**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙФЕДЕРАЦИИ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПБПУ»)

**Институт среднего профессионального образования**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отделение информационных технологий

Специальность: 09.02.03 Программирование в компьютерных системах

Дисциплина: МДК 02.02 Технологии разработки и защиты баз данных

Группа: 32928/4

**КУРСОВОЙ ПРОЕКТ**

**на тему:**

**«Информационная система\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»**

Студент(ка)

(ФИО) (подпись)

Руководитель

(ФИО) (подпись)

Оценка

(оценка) (подпись) (ФИО)

Санкт - Петербург

2021

**Задание на курсовое проектирование**

**по МДК 02.02 «Технологии разработки и защиты баз данных»**

Специальность: 09.02.03 Программирование в компьютерных системах

студенту(ке) группа\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, инициалы)

Тема: Информационная система \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Исходные данные к проекту:**

1. Литература по описанию объекта информатизации
2. Литература по инструментальным средствам разработки прикладных программ
3. Лекции, стандарты.
4. Интернет – сайты по предметной области

**Перечень подлежащих разработке вопросов:**

1. Аналитический обзор программных средств, технологий, аналогов ПП для создания базы данных и интерфейсов пользователей.
2. Проектирование реляционной базы данных (инфологическая и даталогическая модели с указанием связей (с пояснениями) между таблицами).
3. Разработка механизмов управления данными в БД (добавление, удаление и обновление данных).
4. Организация выборки информации из разработанной базы данных (сформулировать не менее 5 запросов всех типов, реализуемых средствами выбранной СУБД).
5. Проработка технологии доступа к базе данных (определение круга пользователей базы данных и оценка возможности разграничения полномочий пользователей).
6. Разработка ПО для решения поставленной задачи.
7. Разработка алгоритмов программных модулей.
8. Тестирование работы приложения.
9. Оформление документации (руководство системного программиста).
10. Оформление пояснительной записки.

**Руководитель**

(подпись, дата) (инициалы, фамилия)

**Задание принял к выполнению:**

**Студент**

(подпись, дата) (инициалы, фамилия)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

**БД** – база данных.

**WPF** – Windows Presentation Foundation – технология, с помощью которой выполнено приложение.

**MVVM** – Model View ViewModel – паттерн (шаблон) разработки, позволяющий отделить логику приложения от визуальной части.

**EFCore** – Entity Framework Core – ORM-инструмент (object-relational mapping – отображение данных на реальные объекты), в C# коде позволяет обращаться с данными как объектами\классами.

**Введение**

База данных должна содержать данные о списке предлагаемых моделей (выберете заранее марку). По каждой модели известны название модели, стоимость, предлагаемые кредитные предложения (процент и срок, банк), поставщики. Известен список банков (название банка, возможные кредитные предложения) с которыми сотрудничает автодилер. Известен список заводов (от завода зависит стоимость автомобиля – затраты на транспортировку, страна, город размещения, время доставки, тип доставки – ж/д, корабль, самолет, фура). Разработать пакет, состоящий из процедур и функций, позволяющий:

1) для указанной модели машины получить стоимость покупки, список кредитных предложений (вывести ежемесячный платеж и срок выплаты кредита, и список доступных банков), список заводов (и страну с городом) и соответствующие сроки доставки.

2) добавлять/удалять/изменять данные о банках, моделях автомобилей, заводах, банках и их кредитных предложениях.

3) по введенному заводу получить список моделей, которые он предоставляет.

4) предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах.

**Задание сформулировано следующим образом:**

1. Дать сравнительную характеристику популярных СУБД, оценить существующие технологии доступа к данным и обосновать выбор ПО для курсового проектирования.
2. Произвести аналитический обзор имеющихся на рынке аналогичных программных продуктов, оценить их интерфейс и сформировать требования к интерфейсу разрабатываемого приложения с учетом выбранного ПО.
3. Построить инфологическую модель выбранной предметной области, а именно предложить список сущностей и список атрибутов, описывающих их с расстановкой связей между сущностями (ER-диаграмма).
4. Реализовать даталогическое проектирование базы данных, на основе инфологической модели.
5. Разработать интерфейсы программы/веб-приложения по работе с базой и форму отчетов.
6. Разработать алгоритмы (блок-схемы) программных модулей.
7. Предусмотреть наличие исключительных ситуаций.
8. Проработать вопросы по обеспечению целостности БД и разграничение прав доступа пользователей (администратор, редактор, пользователь и т. д.). Защита данных.
9. Составить программу, реализующую поставленную задачу.
10. Проанализировать результаты выполнения разработанной программы / веб-приложения.
11. Подготовить «пояснительную записку» к курсовому проекту.
12. Разработать инструкцию (руководство) системному программисту - администратору информационной системы.

**Аналитический обзор**

В данной предметной области существует автодилер, который продает автомобили определенной марки. По желанию, я выбрал марку Renault.

Автодилер работает с банками, предоставляющие ему кредиты. Кредиты имеют ставку, в % (процентах), и дату окончания кредита, в количестве месяцев.

