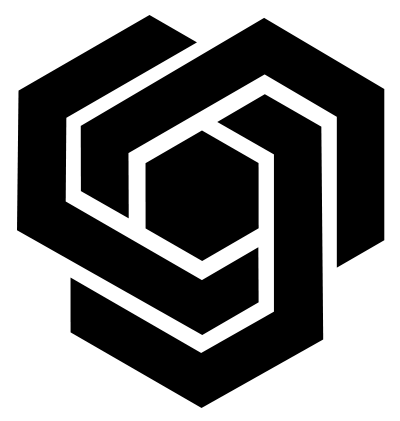
**Т Е Х Н И Ч Е С К И У Н И В Е Р С И Т Е Т - С О Ф И Я**



Направление „ Компютърно и софтуерно инженерство“

**Курсов проект**

**по Програмни среди**

Разработил: Георги Павлов Одобрил: гл. ас. д-р А. Ташева

Катедра: ФКСТ

Фак. № 121214145

Група: 44

**Гр. София**

**2017**

Съдържание

[1.Увод 3](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603291)

[2. Анализ на съществуващи разработки 3](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603292)

[„SDI“ застраховане“ 4](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603293)

[Положителни страни: 4](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603294)

[Отрицателни страни: 4](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603295)

[„24инс“ 4](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603296)

[Положителни страни: 5](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603297)

[Отрицателни страни: 5](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603298)

[„ Застраховане „Broko“ 6](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603299)

[Положителни страни: 6](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603300)

[Отрицателни страни: 6](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603301)

[3.Проектиране 6](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603302)

[Какви данни ще се използват? 7](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603303)

[Как ще бъдат достъпени функционалностите от потребителя? 7](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603304)

[4.Реализация 8](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603305)

[5. Потребителскo ръководство 14](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603306)

[6. Заключение 18](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603307)

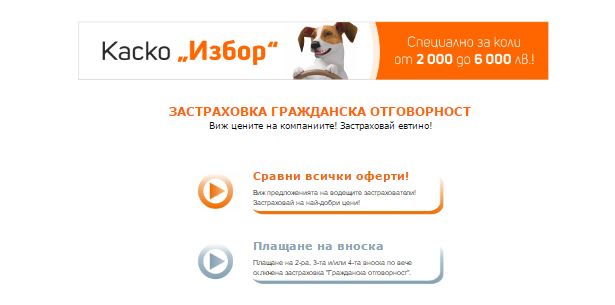
[7.Литература 19](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603308)

[8. Приложение 19](file:///C:\Users\I336317\Downloads\документация.docx#_Toc484603309)

1.Увод

Застрахователни услуги претърпяват динамично развитие в 21 век. Започналото внедряване на застрахователните услуги в интернет довежда не само до подобрено потребителско изживяване но и до засилена конкуренция на пазара и по-добри крайни услуги.

Целта е да се проектира и реализира програма, която да съхранява и управлява информация за автомобили и водачи. Реализира се функционалност свързана с изчислението и издаването на застраховки и проверка за тяхната валидност.  
2. Анализ на съществуващи разработки

Преди да премина към проектирането на разработената от мен системата, ще направя анализ на три съществуващи разработки, наподобяващи моята система и отговарящи на поставеното задание – Информационна система за застраховки „гражданска отговорност“ и „Каско“ Ще изтъкна положителни и отрицателни страни в тяхната реализация и потребителски опит, като в допълнение ще приложа и изображения на

потребителския им интерфейс.

## „SDI“ застраховане

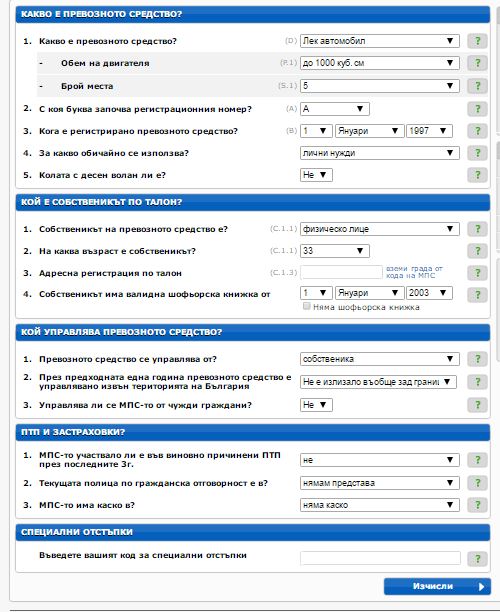
### Положителни страни:

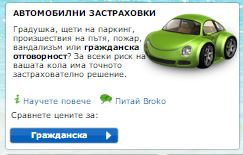
* Семпъл и лесен за навигация интерфейс;
* Приложението позволява повторно попълване на теста.

### Отрицателни страни:

* Приложението няма форма за регистрация(липсва аутентикация на профил);
* Приложението не използа ”https” протокола

## Застраховане „Broko”



****

### 

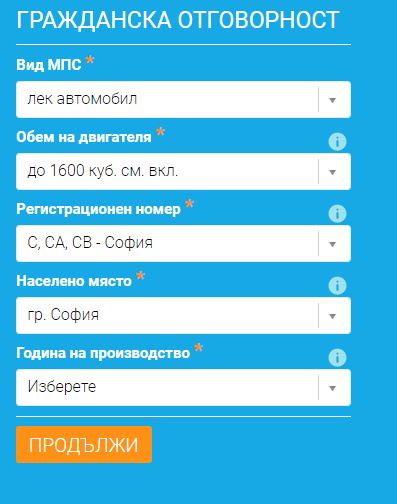
### Положителни страни:

* Безплатно приложение, което го превръща в лесен и достъпен начин за потребителите
* Съществуват бутони за социални мрежи с цел реклама

### Отрицателни страни:

* Лош потребителски интерфейс
* Незащитена комуникация със сайта

## „24инс“



### Положителни страни:

* Семпъл и лесен за навигация интерфейс;
* Безплатно приложение,
* Атрактивен интерфейс;

### Отрицателни страни:

* Приложението няма форма за регистрация(липсва аутентикация на профил);

# 

# 3.Проектиране

Чрез системата да се съкращава процеса на застраховане. Онлайн застраховането спомага клиентите да сключват застраховки при улеснена процедура. При необходимост да се предоставя информация за връзка с застрахователната агенция.

Продуктът ще бъде използван от частни и юридически лица за застраховка и автокаско на техните превозни средства .

