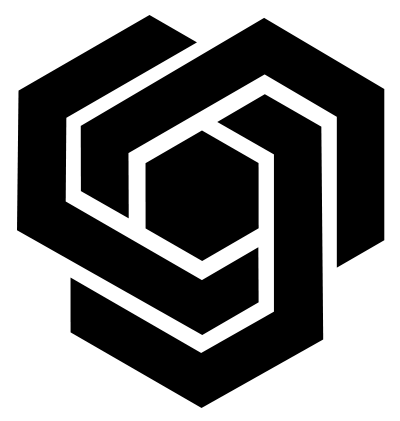
**Т Е Х Н И Ч Е С К И У Н И В Е Р С И Т Е Т - С О Ф И Я**



Направление „ Компютърно и софтуерно инженерство“

**Курсов проект**

**по Програмни среди**

Разработил: Антон Желев Одобрил: гл. ас. д-р А. Ташева

Катедра: ФКСТ

Фак. № 121214021

Група: 44

**15.06.2017**

**Гр. София**

Съдържание

[1.Увод 3](#_Toc484603291)

[2. Анализ на съществуващи разработки 3](#_Toc484603292)

[„SDI“ застраховане“ 4](#_Toc484603293)

[Положителни страни: 4](#_Toc484603294)

[Отрицателни страни: 4](#_Toc484603295)

[„ Застраховане „Broko“ 4](#_Toc484603296)

[Положителни страни: 5](#_Toc484603297)

[Отрицателни страни: 5](#_Toc484603298)

[„24инс““ 6](#_Toc484603299)

[Положителни страни: 6](#_Toc484603300)

[Отрицателни страни: 6](#_Toc484603301)

[3.Проектиране 6](#_Toc484603302)

[Какви данни ще се използват? 7](#_Toc484603303)

[Как ще бъдат достъпени функционалностите от потребителя? 7](#_Toc484603304)

[4.Реализация 8](#_Toc484603305)

[5. Потребителскo ръководство 14](#_Toc484603306)

[6. Заключение 18](#_Toc484603307)

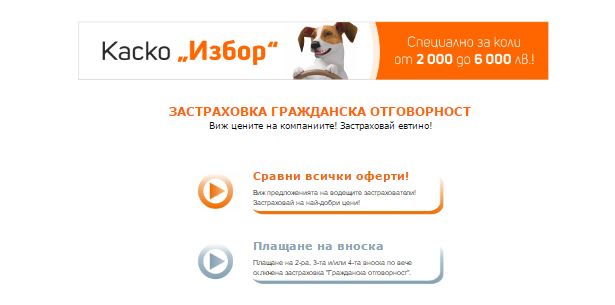
[7.Литература 19](#_Toc484603308)

[8. Приложение 19](#_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

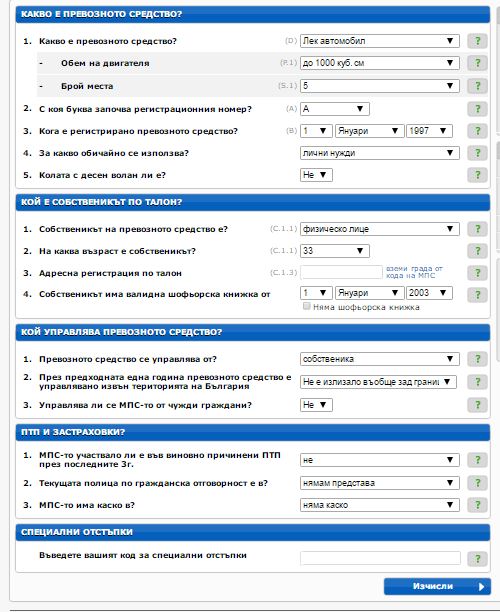
### Положителни страни:

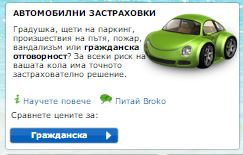
* Семпъл и лесен за навигация интерфейс;
* Приложението позволява повторно попълване на теста.