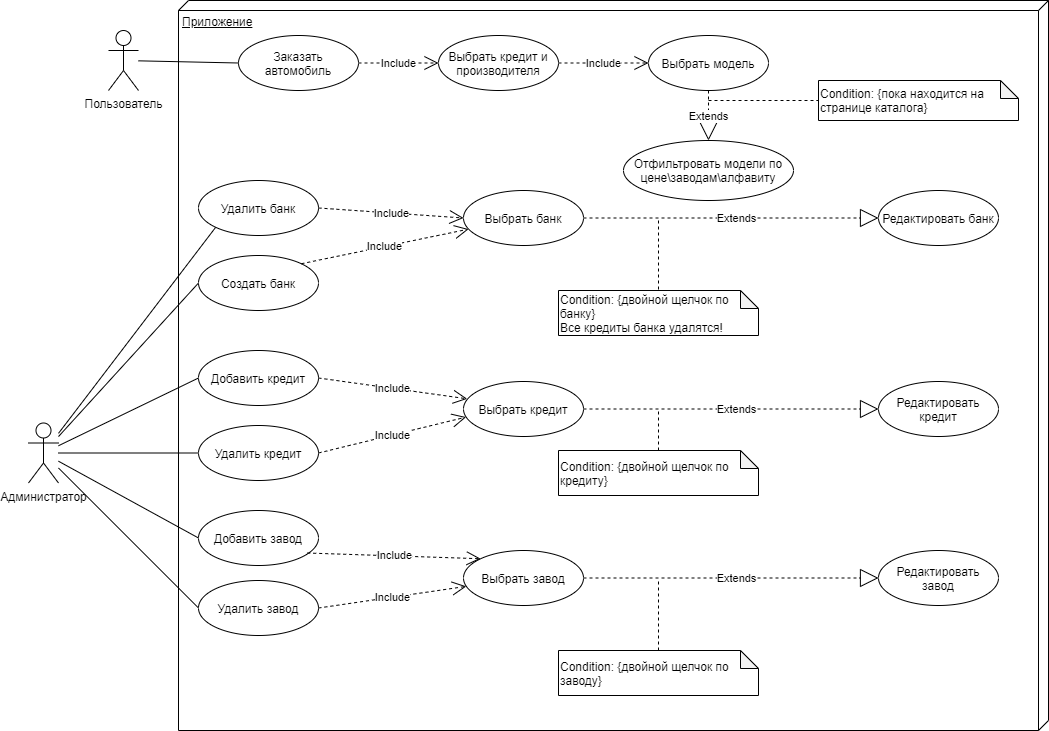
Автомобили производятся на заводах, которые в свою очередь имеют страну и город размещения, затраты на доставку модели и типа перевозки (ж\д, фура, самолет, корабль).

*Примечание:* заводы могут производить множество моделей, как и одну модель может производить множество заводов. Такая же ситуация с кредитами и моделями: один и тот же кредит может предлагаться на несколько моделей, и одна модель может быть куплена несколькими кредитами.

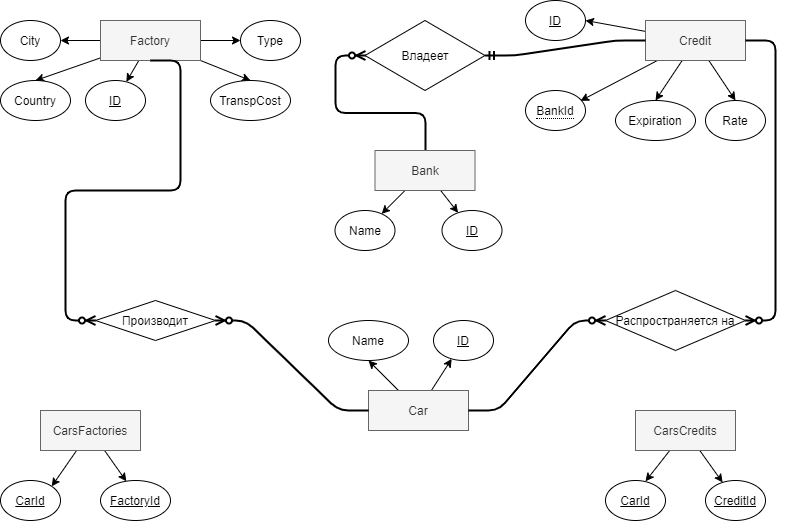
Начиная разработку, я выбирал СУБД, которая бы подошла мне и моей предметной области. Из MySQL, PostgreSQL, MS SQL Server, MongoDB, я выбрал MS SQL Server, так как я наиболее с ней знаком и считаю, что она больше подойдет под эту предметную область, чем документно-ориентированные СУБД, как MongoDB.

Дальше мне нужно было выбрать технологию для создания интерфейса. Мой выбор пал на технологию WPF и принцип разработки MVVM. Данная технология позволит мне спроектировать и создать окно, на котором я смогу отображать страницы с необходимыми данными, а шаблон разработки отделит логику приложения от визуальной части. Визуальная часть в WPF пишется на языке разметки XAML, код – C#.

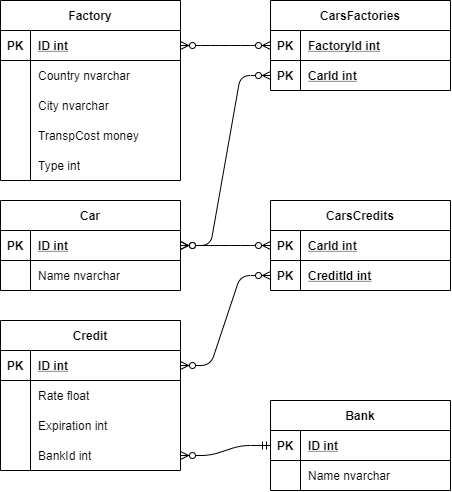
**Диаграмма прецедентов использования (в нотации UML)**

****

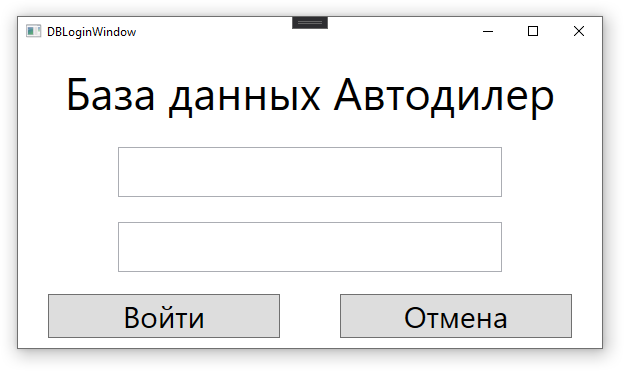
**Инфологическая модель**

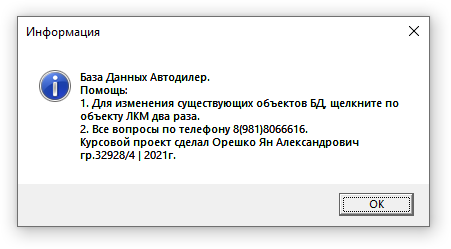
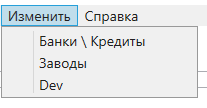
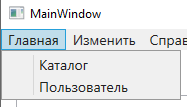
****

