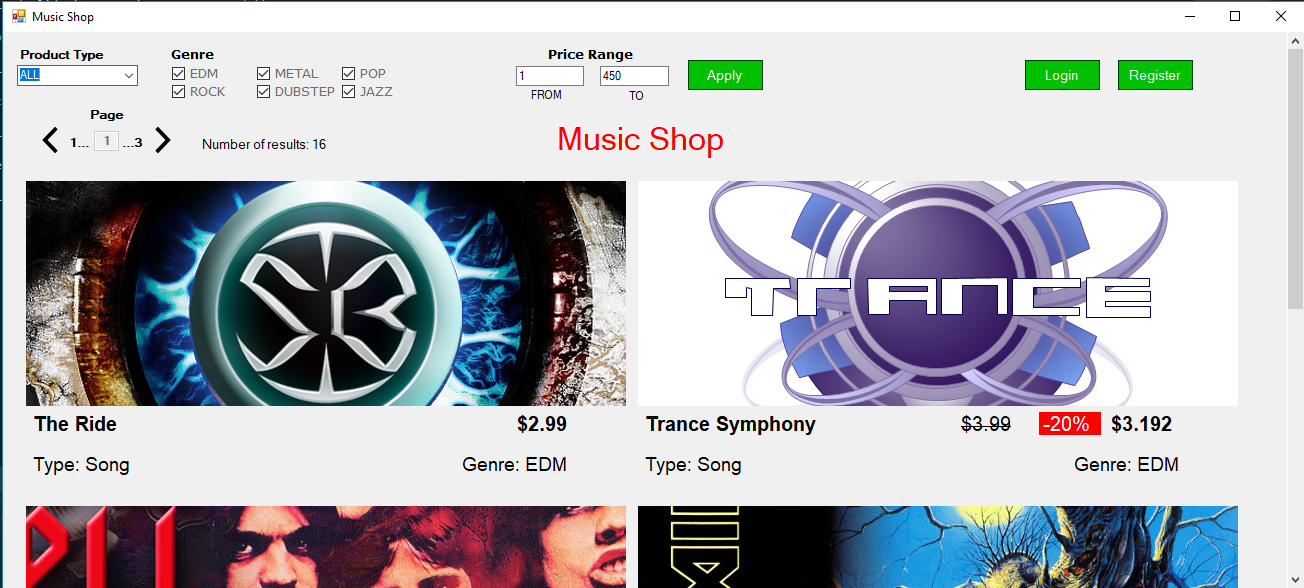
On the main page you can filter products based on product type, genre and price

Select the product type dropdown menu to choose product type

Check genre to choose genre

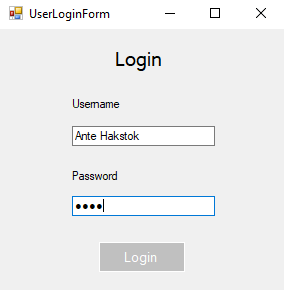
Input from-price and to-price and click Apply to get products in the chosen price range

Click right or left arrow to navigate through results pages



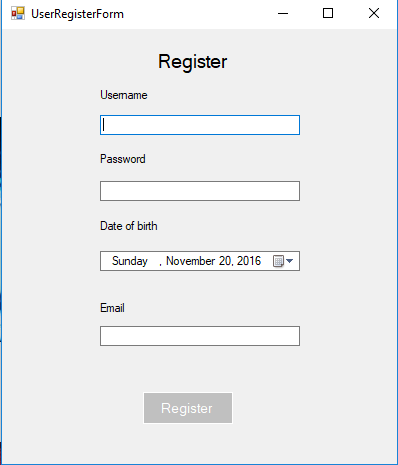
Click the Login button to open Login page