### Отрицателни страни:

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

## Застраховане „Broko”

****

****

### 

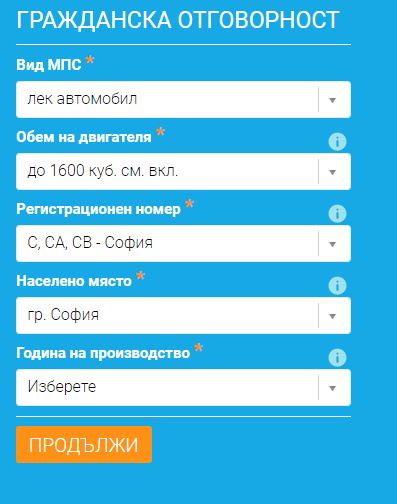
### Положителни страни:

* 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, което го превръща в лесен и достъпен начин за обогатяване на
* 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;

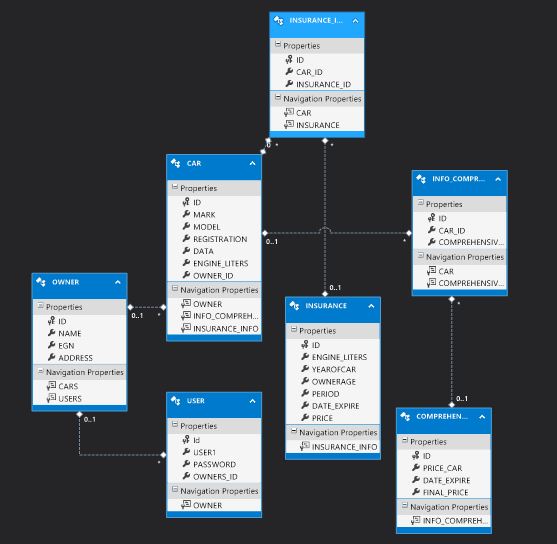
2. Намалява се ангажираността на застрахователните брокери в застрахователния процес;

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?

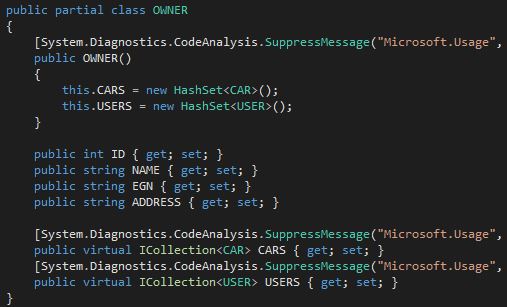
При регистрацията е необходимо да се въведат потребителско име, което да е между 9 и 20 символа и дадена парола между 6 и 20 символа.След като всички условия са изпълнени, вкарва данните за дадения юсър в базата. Ако не са, ще излезе съобщение за грешка!  
При логин формата има също валидация на адрес 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:  


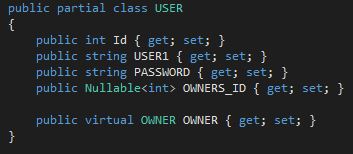
## Как ще бъдат достъпени функционалностите от потребителя?

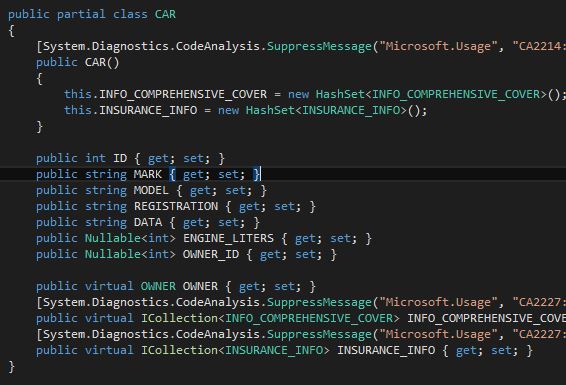
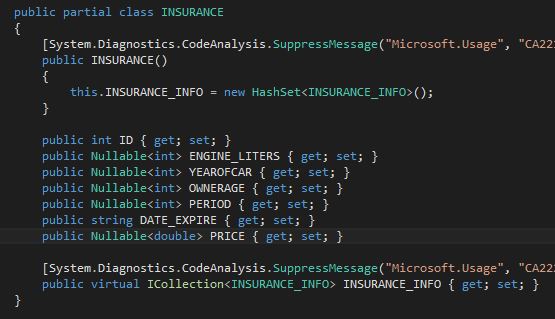
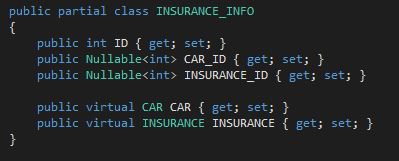
Ако даденият човек все още не разполага с акаунт, ще му бъде предоставена регистрационна форма чрез която да се регистрира.След успешна регистрация може да премине към логин формата , след което да добави автомобил и да избере вида на застраховката.