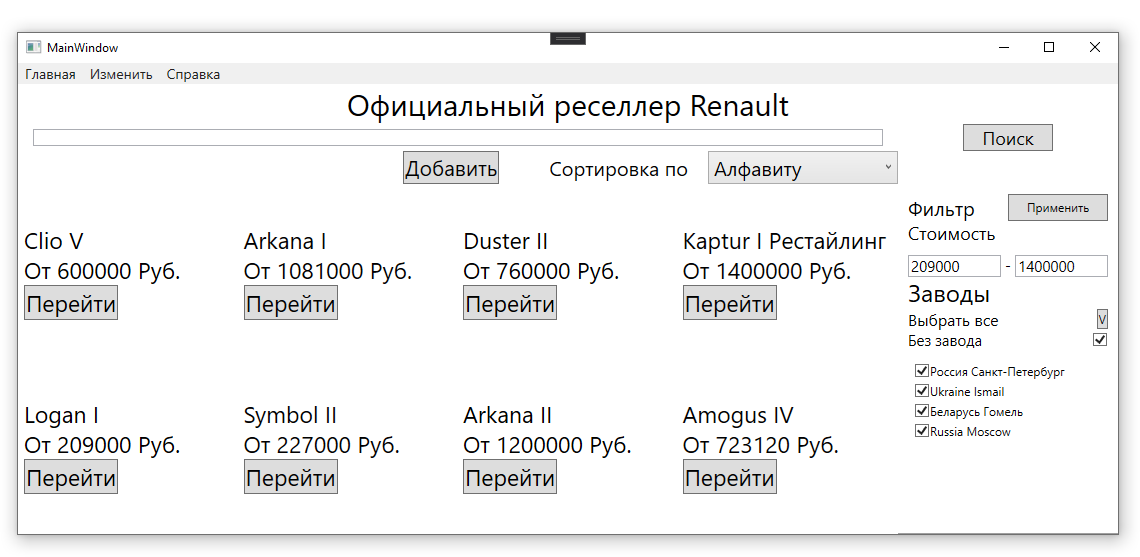
**Даталогическая модель**

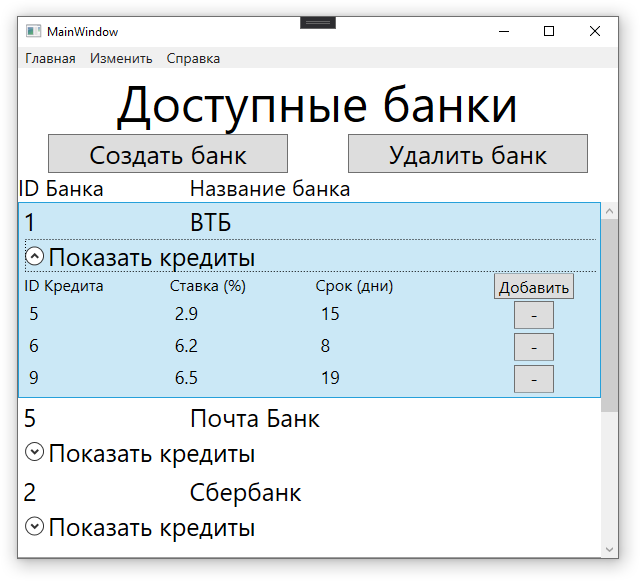
****

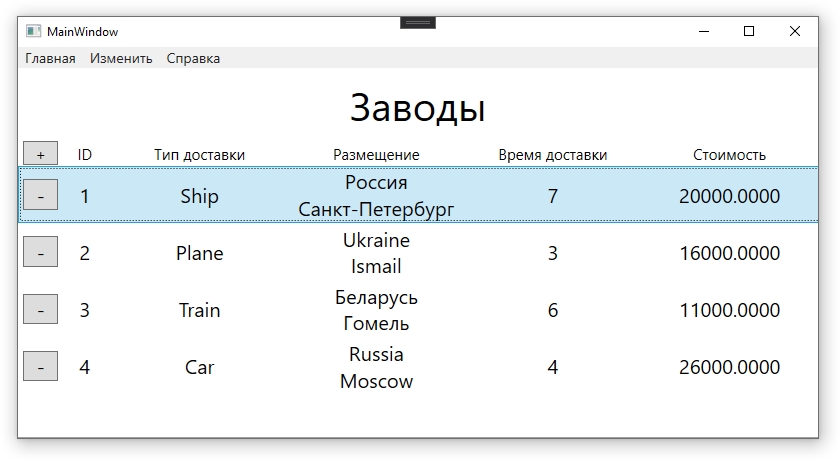
**Основные дисплейные фрагменты (интерфейсы)**