Click the Login button on the Login page to login



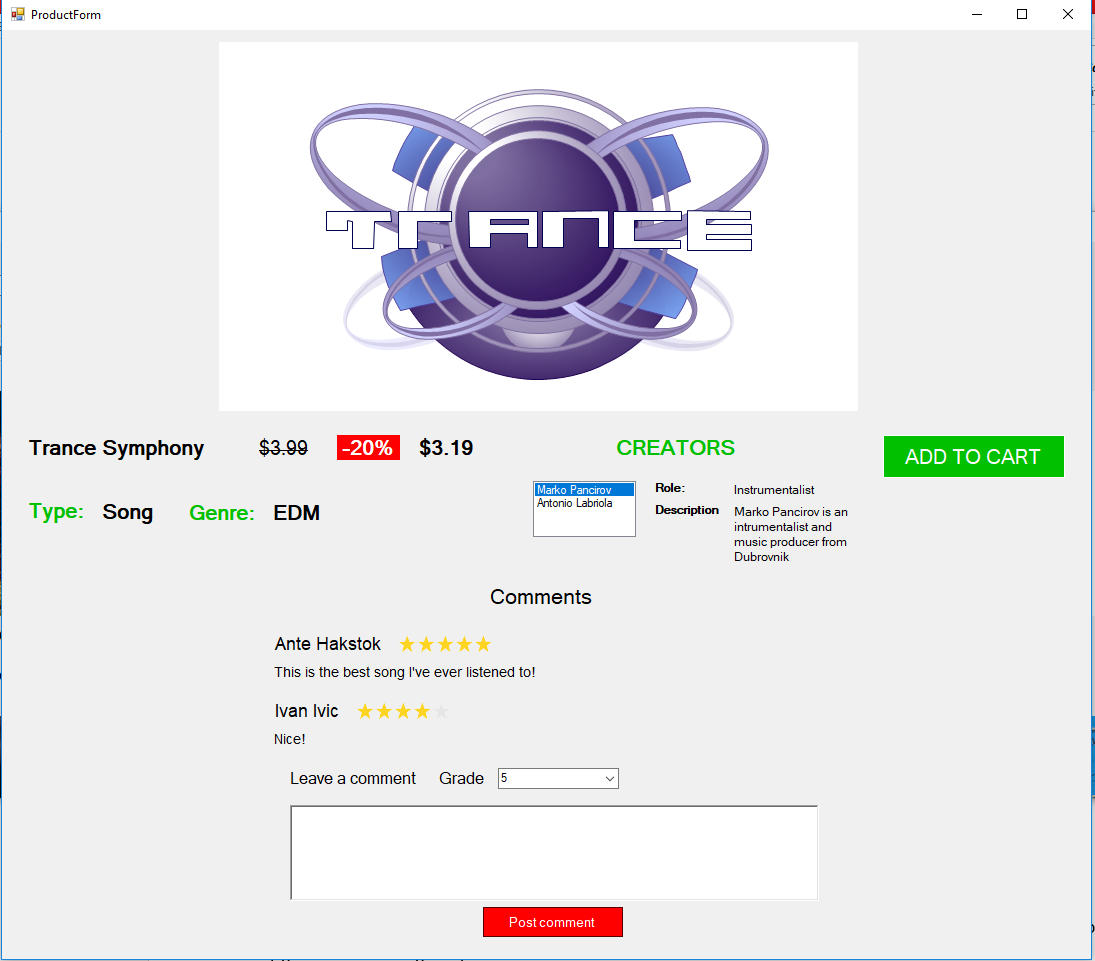
Click the Register button on the Home page to open Register page

Click the Register button on the Register page to register



When you login, on the home page you can click the Account button to manage your account, click SIgn out button to sign out and click the Cart button to open your cart

Click the name of a particular product which will open up the Product page containing information about that product