Реализацията на система за електронно застраховане чрез прилагане на форма за различни застраховки предлага следните предимства:

1. Съкращаване времетраенето за застраховане;

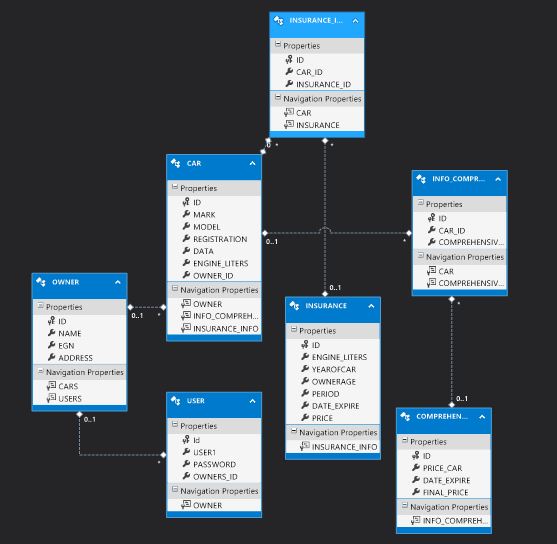
2. Намалява се ангажираността на застрахователните брокери в застрахователния процес;

3. Освобождаване от нуждата за посещение на застрахователен офис;

4. При разглеждания метод за застраховка отпада необходимостта

от ползване на хартиен носител;

## Какви данни ще се използват?

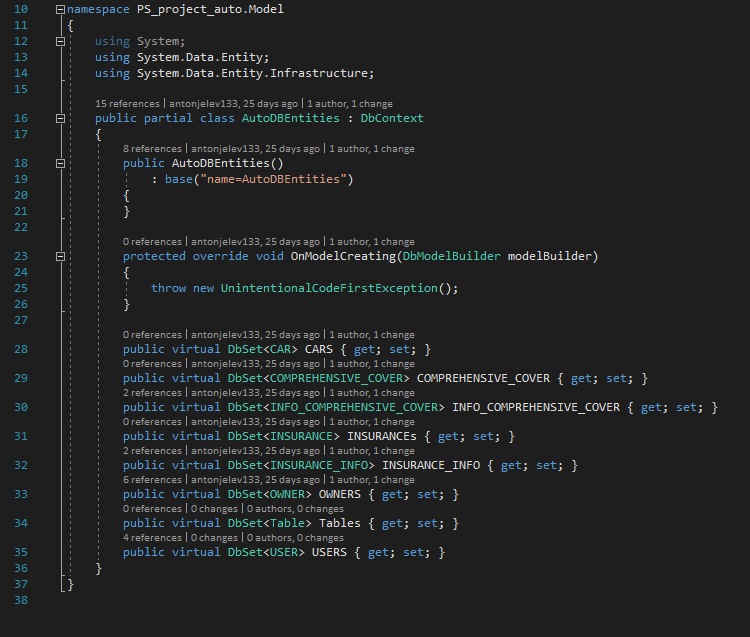
При регистрацията е необходимо да се въведат потребителско име, което да е между 9 и 20 символа и дадена парола между 6 и 20 символа.След като всички условия са изпълнени, вкарва данните за дадения юсър в базата. Ако не са, ще излезе съобщение за грешка!  
При логин формата има също валидация на адрес и име,които трябва да са между 1 и 20 символа. Ако не са, ще излезе съобщение за грешка!  
Използват се 7 модела – USER, OWNER, CAR ,INSURANCE, INSURANCE\_INFO, COMPRAHENSIVE\_COVER , INFO\_COMPRAHENSIVE\_COVER които сме обяснили по-подробно в реализацията.ЕР диаграмата ни е следната:  


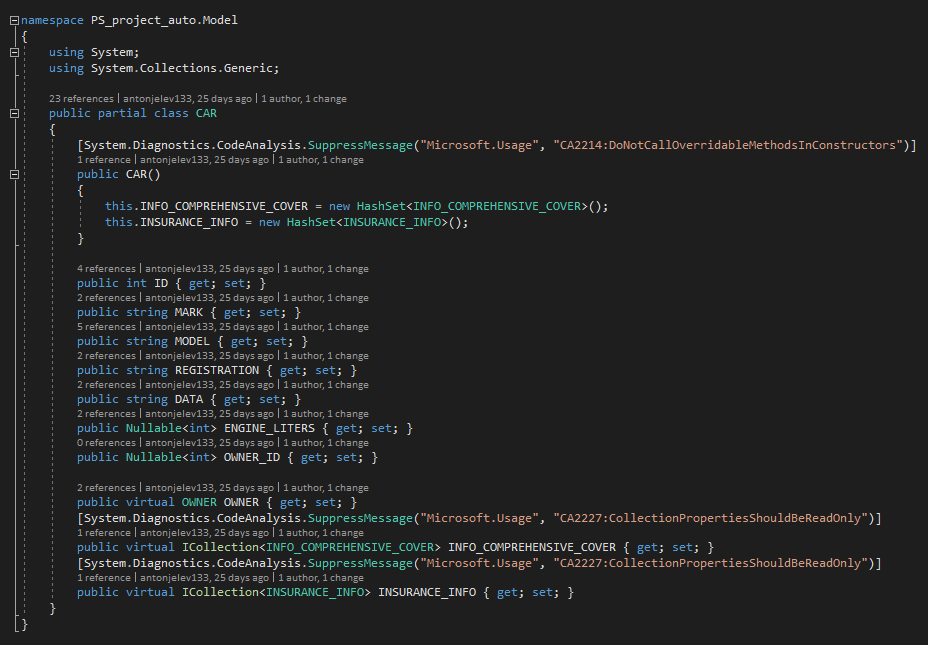
## Как ще бъдат достъпени функционалностите от потребителя?

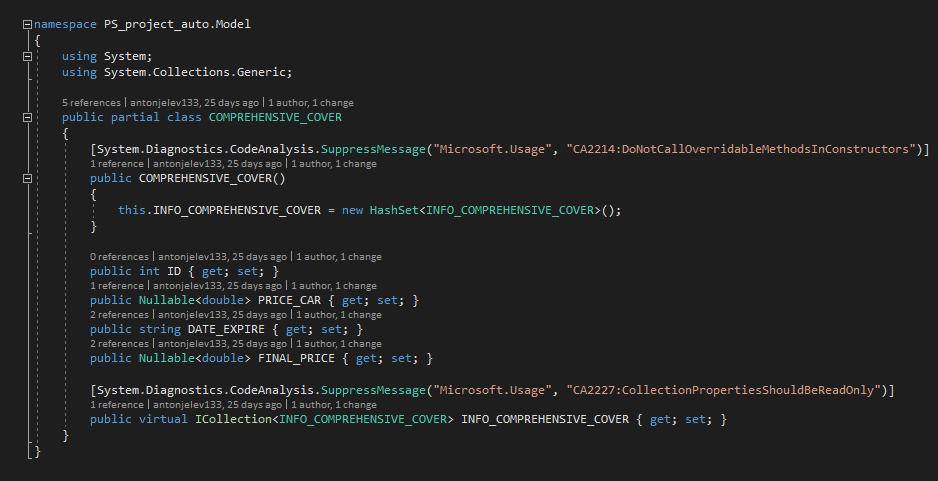
Ако даденият човек все още не разполага с акаунт, ще му бъде придоставена регистрационна форма чрез която да се регистрира.След успешна регистрация може да премине към логин формата и да започне тест.