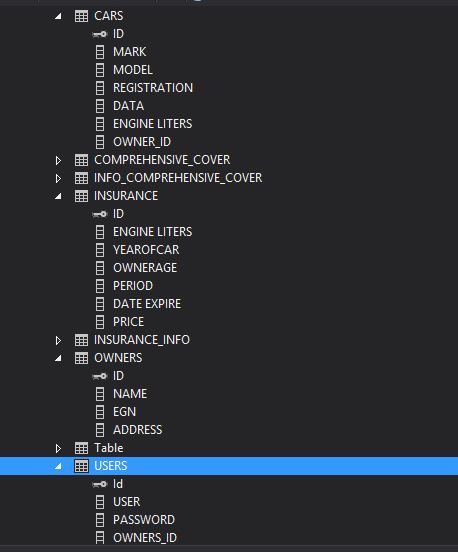
# 4.Реализация

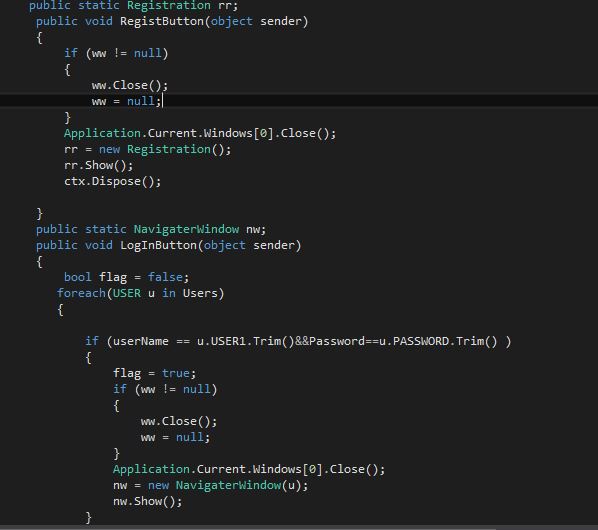
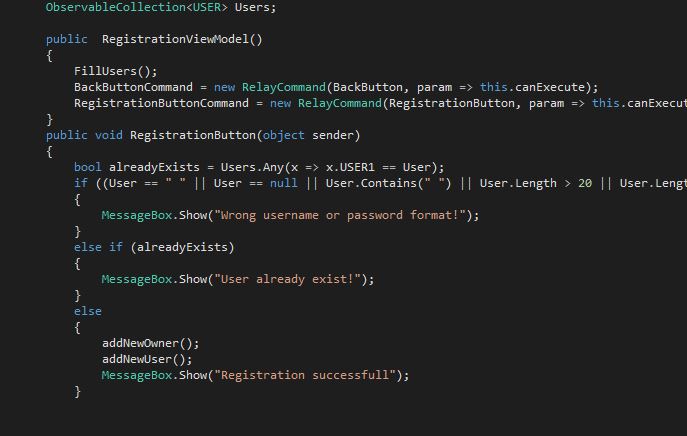
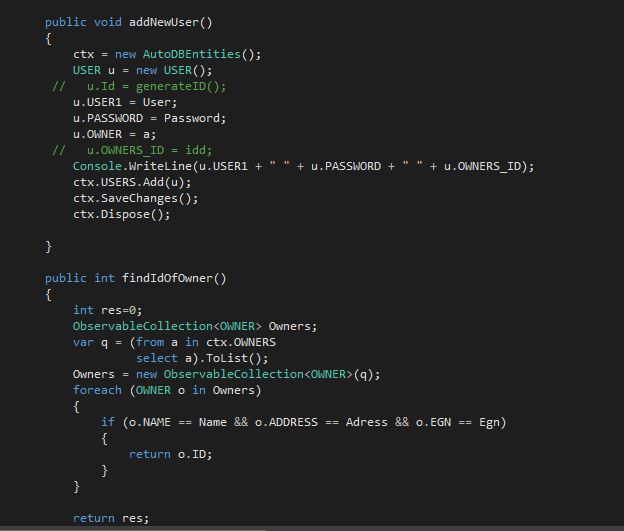
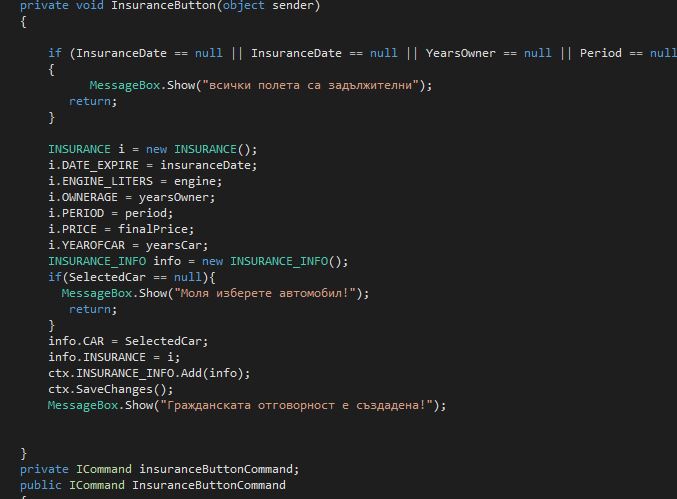
Използван е трислоен подход което означава, че приложението ще има трислойна структура – ViewModel, в който се държи имплементацията , View слой, който представя визуалните страници на приложението и Model слой включва модел на даннитe.