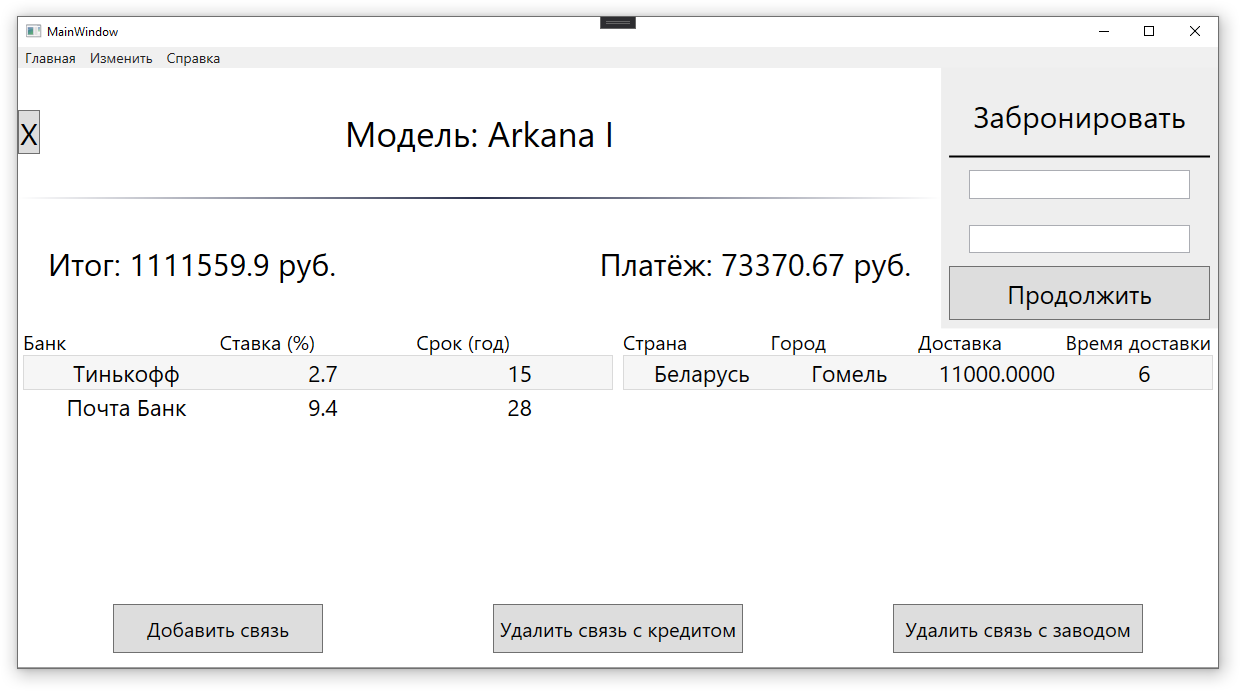
****

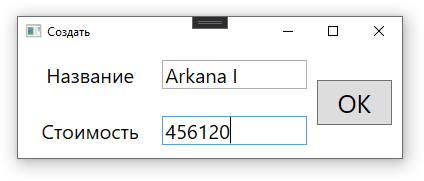
****

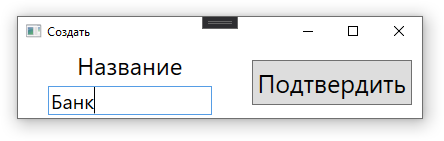
****

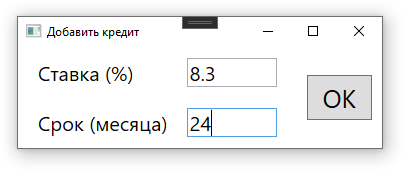
****

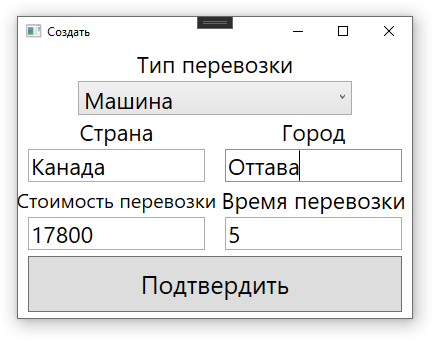
****

****

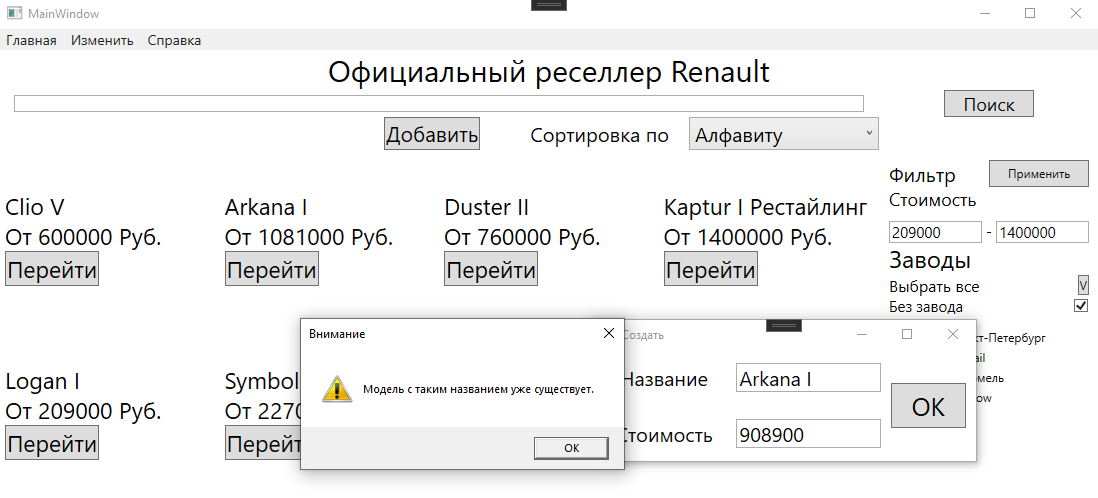
****

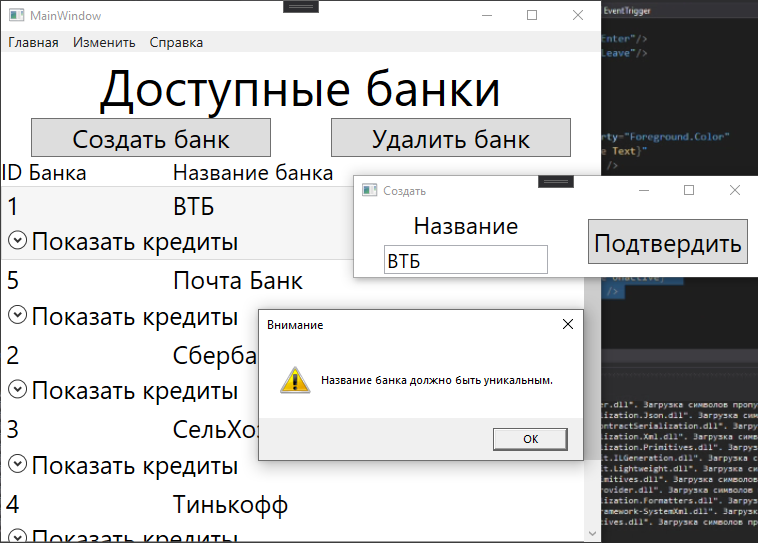
****

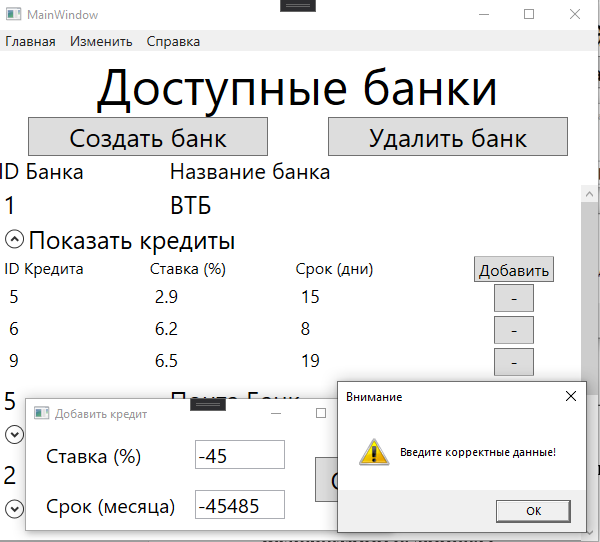
****

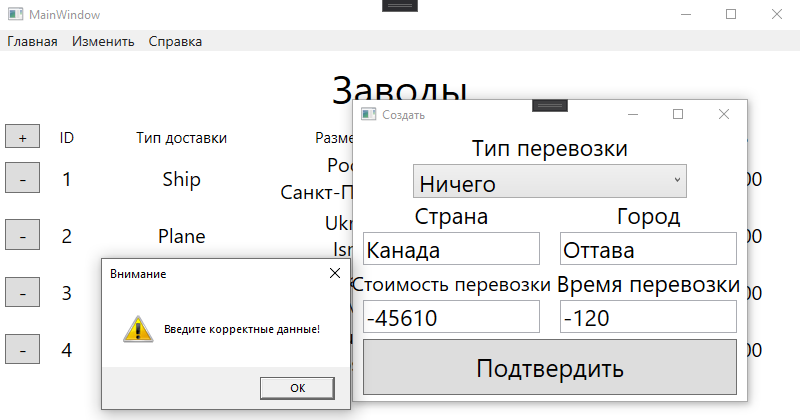
****

**Исключительные ситуации**

****

****

****

****

**Реализация добавления, удаления и обновления**

Ведением БД на стороне приложения занимается EFCore.

**Разработанные запросы**

* Подключение к БД

use CarDealer

* Создание схем

go

create schema Dealer

go

create schema Prog

* Создание таблицы Dealer.Factories

create table Dealer.Factories

(

id int identity primary key,

country nvarchar(50),

city nvarchar(50),

transpCost money,

deliveryTime int,

[type] int

)

* Создание таблицы Dealer.Cars

create table Dealer.Cars

(

id int identity primary key,

[name] nvarchar(30) not null unique,

cost money

)

* Создание таблицы Dealer.CarsFactories

create table Dealer.CarsFactories

(

carId int references Dealer.Cars(id) on delete cascade not null primary key clustered(carId asc, factoryId asc),

factoryId int references Dealer.Factories(id) on delete cascade not null

)

* Создание таблицы Dealer.Banks

create table Dealer.Banks

(

id int identity primary key,

[name] nvarchar(50) not null unique

)

* Создание таблицы Dealer.Credits

create table Dealer.Credits

(

id int identity primary key,

bankId int references Dealer.Banks(id) on delete cascade not null,

rate float not null,

expiration int not null

)

* Создание таблицы Dealer.CarsCredits

create table Dealer.CarsCredits

(

carId int references Dealer.Cars(id) on delete cascade not null primary key clustered(carId asc, creditId asc),

creditId int references Dealer.Credits(id) on delete cascade not null

)

* Добавление строк в таблицу Factories

insert into Dealer.Factories values

('Russia', 'Saint-Petersburg', 20000, 7, 3),

('Ukraine', 'Ismail', 16000, 3, 1),