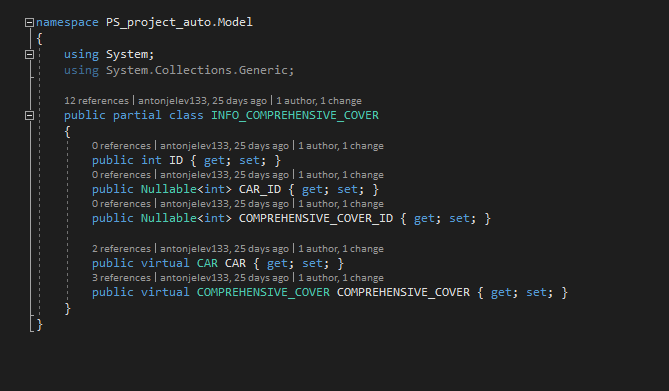
# 4.Реализация

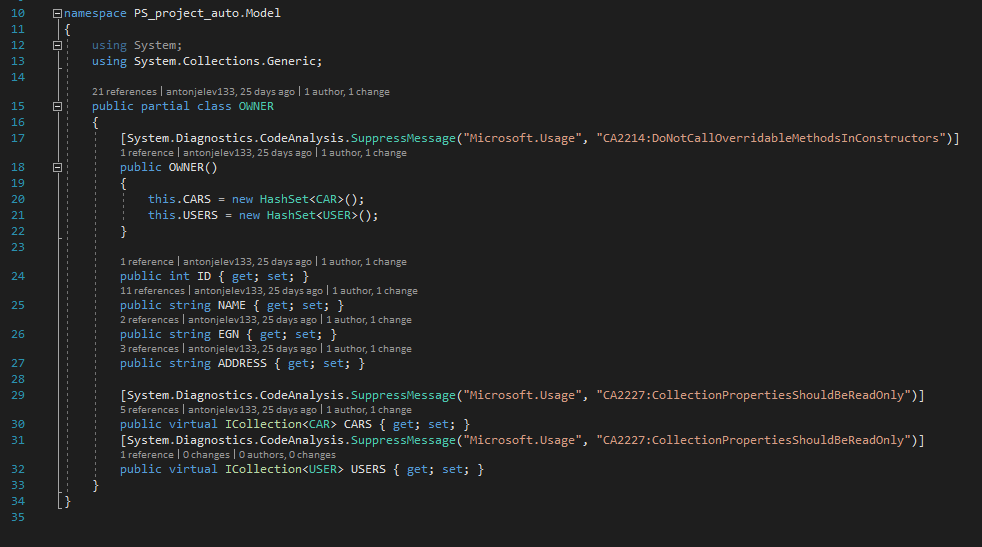
Използван е трислоен подход което означава, че приложението ще има трислоен структура.  
Класовете - модели, които са използвани от мен са USER,CAR,OWNER,COMPRAHENSIVE\_COVER, INFO\_COMPRAHENSIVE\_COVER  
USER притежава данните за дадения юсър,както и връзка със съотвения собственик (OWNER) , Таблицата CAR има връзка със OWNER (много към 1).Таблицата INFO\_COMPRAHENSIVE\_COVER съдържа връзка към CAR и COMPRAHENSIVE\_COVER.Използвайки подхода за създаване на базата с данни първа генерираме модели ,които притежават същите имана. Генериран е и класа AutoDBEntities , който имплементира DbContext.  
Класове в пакетът model:

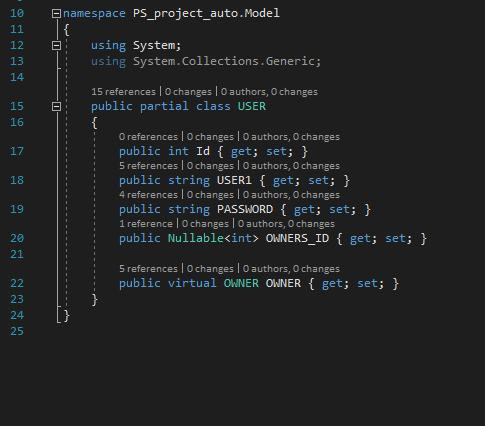




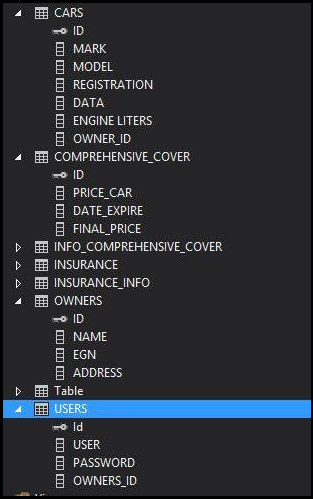








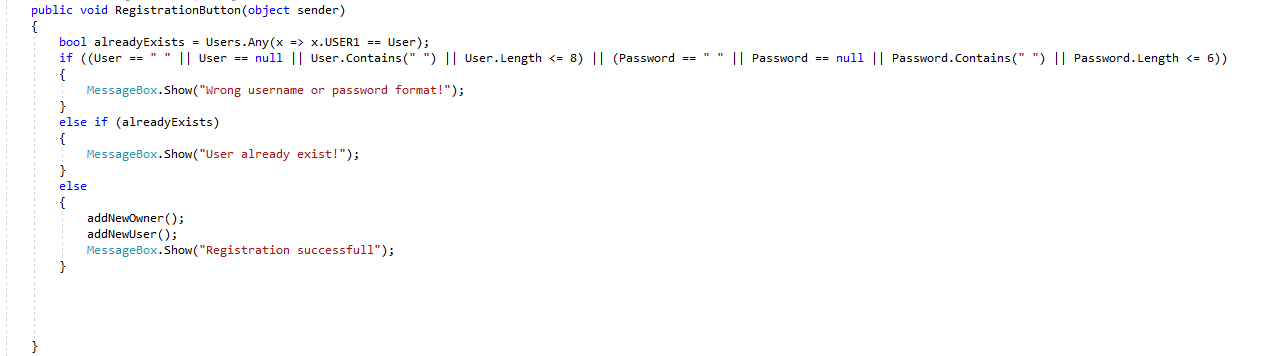
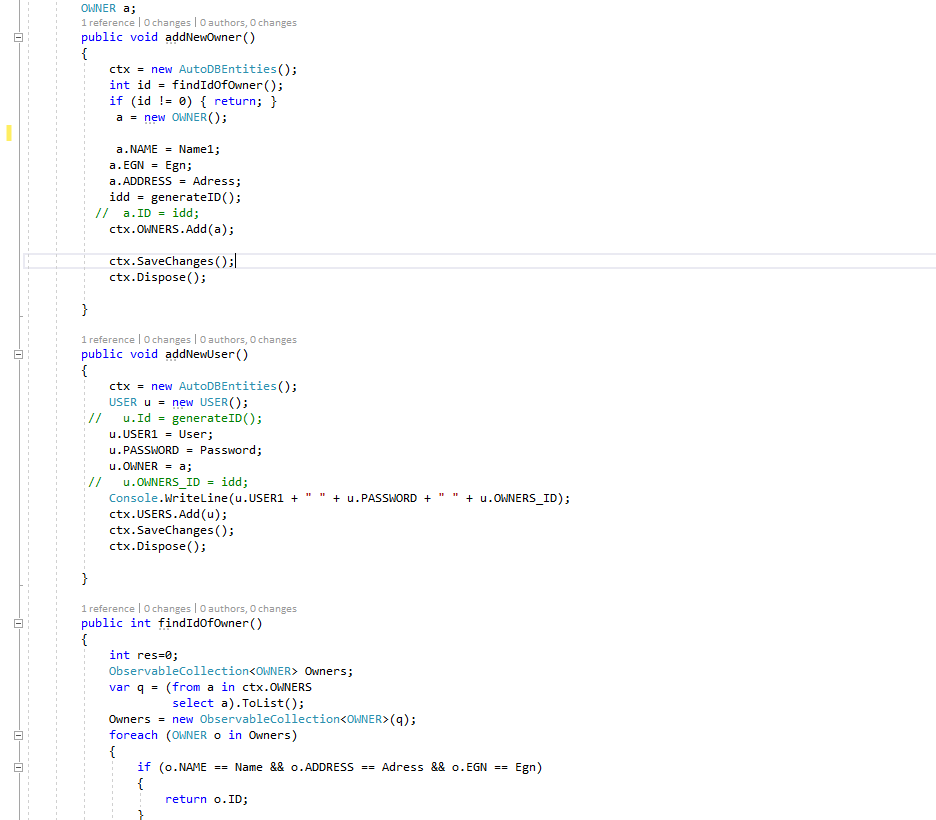
Базата данни ни е следната:



В моята реализация базата с данни е локална и се съхранява в AutoDB.mdf

Имплементираме ICommand интерфейса ,с чиято помощ ше изпълняваме еvent-ти



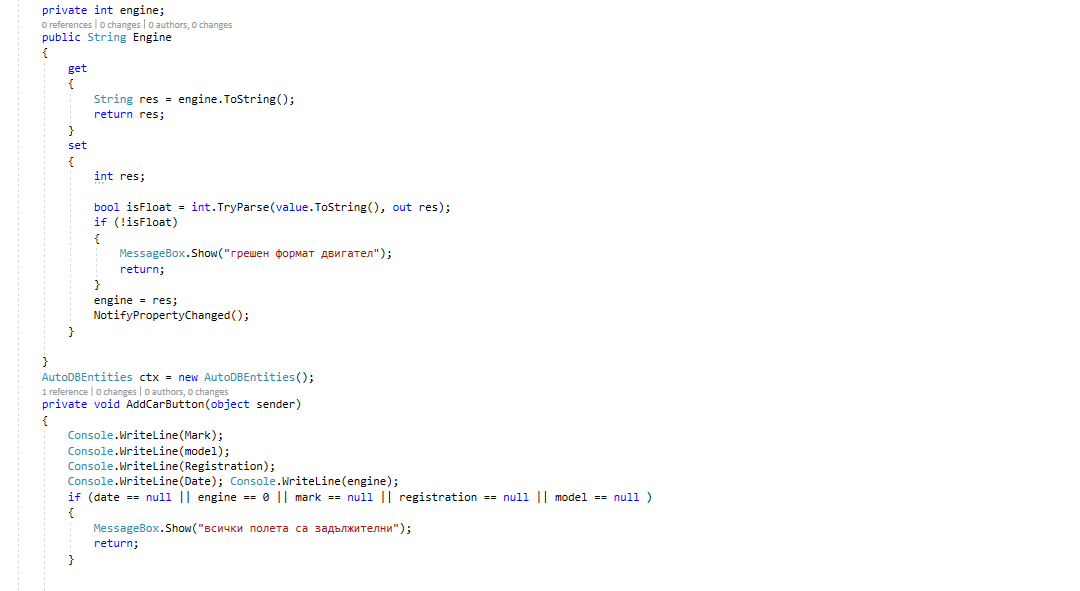
Приложението ни започва от MainWindow.xaml чрез който извикваме страницата Registration.xaml .Логиката и е дефинирана в класа RegistrationViewModel .След като се направи необходимата валидация на данните, се прави проверка дали даденият юсър съществува в базата.  
  
  
След това продължаваме към методите addNewOwner() и , addNewUser където обектите биват добавяни като редове в базата с данни.  
  


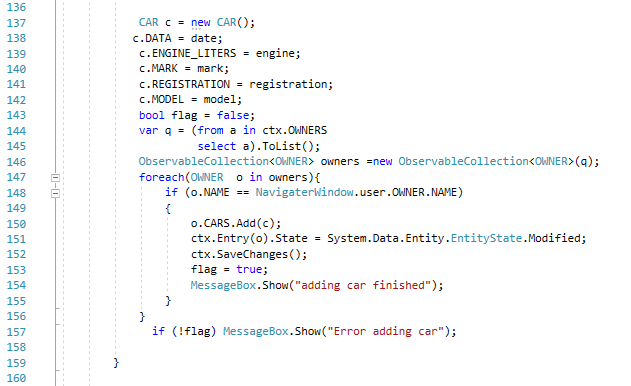
След като успешно преминем тази стъпка, и влезнем със създаденото ни име и парола от главния прозорец пред нас се отваря прозорецът NavigatorWindow.xaml .

Преминавайки към неговата имплементация на DataContext стигаме до NavigatorViewModel. В него наблюдаваме бизнес логиката ,чрез която преминаваме на в страниците на прозореца – а именно – NavigatorPage,Comprahensive.page,InsurancePage,CarPage. Имплементацията за навигация в NavigatorViewModel:



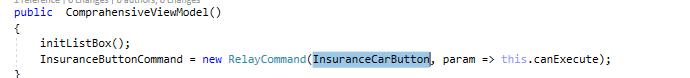
За да направим автокаско на даден автомобил е редно преди това да сме добавили такъв следователно преминаваме към страницата CarPage.xaml чиято имплементация на DataContext намираме в CarViewModel.cs. След попълване на формата и натискане на бутона за добавяне идва ред на валидацията:

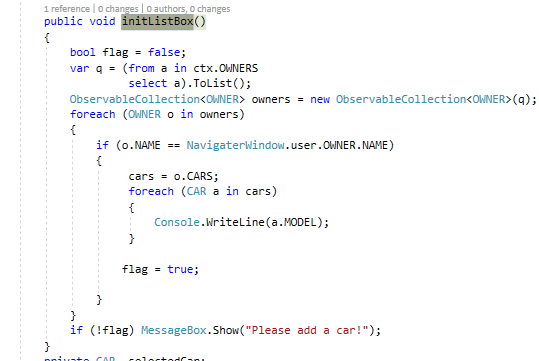
  
Продължаваме с добавяне на автомобила в базата с данни:

  
  
  
Следващата стъпка при успешно добавяне на автомобил би била да добавим автокаско ,застраховка или да изискаме фактура. Приносът ми към проекта продължава с добавянето на автокаско, затова от NavigatorPage избираме автокаско.