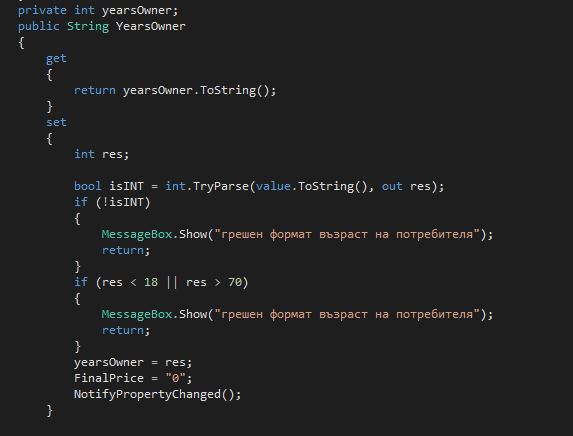
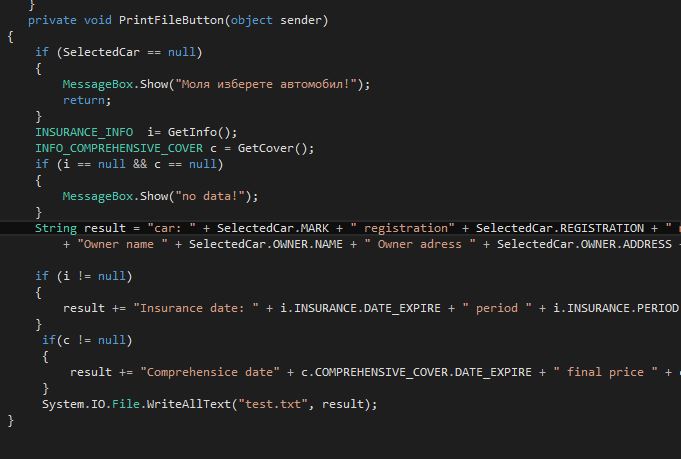
  
Класовете - модели, които са използвани са OWNER,USER,CAR,INSURANCE,INSURANCE\_INFO.  
USER притежава данните за дадения юзър,OWNER съдържа личните данните за юзъра .  
CAR съдържа данните за автомобилите на конкретния USER, INSURANCE съдържа данни за застраховка и INSURANCE\_INFO съдържа застраховката за конкретна кола





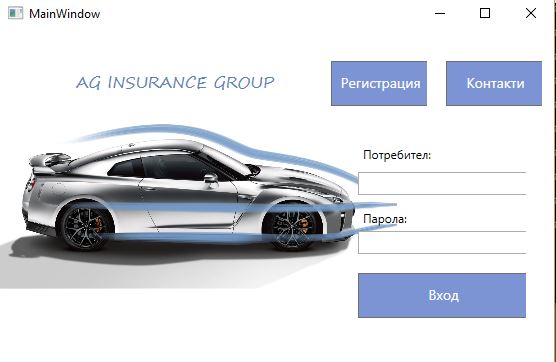
Базата данни ни е следната:  
  
  


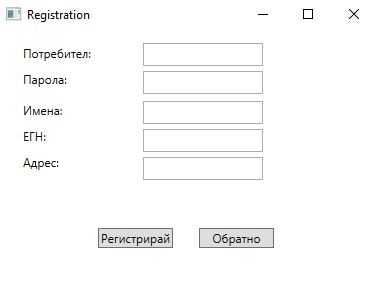
Приложението ни започва от MainWindowViewModel, който наследява Window,а сорса му е зададен към MainWindow.xaml за да ни пренасочва към началната страница.Нека преминем към Registration страницата.След като се направи необходимата валидация на данните, се прави проверка дали даденият юсър съществува в базата.  
  