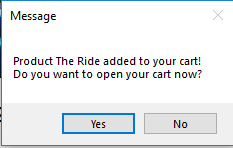


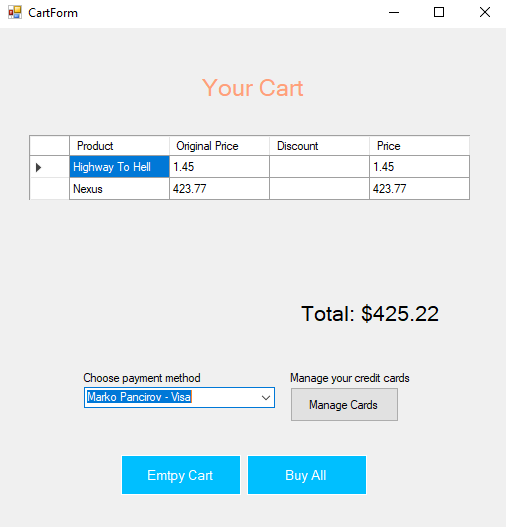
Click on a creator to see its role and description

Choose rating grade from the dropdown menu and input text for comment and click the Post comment button to post your rating and comment

Add product to cart with the Add to cart button

Choose whether to open cart now or not

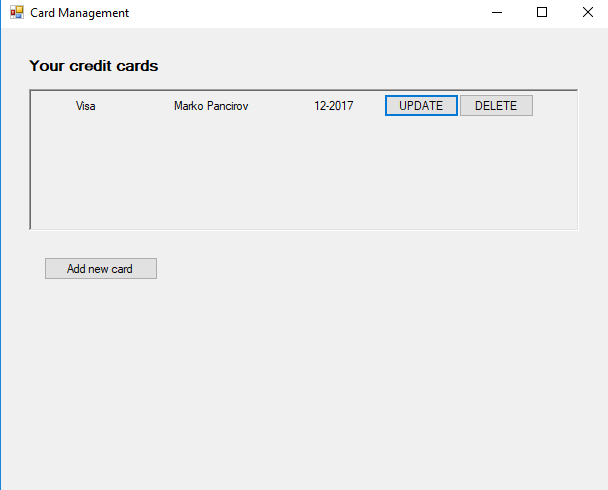




Buy products in the cart with the Buy All button

Empty your cart with the Empty Cart button

Click Manage Cards to open Card management page

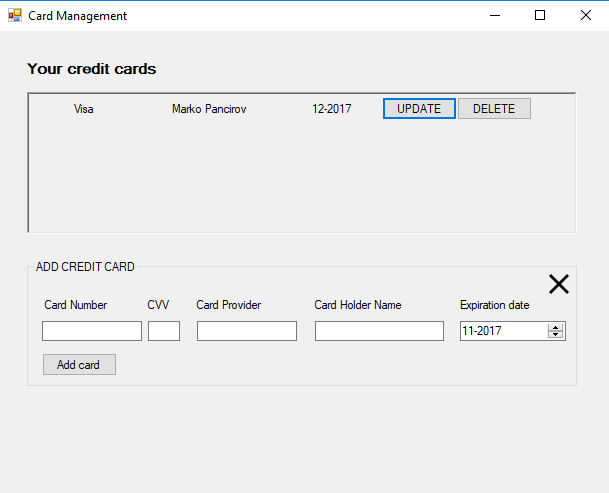


Click Add new card to add a new credit card

When you click Add new card a window will open where you can input credit card data