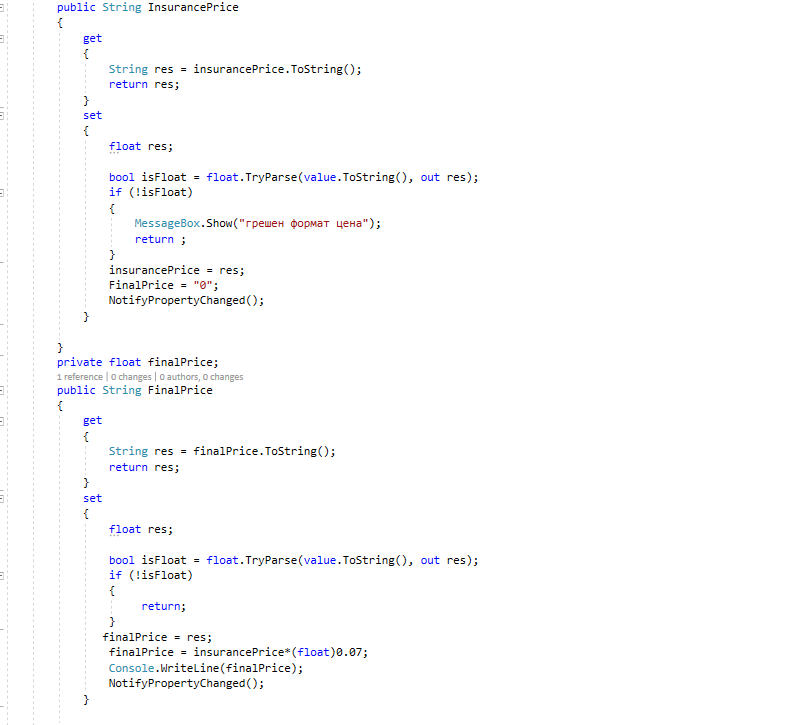
Така стигаме до ComprahensivePage.xaml . От там в имплементацията на нейният DataContext – ComprahensiveViewModel.cs

Първо попълваме ListView с автомобилите на собственика

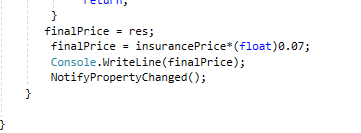




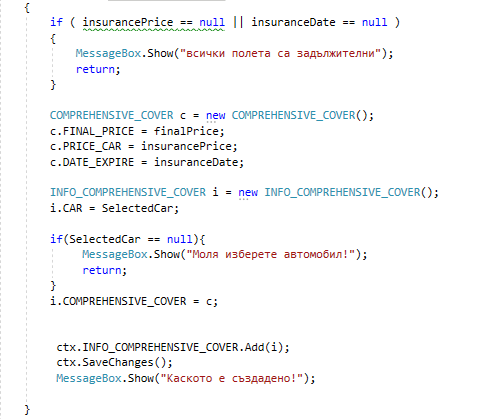
След попълването на формата натискаме бутона добави ,при което попълнените данни биват валидирани:



След това от полето на формата крайна цена последстном следната формула ще излезне крайната цена:



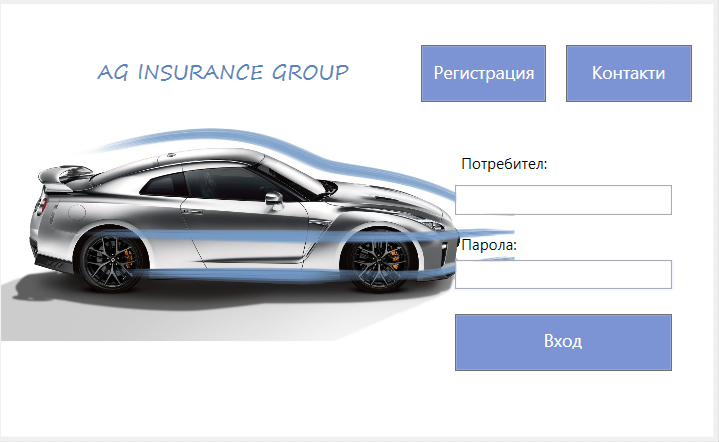
Продължаваме с командата за бутон “застраховай”



При успех автокаското е добавено в базата с данни и ние може да се върнем къмто навигационната страница къде да извършим различна операция.

# 5. Потребителскo ръководство

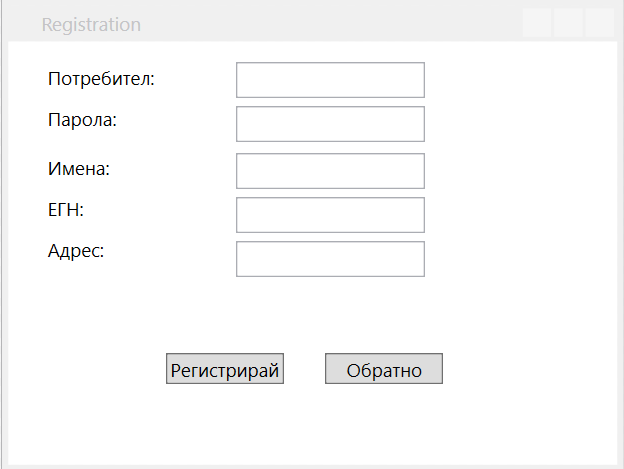
Нека първо започнем с логин формата, която е представена на фигура 13.Ако даденият потребител не притежава акаунт ще е необходимо първо да се регистрира.



Фиг 13.

На фиг.14 е представена регистрационната форма, през която даден юсър ще може да си създаде акаунт.Името трябва да са между 7 и 20 символаа , потребителят между 7 и 20,паролата – 8-20,eгн-то 10 ,а адресът между 7 и 20 !

*Фиг. 15 home форма*

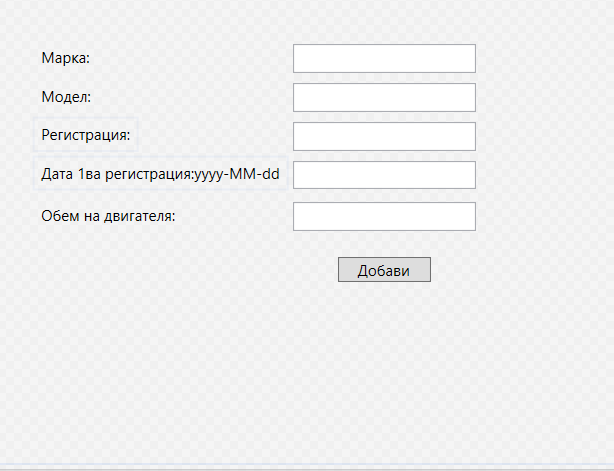
Фиг 14.  
  