('Belarus', 'Gomel', 11000, 6, 4),

('Russia', 'Moscow', 26000, 4, 2)

* Добавление строк в таблицу Cars

insert into Dealer.Cars values

('Clio V', 600000),

('Arkana I', 1081000),

('Duster II', 760000),

('Kaptur I Рестайлинг', 1400000),

('Logan I', 209000),

('Symbol II', 227000)

* Добавление строк в таблицу CarsFactories

insert into Dealer.CarsFactories values

(4,4),

(6,3),

(5,1),

(3,2),

(4,1),

(5,4),

(2,3),

(1,2)

* Добавление строк в таблицу Banks

insert into Dealer.Banks values

('ВТБ'),

('Сбербанк'),

('СельХозБанк'),

('Тинькофф')

* Добавление строк в таблицу Credits

insert into Dealer.Credits values

(3, 3.2, 18),

(2, 3.4, 20),

(4, 2.7, 15),

(2, 3.4, 12),

(1, 2.8, 13)

* Добавление строк в таблицу CarsCredits

insert into Dealer.CarsCredits values

(5, 5),

(3, 4),

(1, 4),

(2, 3),

(1, 2),

(4, 2),

(6, 1),

(6, 5),

(4, 3)

* Функция, возвращающая ежемесячный платёж

create function Dealer.GetMonthlyPay

(@cost int, @months int, @koef float)

returns float

as begin

declare @monthly float = (@koef/100) / 12

return @cost \* (@monthly \* power((1 + @monthly), @months) / (power((1 + @monthly), @months) - 1))

end

* Функция, возвращающая сумму процентов

create function Dealer.GetPercents

(@cost int, @months int, @koef float)

returns float

as begin

declare @monthly float = (@koef / 100) / 12

declare @monthPay float = @cost \* (@monthly \* power((1 + @monthly), @months) / (power((1 + @monthly), @months) - 1))

declare @percents float = @cost \* @monthly

while @months != 0

begin

set @cost -= @monthPay - @cost \* @monthly

set @percents += @cost \* @monthly

set @months -= 1

end

return @percents

end

* Функция, возвращающая таблицу с информацией о машине

create function Prog.GetCarInfo

(@model nvarchar(50))

returns @ret table

(

creditId int,

factoryId int,

carCost money,

totalCost money,

monthlyPay money,

expiration date,

bankName nvarchar(50),

country nvarchar(50),

city nvarchar(50),

transpCost money,

arrival date

)

as begin

declare @carId int = (select id from Dealer.Cars where [name] = @model)