Click Add card to add your credit card

Click the X to close the Add credit card window

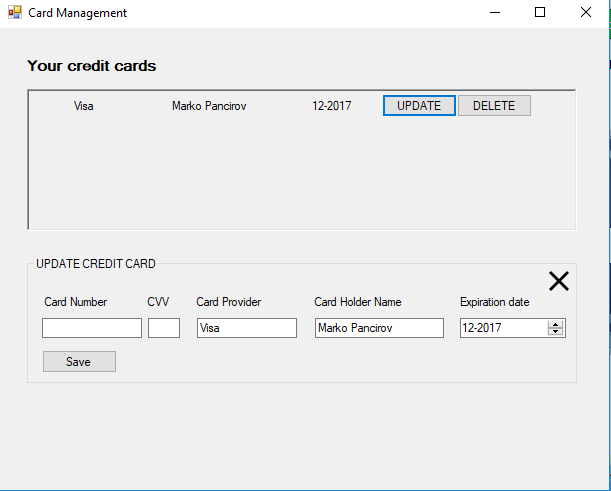


Click Update on an existing card to Update it

When you click Update a window will open where you can input credit card data

Click Save to save changes to your credit card

Click X to close the Update credit card window



Click Delete on an existing card to Delete it

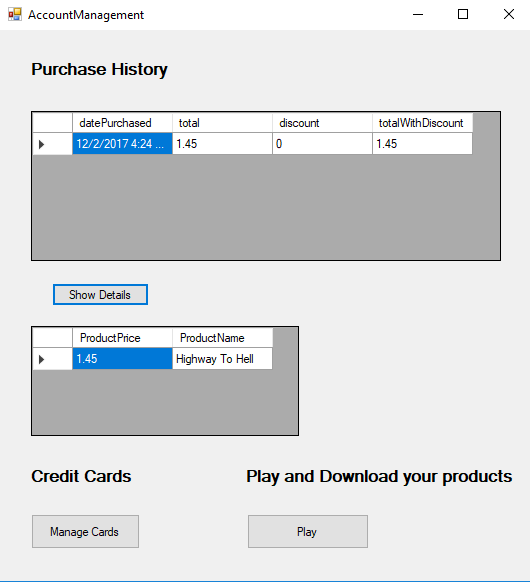
**Account management & product download**

From the main page you can click Account button, this will open the account management page

Here you can see your purchase history

Select a purchase and click Show Details to see products from that purchase

You can click Manage Cards to open Card management page or click Play to open Play products page

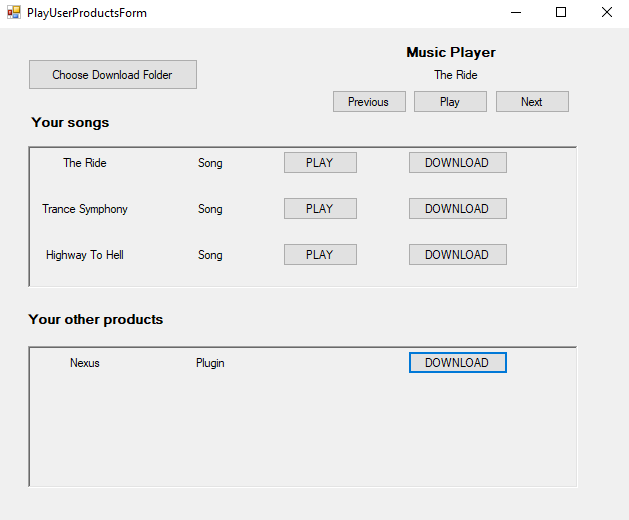


Play products page enables you to play your purchased songs and download purchased products

Click Choose Download Folder to choose a folder in which you wish to save your purchased products

Click Download on any product and that product will be saved in the chosen download folder

Click Play on a song to play it: click Play to play; Pause to pause, Next and Previous to navigate between songs



Admin page is only for admin users. It can be accessed from the home page by clicking on the Admin button

You can do following changes to each table:

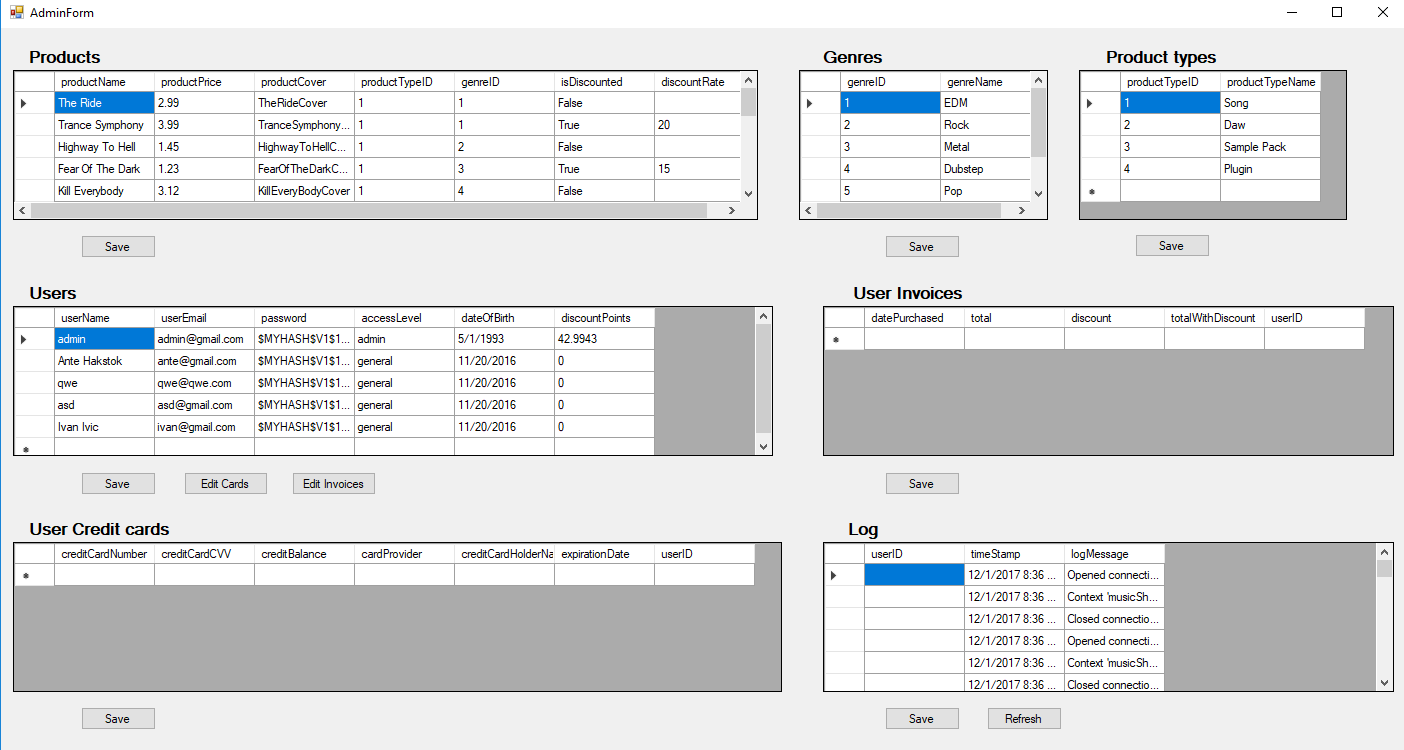
1. add new row - just start typing in a new row
2. update row - change data in an existing row
3. delete rows - select rows and click Delete on your keyboard

When you are done click Save to save changes

To edit user credit cards select a user and click Edit Cards

To edit user invoices select a user and click Edit Invoices

Click Refresh to refresh the Log so you can see changes made to the database



# Installation, Configuration and Acceptance Testing

## Installation

*[Provide technical manual with all prerequisites for application installation, as well as and installation and configuration process with all the details and default configuration values.]*

**Manual for the user**

Download MusicShop.exe file on your Windows computer and double-click it.

**Note to the professor**

**Installation**

No specific kind of Installation is needed because we produced a self containing MusicShop.exe file. User just has to download it, click it and the application will start. This file can be found in the bin/Release folder. Our project is built with the Release configuration for any Windows CPU platform.

**Configuration**

As we explained in 2.6 our application is local so to start our app what the user really needs to do is to run the musicShopDump sql file to create the database and also input database server, username, password and database name when the app starts (on the first form). Database name would be musicshop and other parameters depend on the user’s MySQL settings.