След като успешно влезете в акаунта си, ще ви излезе прозорецът, представен на фиг.15.Ще имате право да изберете дали искате да направите добавяне на кола,застраховка, автокаско или извличане на данни и запис в файл(проверка в

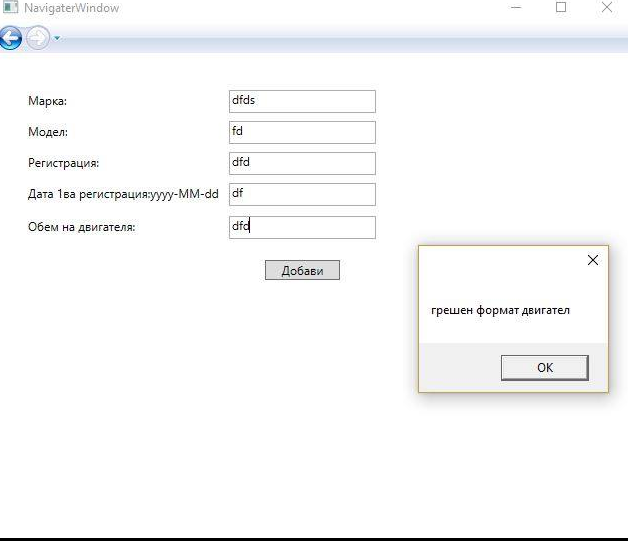
системата).



Фиг 15

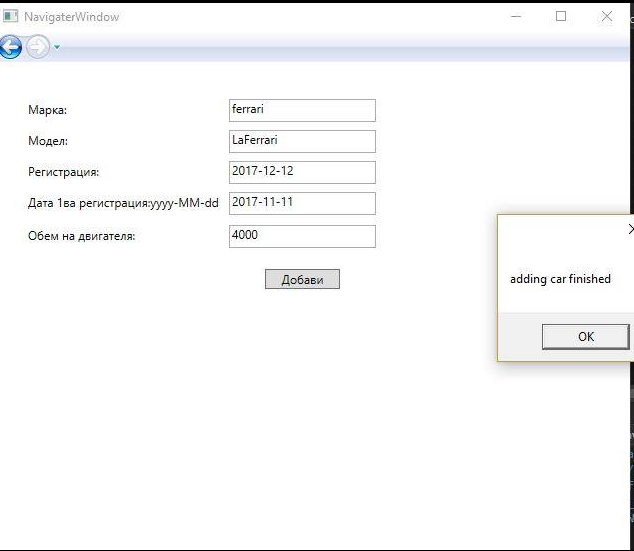
Нека преминем към добавянето на автомобил:



Следва попълване на зададената форма със зададените от нас данни. При некоректно попълнени полета или не попълнена форма бихме получили следното съобщение:  


*Фиг. 17 тест форма без време*

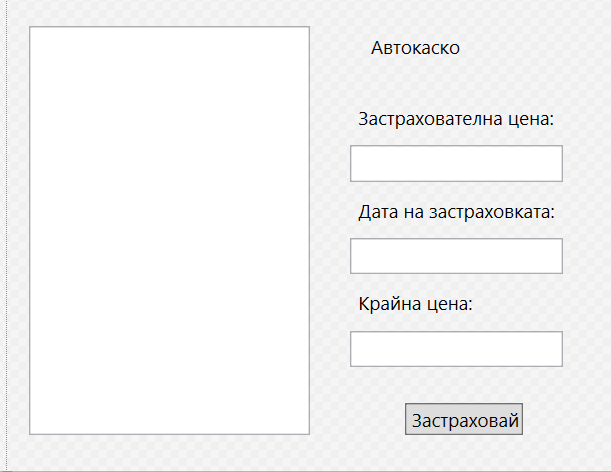
При успех съобщението е:



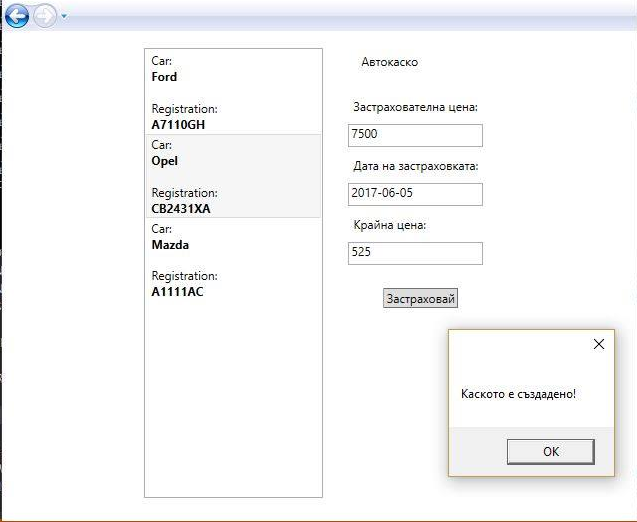
*Фиг. 18 тест форма с време*

Нека да преминем към добавянето на автокаско.При избор автокаско от NavigatorPage би се отворила следната страница:

*Фиг. 19 резултат*



При успешна създаване на автокаско виждаме следното съобщение:



# 6. Заключение

Настоящата разработка предоставя една система за застраховане и каско базирана на WPF . Използвана е методика за приложения в застрахователния сектор. Потребителите могат да използват системата за застраховане на автомобили, при които отпада необходимостта от посещаване на офис. При поискване, експедитивно се осигурява извлечение от наличните данни за автокаско и застраховка на даден автомобил по електронен път. Разгледаната система за онлайн застраховане се използва все по-често в практиката и ще намира все по-широко приложение в съвременната застрахователна система.

# 7.Литература

<https://stackoverflow.com/>

<https://docs.microsoft.com/en-us/dotnet/csharp/csharp>

<https://www.dzi.bg/bg/%D0%B7%D0%B0-%D1%84%D0%B8%D1%80%D0%BC%D0%B8/%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%BE%D0%B1%D0%B8%D0%BB%D0%BD%D0%BE-%D0%B7%D0%B0%D1%81%D1%82%D1%80%D0%B0%D1%85%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5/%D0%BA%D0%B0%D1%81%D0%BA%D0%BE>

<https://24ins.bg/>

<http://www.broko.bg/>

# 8. Приложение

using System;

using System.Windows.Input;

namespace PS\_project\_auto.ModelView

{

public class RelayCommand:ICommand

{

private Action<object> execute;

private Predicate<object> canExecute;

private event EventHandler CanExecuteChangedInternal;

public RelayCommand(Action<object> execute)

: this(execute, DefaultCanExecute)

{

}

public RelayCommand(Action<object> execute, Predicate<object> canExecute)

{

if (execute == null)

{

throw new ArgumentNullException("execute");

}

if (canExecute == null)

{

throw new ArgumentNullException("canExecute");

}

this.execute = execute;

this.canExecute = canExecute;

}

public event EventHandler CanExecuteChanged

{

add

{

CommandManager.RequerySuggested += value;

this.CanExecuteChangedInternal += value;

}

remove

{

CommandManager.RequerySuggested -= value;

this.CanExecuteChangedInternal -= value;

}

}

public bool CanExecute(object parameter)

{

return this.canExecute != null && this.canExecute(parameter);

}

public void Execute(object parameter)

{

this.execute(parameter);

}

public void OnCanExecuteChanged()