declare @creditId int, @months int, @percent float, @bankId int

declare creditCursor cursor local

for

select id, expiration, rate, bankId from Dealer.Credits

join Dealer.CarsCredits on CarsCredits.creditId = Credits.id

where carId = @carId

open creditCursor

fetch next from creditCursor into @creditId, @months, @percent, @bankId

while @@FETCH\_STATUS = 0

begin

declare @cost money = (select cost from Dealer.Cars where id = @carId)

declare @bankName nvarchar(50) = (select [name] from Dealer.Banks where id = @bankId)

declare @factoryId int, @country nvarchar(50), @city nvarchar(50), @arrival int, @transpCost money

declare factoriesCursor cursor local

for

select id, country, city, deliveryTime, transpCost from Dealer.Factories

join Dealer.CarsFactories on factoryId = id

where carId = @carId

open factoriesCursor

fetch next from factoriesCursor into @factoryId, @country, @city, @arrival, @transpCost

while @@FETCH\_STATUS = 0

begin

insert into @ret values

(@creditId, @factoryId, @cost,

round(@cost + Prog.GetPercents(@cost, @months, @percent) + @transpCost, 2),

round(Prog.GetMonthlyPay(@cost, @months, @percent), 2),

dateadd(month, @months, GETDATE()),

@bankName, @country, @city, @transpCost, dateadd(day, @arrival, GETDATE()))

fetch next from factoriesCursor into @factoryId, @country, @city, @arrival, @transpCost

end

close factoriesCursor

deallocate factoriesCursor

fetch next from creditCursor into @creditId, @months, @percent, @bankId

end

close creditCursor

deallocate creditCursor

return

end

* Функция, возвращающая модели, созданные указанным заводом

create function Dealer.GetCarsByFactory(@factoryId int)

returns table

as return

select Cars.\* from Dealer.Cars

join Dealer.CarsFactories on carId = id

where factoryId = @factoryId

* Функция, возвращающая модели с указанным названием модели

create function Dealer.GetCarsByName(@name nvarchar(50))

returns table

as return

select \* from Dealer.Cars

where [name] like '%'+@name+'%'

* Функция, возвращающая авто, у которых цена в диапазоне между @upper и @down

create function Dealer.GetCarsByPrice(@upper money, @down money)

returns table

as return

select \* from Dealer.Cars

where cost <= @upper and cost >= @down

* Процедура, добавляющая Авто и связывающая его с заводом и кредитом

create procedure Dealer.AddCar

(@name nvarchar(50), @cost money)

as begin

begin try

begin transaction

insert into Dealer.Cars values (@name, @cost)

commit transaction

end try

begin catch

rollback transaction

raiserror('Модель с таким названием уже существует.', 14, 1)

end catch

end

* Процедура, добавляющая Завод

create procedure Dealer.AddFactory

(@country nvarchar(50), @city nvarchar(50), @deliveryTime int, @type int, @transpCost money)

as begin

begin try

begin transaction

insert into Dealer.Factories values (@country, @city, @transpCost, @deliveryTime, @type)

commit transaction

end try

begin catch

rollback transaction

raiserror('Ошибка при добавлении завода.', 14, 1)

end catch

end

* Процедура, добавляющая Завод и Авто, связывающая их вместе и кредитом

create procedure Dealer.AddCarAndFactory

(@name nvarchar(50), @cost money, @creditId int,

@country nvarchar(50), @city nvarchar(50), @deliveryTime int, @type int, @transpCost money)

as begin

insert into Dealer.Cars values

(@name, @cost)

declare @carId int = @@IDENTITY

insert into Dealer.CarsCredits values

(@carId, @creditId)

insert into Dealer.Factories values

(@country, @city, @transpCost, @deliveryTime, @type)

insert into Dealer.CarsFactories values

(@carId, @@IDENTITY)

end

* Процедура, добавляющая Банк

create procedure Dealer.AddBank

(@name nvarchar(50))

as begin

begin try

begin transaction

insert into Dealer.Banks values (@name);

commit transaction

end try

begin catch

rollback transaction

raiserror('Название банка должно быть уникальным.', 14, 1)

end catch

end

* Процедура, добавляющая Кредит

create procedure Dealer.AddCredit

(@bankId int, @rate float, @expiration int)

as begin

begin try

begin transaction

insert into Dealer.Credits values

(@bankId, @rate, @expiration)

commit transaction

end try

begin catch

rollback transaction

raiserror('ID выбранного банка не существует.', 14, 1)

end catch

end

* Процедура, добавляющая связь между машиной, кредитом и заводом

create procedure Dealer.AddCarCreditFactoryRef

(@carId int, @creditId int, @factoryId int)

as begin

begin try

begin transaction

if not exists(select \* from Dealer.CarsCredits where carId = @carId and creditId = @creditId)

insert into Dealer.CarsCredits values (@carId, @creditId)

if not exists(select \* from Dealer.CarsFactories where carId = @carId and factoryId = @factoryId)

insert into Dealer.CarsFactories values (@carId, @factoryId)

commit transaction

end try

begin catch

rollback transaction

end catch

end

* Триггер, используемый вместо insert на таблицу Cars

create trigger Dealer.OnCarInsert

on Dealer.Cars

instead of insert

as begin

insert into Dealer.Cars

select [name], cost from inserted

where [name] not in (select [name] from Dealer.Cars)

end

* Триггер вместо delete на таблице Factories

create trigger Dealer.OnFactoryDelete

on Dealer.Factories

instead of delete

as begin

if not exists(select \* from Dealer.Factories

join deleted on Factories.id = deleted.id)

raiserror('Завода(-ов) с указанным id не существует.', 14, 1)

else

delete from Dealer.Factories

where id in (select id from deleted)

end

* Триггер вместо delete на таблице Banks

create trigger Dealer.OnBankDelete

on Dealer.Banks

instead of delete

as begin

if not exists(select \* from Dealer.Banks

join deleted on Banks.id = deleted.id)

raiserror('Указанных банков не существует.', 14, 1)

else

delete from Dealer.Banks

where id in (select id from deleted)

end

* Создание логинов БД

create login Administrator with password = 'password'

create login GuestLogin with password = '123456'

* Создание пользователя для логина Administrator и выдача ему прав

create user [Admin] for login Administrator

grant select, insert, update, delete, execute on schema :: Dealer to [Admin]

grant select, insert, update, delete, execute on schema :: Prog to [Admin]

* Создание пользователя для логина GuestLogin и выдача ему прав

create user GuestUser for login GuestLogin

grant select on schema :: Dealer to GuestUser

grant select on schema :: Prog to GuestUser

**Обеспечение целостности БД**

Целостность базы данных обеспечивается с помощью ключевых слов “on delete cascade” на каждом столбце таблиц, где есть “foreign key” или “references”.