## Acceptance Testing

*[Provide acceptance tests that have to be done to determine if the requirements are met – describe typical testing scenarios and expected results.]*

After inputting database connection configuration on the first form, click Connect and the Music Shop home page will be shown. There should be at least 16 products listed. In the upper left corner you can change the product type, genre filter and the price range filter. The product type dropdown filter and the genre filter update the home page automatically. For the price range filter to apply, click the Apply button. If the list of products changes according to these 3 filters that means that everything is working fine.

We suggest you register as a new user (enter your real email so you can receive an invoice for your purchase). Just click the register button on the home page and fill out the register form. After you click Register, you will automatically be logged in if everything goes well.

You can click on any product (click on product name label) to open a new window showing detailed info about that product. You can post comments (by filling the textbox at the bottom and selecting a grade from the combobox and clicking the Post comment button) and add product to the cart by clicking Add To Cart button. This prompts you to open the cart (click yes) or continue shopping (click no).

On the cart page you can empty your cart or buy all products. To buy products you first have to select the payment method in the dropdown. If you don’t have any credit cards yet then click Manage cards to add a new credit card. When you add a new card the Cart form payment method dropdown should update and give you an option to use that credit card. When you click Buy All button, you will be shown a message box thanking you for the purchase and a notice that we sent you an email with the information about your purchase (invoice). If you entered a valid email address when registering you should receive an email. If transaction fails you will be shown a message box telling you that the transaction failed (the reason could be if you don’t have enough money on your credit card).

Then, return to the home page and click the Account button. This opens up the Account form where you can see your purchase history (you should see the purchases and products that you bought there), manage credit cards or play/download purchased products by clicking on the corresponding buttons. If you click Manage Cards button the Card management form will open. There you can add, update and delete your cards.

If you click the Play button on the Account form the Play user products form will open where you can play purchased songs or download purchased products by clicking on the corresponding buttons. To download a product you first have to select the Download folder by clicking on the Choose Download folder button and selecting a folder in the folder picker. When you click download, the particular product files should appear in the folder that you have chosen.

Now, we suggest you to sign out on the home page and login in as admin so you can test the admin page. Just click login and enter username: admin and password: admin.

Click the admin button on the home page. A new window opens. Try to edit any of the tables and click the Save button. Select a row in the Users table and click Edit Cards or Edit Invoices. This will fill the User credit cards or User Invoices grids enabling you to edit cards and invoices for a particular user. When done editing click the Save button.

If all of the above actions don’t produce a user friendly message saying that something went wrong that means that the application is working as expected.

# Final Remarks and Conclusion

*[Summarize the experiences of this project development, both in terms of the produced results and actual work on the project.*

*List project deliverables produces, as well as positive (and negative) experiences and concerns.*

*Comment on missing functionalities and possibilities for improvement and extensions.*

*Estimate project effort (person-hours) and how it was distributed in time and per team roles.]*

**Experience**

We are very satisfied with produced results because everything is working as expected. The actual development of the project went smoothly without significant setbacks.

**Project deliverables**

* Documentation
* Project code solution (.net)
* Database model in the .mwb format (musicShopModel file)
* Database data and structure SQL script for making the database (musicShopDump file)

**Problems**

We haven’t encountered any serious errors or problems but we can mention a few minor problems that we successfully solved:

* setting up the Entity Framework, .NET environment and the database connection in Visual Studio were issues that we encountered at the beginning of the project
* difficulties with formatting different exception messages
* we designed the database poorly at the beginning but later we improved it significantly

**Missing functionalities & room for improvement**

* We wished to make this project as a web service with mobile app client
* Another place for improvement would be to improve the music player so the user can make a playlist.
* Integrate a real payment system API instead of pretending to know how much money user has on his credit card
* Make this application also work for non-digital goods
* Improve security and performance

**Project Effort**

This project did not take too long to complete as each team member had approximately equal amount of work to do and that saved us a lot of time. We didn’t track exact hours of work but we would guess that each team member did at least 15 hours of work over the course of 5 weeks. Here is the github link:

<https://github.com/mxp7064/MusicShop>

There you can see the commits of each team member