{

EventHandler handler = this.CanExecuteChangedInternal;

if (handler != null)

{

//DispatcherHelper.BeginInvokeOnUIThread(() => handler.Invoke(this, EventArgs.Empty));

handler.Invoke(this, EventArgs.Empty);

}

}

public void Destroy()

{

this.canExecute = \_ => false;

this.execute = \_ => { return; };

}

private static bool DefaultCanExecute(object parameter)

{

return true;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows.Input;

using PS\_project\_auto.Model;

using PS\_project\_auto.View;

using System.Windows.Navigation;

using System.Windows.Controls;

using System.Windows;

using System.Collections.ObjectModel;

namespace PS\_project\_auto.ModelView

{

public class RegistrationViewModel: Window,INotifyPropertyChanged

{

private String user;

public String User

{

get

{

return user;

}

set

{

user = value;

NotifyPropertyChanged();

}

}

private String password;

private int idd;

public String Password

{

get

{

return password;

}

set

{

password = value;

NotifyPropertyChanged();

}

}

private String name;

public String Name1

{

get

{

return name;

}

set

{

name = value;

NotifyPropertyChanged();

}

}

private String egn;

public String Egn

{

get

{

return egn;

}

set

{

egn = value;

NotifyPropertyChanged();

}

}

private String adress;

public String Adress

{

get

{

return adress;

}

set

{

adress = value;

NotifyPropertyChanged();

}

}

public int generateID()

{

Random rnd = new Random();

int r= rnd.Next(1,111111111);

return r;

}

AutoDBEntities ctx = new AutoDBEntities();

public void FillUsers()

{

var q = (from a in ctx.USERS

select a).ToList();

this.Users = new ObservableCollection<USER>(q);

foreach (object o in Users)

{ Console.WriteLine(o); }

ctx.Dispose();

}

ObservableCollection<USER> Users;

public RegistrationViewModel()

{

FillUsers();

BackButtonCommand = new RelayCommand(BackButton, param => this.canExecute);

RegistrationButtonCommand = new RelayCommand(RegistrationButton, param => this.canExecute);

}

public void RegistrationButton(object sender)

{

bool alreadyExists = Users.Any(x => x.USER1 == User);

if ((User == " " || User == null || User.Contains(" ") || User.Length <= 8) || (Password == " " || Password == null || Password.Contains(" ") || Password.Length <= 6))

{

MessageBox.Show("Wrong username or password format!");

}

else if (alreadyExists)

{

MessageBox.Show("User already exist!");

}

else

{

addNewOwner();

addNewUser();

MessageBox.Show("Registration successfull");

}

}

OWNER a;

public void addNewOwner()

{

ctx = new AutoDBEntities();

int id = findIdOfOwner();

if (id != 0) { return; }

a = new OWNER();

a.NAME = Name1;

a.EGN = Egn;

a.ADDRESS = Adress;

idd = generateID();

// a.ID = idd;

ctx.OWNERS.Add(a);

ctx.SaveChanges();

ctx.Dispose();

}

public void addNewUser()

{

ctx = new AutoDBEntities();

USER u = new USER();

// u.Id = generateID();

u.USER1 = User;

u.PASSWORD = Password;

u.OWNER = a;

// u.OWNERS\_ID = idd;

Console.WriteLine(u.USER1 + " " + u.PASSWORD + " " + u.OWNERS\_ID);

ctx.USERS.Add(u);

ctx.SaveChanges();

ctx.Dispose();

}

public int findIdOfOwner()

{

int res=0;

ObservableCollection<OWNER> Owners;

var q = (from a in ctx.OWNERS

select a).ToList();

Owners = new ObservableCollection<OWNER>(q);

foreach (OWNER o in Owners)

{

if (o.NAME == Name && o.ADDRESS == Adress && o.EGN == Egn)

{

return o.ID;

}

}

return res;

}

public void BackButton(object sender)

{

if (MainWindowViewModel.rr != null)

{

MainWindowViewModel.rr.Close();

MainWindowViewModel.rr = null;

}

MainWindowViewModel.ww = new MainWindow();

MainWindowViewModel.ww.Show();

}

private ICommand registrationButtonCommand;

public ICommand RegistrationButtonCommand

{

get

{

return registrationButtonCommand;

}

set

{

registrationButtonCommand = value;

}

}

private ICommand backButtonCommand;

public ICommand BackButtonCommand

{

get

{

return backButtonCommand;

}

set

{

backButtonCommand = value;

}

}

private bool canExecute = true;

public bool CanExecute

{

get { return this.canExecute; }

set

{

if (this.canExecute == value) { return; }

this.canExecute = value;

}

}

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged([CallerMemberName] String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

}

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows.Input;

using PS\_project\_auto.Model;

using PS\_project\_auto.View;

using System.Windows.Navigation;

using System.Windows.Controls;

using System.Windows;

using System.Collections.ObjectModel;

using System.Data.SqlClient;

using System;

namespace PS\_project\_auto.ModelView

{

public class NavigationViewModel : Window

{

public NavigationViewModel()

{

AddCarButtonCommand = new RelayCommand(AddCarButton, param => this.canExecute);

CheckButtonCommand = new RelayCommand(CheckButtonButton, param => this.canExecute);

InsuranceButtonCommand = new RelayCommand(InsuranceButtonButton, param => this.canExecute);

ComprehensiveCoverButtonCommand = new RelayCommand(ComprehensiveCoverButton, param => this.canExecute);

}

private void AddCarButton(object sender)

{

NavigationPage.page.NavigationService.Navigate(new Uri("View/CarPage.xaml", UriKind.Relative));

}

private void CheckButtonButton(object sender)

{

NavigationPage.page.NavigationService.Navigate(new Uri("View/PrintPage.xaml", UriKind.Relative));

}

private void InsuranceButtonButton(object sender)

{

NavigationPage.page.NavigationService.Navigate(new Uri("View/InsurancePage.xaml", UriKind.Relative));

}

private void ComprehensiveCoverButton(object sender)

{

NavigationPage.page.NavigationService.Navigate(new Uri("View/ComprahensivePage.xaml", UriKind.Relative));

}

private ICommand addCarButtonCommand;

public ICommand AddCarButtonCommand

{

get

{

return addCarButtonCommand;

}

set

{

addCarButtonCommand = value;

}

}

private ICommand checkButtonCommand;

public ICommand CheckButtonCommand

{

get

{

return checkButtonCommand;

}

set

{

checkButtonCommand = value;

}

}

private ICommand insuranceButtonCommand;

public ICommand InsuranceButtonCommand

{

get

{

return insuranceButtonCommand;

}

set

{

insuranceButtonCommand = value;

}

}

private ICommand comprehensiveCoverButtonCommand;

public ICommand ComprehensiveCoverButtonCommand

{

get

{

return comprehensiveCoverButtonCommand;

}

set

{

comprehensiveCoverButtonCommand = value;