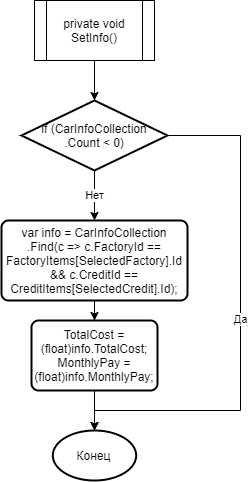
**Разграничение прав доступа. Защита данных**В БД существует два логина и два пользователя:

* Логин “Administrator” с паролем “password” – пользователь “Admin”
* Логин “GuestLogin” с паролем “123456” – пользователь “GuestUser”

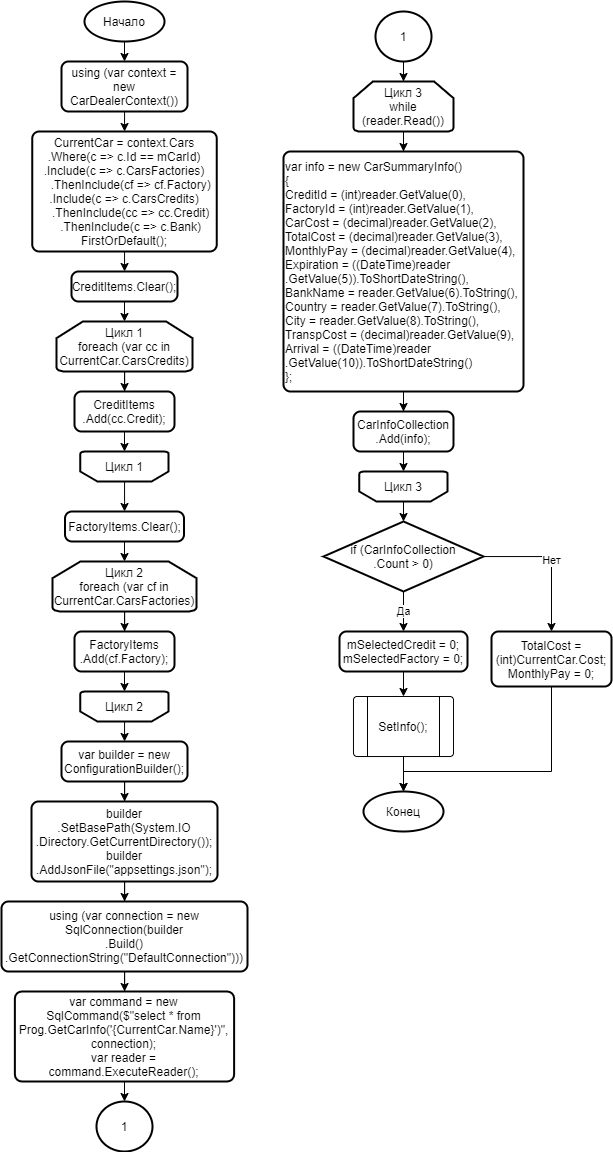
Пользователь “Admin” имеет разрешения на следующие команды: *select, insert, delete, update, execute*. Эти разрешения распространяются на схемы “Dealer” и “Prog”. “Admin” выполняет роль администратора БД, который может изменять данные каждой таблицы, выполнять все процедуры и функции.

Пользователь “GuestUser”, как следует из названия, выполняет роль «Гостя», который не может изменять строки в таблицах. Он имеет разрешения только на команду *select* на схемы “Dealer” и “Prog”.

**Разработка алгоритмов (блок-схемы) программных модулей**

****

Блок-схема функции SetInfo()

****

Блок-схема функции UpdateData()

**Справочная система**

В приложении, вверху находится меню с пунктом «Справка». При нажатии на нее появится диалоговое окно с необходимой информацией по пользованию программой.

**Тестирование приложения**

Цель: Добавить новый банк и вставить в него кредит.

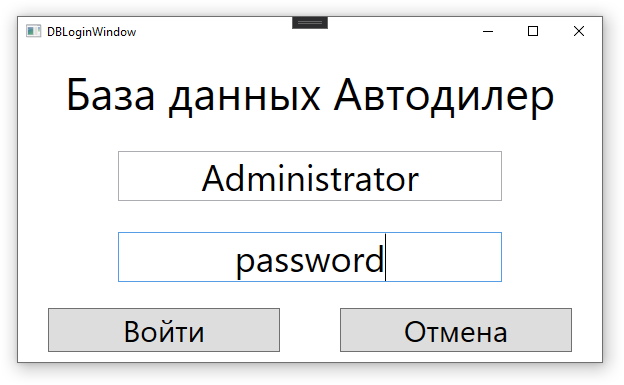


Рис. 1. Вход в БД под логином Administrator.