  
При успешна регистрация се връща към MainWindow. Въвеждат се логин данните и се проверява юзера за съществуването му. При успешен вход се преминава към NavigaterWindow.xaml   
  
  
  
В NavigaterWindow може да изберем бутона за застраховка гражданска отговорност или проверка в системата. При избор на ГО се преминава към InsurancePage , където се избира автомобил и се въвеждат данни за него.  
  
  
За добавяне на ново застраховане е сложена валидация на всички полета.

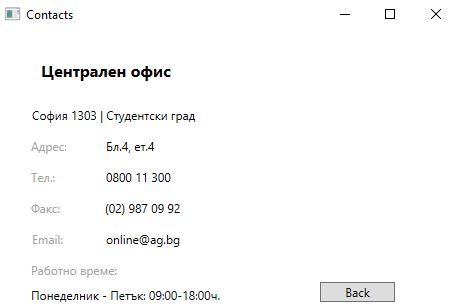
  
При избор от NavigaterWindow на Проверка в системата се зарежда PrintPage. Съдържа колите на юзъра. Той може да избере автомобил и да принтира данните за него (прехвърлят се във файл).  
  
  
  
  
  
Данните за избраният автомобил се сейфат във файл на име “test.txt”.

# 5. Потребителскo ръководство

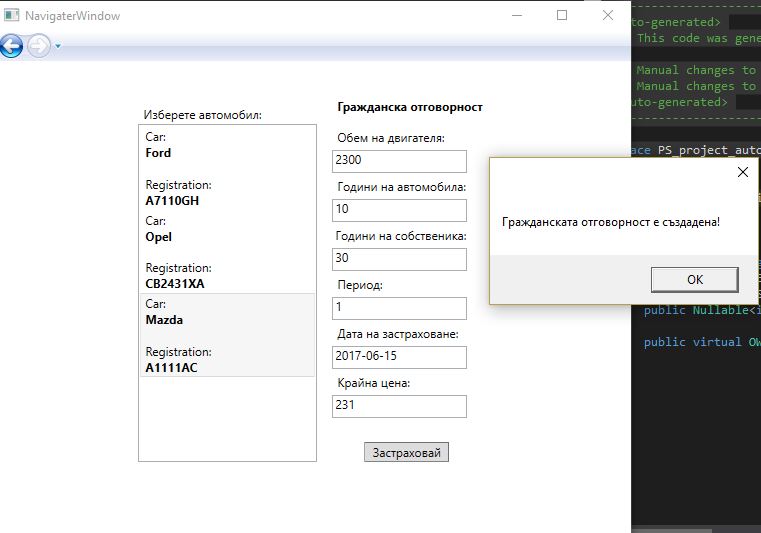
Нека първо започнем с логин формата,която е представена по-долу.Ако даденият потребител не притежава акаунт ще е необходимо първо да се регистрира през бутона регистрация



По-долу е представена регистрационната форма, през която даден юсър ще може да си създаде акаунт.Първо и последно име трябва да са между 1 и 20 символа.  
  
При срещане на трудности или проблеми може да се избере бутона Контакти, от който се появява прозорец с контакти за връзка.

  
След като успешно влезете в акаунта си, ще ви излезе прозорецът, представен по-долу.

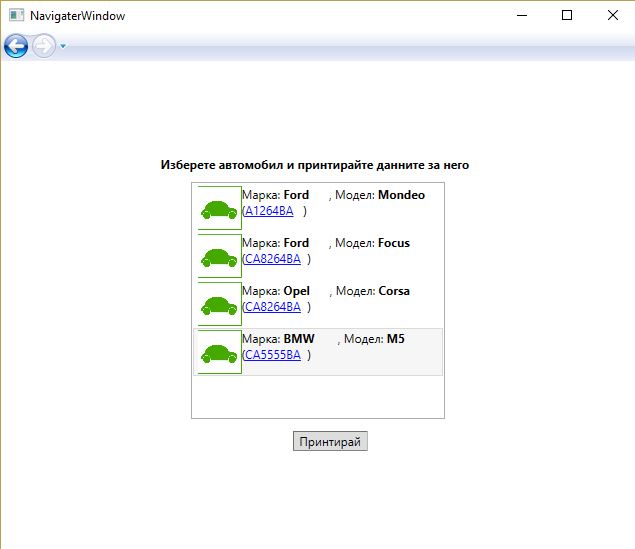
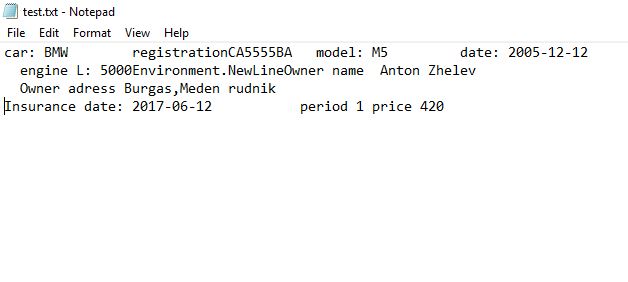