}

}

private bool canExecute = true;

public bool CanExecute

{

get { return this.canExecute; }

set

{

if (this.canExecute == value) { return; }

this.canExecute = value;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows.Input;

using PS\_project\_auto.Model;

using PS\_project\_auto.View;

using System.Windows.Navigation;

using System.Windows.Controls;

using System.Windows;

using System.Collections.ObjectModel;

using System.Data.SqlClient;

using System.Globalization;

namespace PS\_project\_auto.ModelView

{

public class ComprahensiveViewModel : Window, INotifyPropertyChanged

{

AutoDBEntities ctx = new AutoDBEntities();

public ComprahensiveViewModel()

{

initListBox();

InsuranceButtonCommand = new RelayCommand(InsuranceCarButton, param => this.canExecute);

}

private void InsuranceCarButton(object sender)

{

if ( insurancePrice == null || insuranceDate == null )

{

MessageBox.Show("всички полета са задължителни");

return;

}

COMPREHENSIVE\_COVER c = new COMPREHENSIVE\_COVER();

c.FINAL\_PRICE = finalPrice;

c.PRICE\_CAR = insurancePrice;

c.DATE\_EXPIRE = insuranceDate;

INFO\_COMPREHENSIVE\_COVER i = new INFO\_COMPREHENSIVE\_COVER();

i.CAR = SelectedCar;

if(SelectedCar == null){

MessageBox.Show("Моля изберете автомобил!");

return;

}

i.COMPREHENSIVE\_COVER = c;

ctx.INFO\_COMPREHENSIVE\_COVER.Add(i);

ctx.SaveChanges();

MessageBox.Show("Каското е създадено!");

}

private ICommand insuranceButtonCommand;

public ICommand InsuranceButtonCommand

{

get

{

return insuranceButtonCommand;

}

set

{

insuranceButtonCommand = value;

}

}

private float insurancePrice;

public String InsurancePrice

{

get

{

String res = insurancePrice.ToString();

return res;

}

set

{

float res;

bool isFloat = float.TryParse(value.ToString(), out res);

if (!isFloat)

{

MessageBox.Show("грешен формат цена");

return ;

}

insurancePrice = res;

FinalPrice = "0";

NotifyPropertyChanged();

}

}

private float finalPrice;

public String FinalPrice

{

get

{

String res = finalPrice.ToString();

return res;

}

set

{

float res;

bool isFloat = float.TryParse(value.ToString(), out res);

if (!isFloat)

{

return;

}

finalPrice = res;

finalPrice = insurancePrice\*(float)0.07;

Console.WriteLine(finalPrice);

NotifyPropertyChanged();

}

}

private String insuranceDate;

public String InsuranceDate

{

get

{

return insuranceDate;

}

set

{

insuranceDate = value;

NotifyPropertyChanged();

}

}

private ICollection<CAR> \_cars;

public ICollection<CAR> cars

{

get { return this.\_cars; }

set

{

this.\_cars = value;

NotifyPropertyChanged();

}

}

public void initListBox()

{

bool flag = false;

var q = (from a in ctx.OWNERS

select a).ToList();

ObservableCollection<OWNER> owners = new ObservableCollection<OWNER>(q);

foreach (OWNER o in owners)

{

if (o.NAME == NavigaterWindow.user.OWNER.NAME)

{

cars = o.CARS;

foreach (CAR a in cars)

{

Console.WriteLine(a.MODEL);

}

flag = true;

}

}

if (!flag) MessageBox.Show("Please add a car!");

}

private CAR \_selectedCar;

public CAR SelectedCar

{

get

{

return this.\_selectedCar;

}

set

{

this.\_selectedCar = value;

NotifyPropertyChanged();

}

}

private bool canExecute = true;

public bool CanExecute

{

get { return this.canExecute; }

set

{

if (this.canExecute == value) { return; }

this.canExecute = value;

}

}

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged([CallerMemberName] String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows.Input;

using PS\_project\_auto.Model;

using PS\_project\_auto.View;

using System.Windows.Navigation;

using System.Windows.Controls;

using System.Windows;

using System.Collections.ObjectModel;

using System.Data.SqlClient;

using System.Globalization;

namespace PS\_project\_auto.ModelView

{

class CarViewModel : Window, INotifyPropertyChanged

{

public CarViewModel()

{

Console.WriteLine("days : " + compare());

AddCarButtonCommand = new RelayCommand(AddCarButton, param => this.canExecute);

}

double compare()

{

DateTime dtFromDate = DateTime.ParseExact("11/06/2017", "dd/MM/yyyy",

CultureInfo.InvariantCulture);

DateTime dtToDate = DateTime.ParseExact("11/07/2017", "dd/MM/yyyy",

CultureInfo.InvariantCulture);

TimeSpan difference = dtFromDate - dtToDate;

double days = difference.TotalDays;

return days;

}

private String mark;

public String Mark

{

get

{

return mark;

}

set

{

mark = value;

NotifyPropertyChanged();

}

}

private String model;

public String Model

{

get

{

return model;

}

set

{

model = value;

NotifyPropertyChanged();

}

}

private String registration;

public String Registration

{

get

{

return registration;

}

set

{

registration = value;

NotifyPropertyChanged();

}

}

private String date;

public String Date

{

get

{

return date;

}

set

{

date = value;

NotifyPropertyChanged();

}

}

private int engine;

public String Engine

{

get

{

String res = engine.ToString();

return res;

}

set

{

int res;

bool isFloat = int.TryParse(value.ToString(), out res);

if (!isFloat)

{

MessageBox.Show("грешен формат двигател");

return;

}

engine = res;

NotifyPropertyChanged();

}

}

AutoDBEntities ctx = new AutoDBEntities();

private void AddCarButton(object sender)

{

Console.WriteLine(Mark);

Console.WriteLine(model);

Console.WriteLine(Registration);

Console.WriteLine(Date); Console.WriteLine(engine);

if (date == null || engine == 0 || mark == null || registration == null || model == null )

{

MessageBox.Show("всички полета са задължителни");

return;

}

CAR c = new CAR();

c.DATA = date;

c.ENGINE\_LITERS = engine;

c.MARK = mark;

c.REGISTRATION = registration;

c.MODEL = model;

bool flag = false;

var q = (from a in ctx.OWNERS

select a).ToList();

ObservableCollection<OWNER> owners =new ObservableCollection<OWNER>(q);

foreach(OWNER o in owners){

if (o.NAME == NavigaterWindow.user.OWNER.NAME)

{

o.CARS.Add(c);

ctx.Entry(o).State = System.Data.Entity.EntityState.Modified;

ctx.SaveChanges();

flag = true;

MessageBox.Show("adding car finished");

}

}

if (!flag) MessageBox.Show("Error adding car");

}

private ICommand addCarButtonCommand;

public ICommand AddCarButtonCommand

{

get

{

return addCarButtonCommand;

}

set

{

addCarButtonCommand = value;

}

}

private bool canExecute = true;

public bool CanExecute

{

get { return this.canExecute; }

set

{

if (this.canExecute == value) { return; }

this.canExecute = value;

}

}

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged([CallerMemberName] String propName = "")

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propName));

}

}

}

}