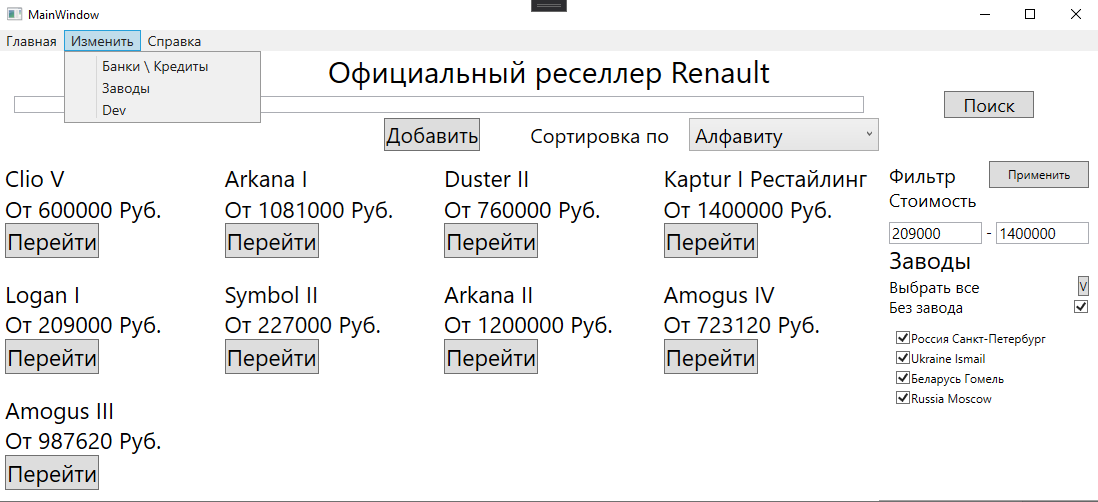


Рис. 2. Выбор пункта меню «Банки \ Кредиты»

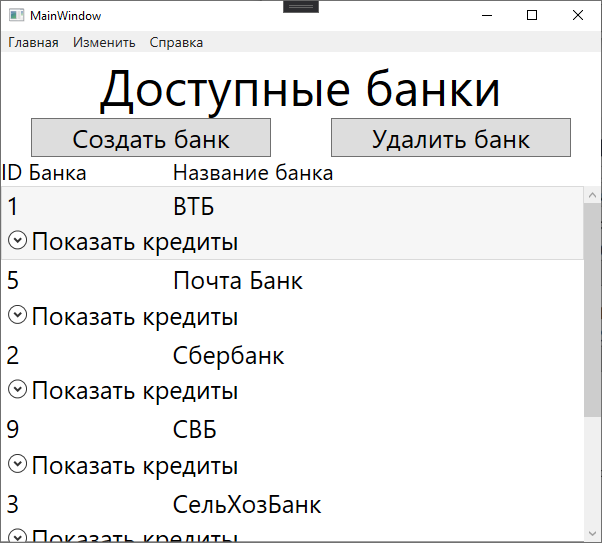


Рис. 3. Страница “BanksPage”

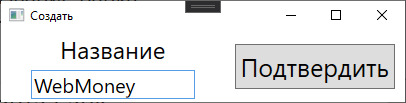


Рис. 4. Окно создания банка

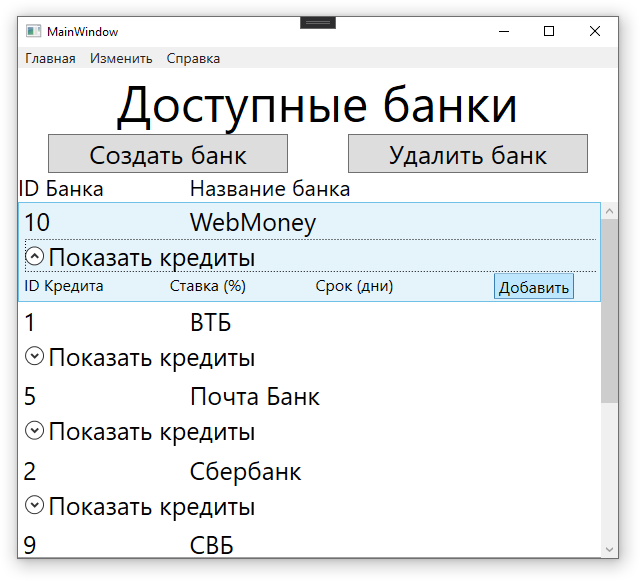


Рис. 5. Страница “BanksPage”

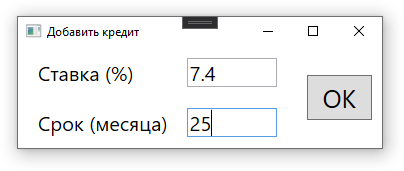


Рис. 6. Окно создания кредита

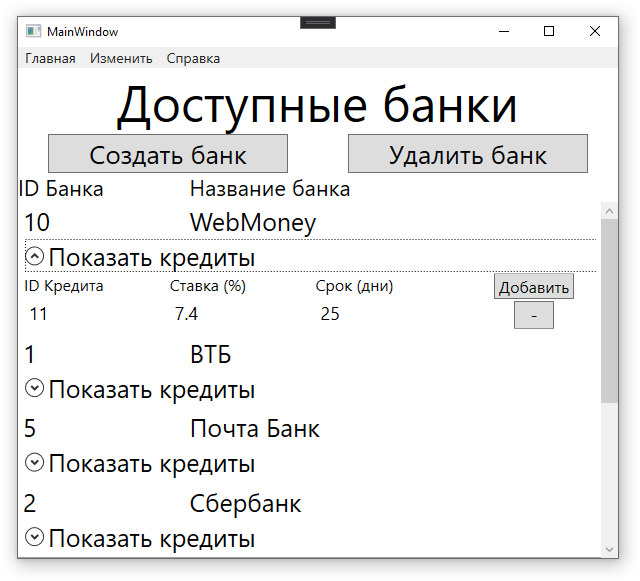


Рис. 7. Результат добавления

**Заключение**

В результате машинного эксперимента приложения, я нашел способ сделать SQL инъекцию и потенциальный способ навредить данным в БД. Это может сделать пользователь, который вошел в приложение под логином “Administrator”, в окнах добавления банка\завода\кредита\машины.

Для решения этой проблемы, следует использовать классы SqlCommand и SqlParameter из ADO.NET, вместо класса DBContext из EFCore.

В остальных аспектах приложения я не нашел проблем, и считаю ее достаточной.

Для создания приложения и БД использовалась операционная система Windows 10 Pro, и следующее программное обеспечение: Visual Studio 19, MS SQL Management Studio 18. В среде разработки Visual Studio 19 я работал с технологиями EFCore, WPF, а также установил несколько NuGet пакетов: PropertyChanged.Fody, Newtonsoft.Json, Extensions.Configuration.

**Список использованных источников**

[https://metanit.com](https://metanit.com/)

[https://dribbble.com](https://dribbble.com/)

<https://professorweb.ru/>