От менюто може да направим гражданска отговорност на автомобила си или да проверим данни за автомобилите Всеки юзър може да застрахова или да проверява данните само за собствените си автомобили.  


След въвеждане на данните и натискане на бутона застраховай излиза съобщение за завършена застраховка . При въвеждане на невалидни данни ще изведе съобщение за грешка.

При избор за проверка в системата се избира кола от полето и при натискане на принтирай ще се отпечатат данните за автомобила във файл.

*Фиг. 20 Преглед на изминали тестове*

   
  
  
Съдържанието на файла след избора за кола

# 

# 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/>

<http://sdi.bg/>

http://broko.bg/

# 8. Приложение

namespace PS\_project\_auto.ModelView

{

public class MainWindowViewModel : Window,INotifyPropertyChanged

{

AutoDBEntities ctx = new AutoDBEntities();

public MainWindowViewModel()

{

FillUser2s();

FillUsers();

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

ContactsButtonCommand = new RelayCommand(ContactsButton, param => this.canExecute);

LogInButtonCommand = new RelayCommand(LogInButton, param => this.canExecute);

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

}

public static Registration rr;

public void RegistButton(object sender)

{

if (ww != null)

{

ww.Close();

ww = null;

}

Application.Current.Windows[0].Close();

rr = new Registration();

rr.Show();

ctx.Dispose();

}

public static NavigaterWindow nw;

public void LogInButton(object sender)

{

bool flag = false;

foreach(USER u in Users)

{

if (userName == u.USER1.Trim()&&Password==u.PASSWORD.Trim() )

{

flag = true;

if (ww != null)

{

ww.Close();

ww = null;

}

Application.Current.Windows[0].Close();

nw = new NavigaterWindow(u);

nw.Show();

}

}

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

}

public ObservableCollection<USER> \_Users;

public ObservableCollection<USER> Users

{

get { return this.\_Users; }

set

{

this.\_Users = value;

NotifyPropertyChanged();

}

}

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); }

}

public void FillUser2s()

{

USER u = new USER();

u.USER1 = "12345678";

u.PASSWORD="12345678";

ctx.USERS.Add(u);

ctx.SaveChanges();

// ctx.Database.ExecuteSqlCommand("INSERT INTO OWNERS(ID,EGN,NAME,ADDRESS) values(5,'5','a','b')");

// ctx.SaveChanges();

}

static Contacts cc;

public static MainWindow ww;

public void BackButton(object sender)

{

if (cc != null)

{

cc.Close();

cc = null;

}

ww = new MainWindow();

ww.Show();

}

public void ContactsButton(object sender)

{

if (ww != null)

{

ww.Close();

ww = null;

}

Application.Current.Windows[0].Close();

cc = new Contacts();

cc.Show();

}

private ICommand registrationButtonCommand;

public ICommand RegistrationButtonCommand

{

get

{

return registrationButtonCommand;

}

set

{

registrationButtonCommand = value;

}

}

private ICommand loginButtonCommand;

public ICommand LogInButtonCommand

{

get

{

return loginButtonCommand;

}

set

{

loginButtonCommand = value;

}

}

private ICommand backButtonCommand;

public ICommand BackButtonCommand

{

get

{

return backButtonCommand;

}

set

{

backButtonCommand = value;

}

}

private ICommand contactsButtonCommand;

public ICommand ContactsButtonCommand

{

get

{

return contactsButtonCommand;

}

set

{

contactsButtonCommand = value;

}

}

private String userName;

public String UserName

{

get

{

return userName;

}

set

{

userName = value;

NotifyPropertyChanged();

}

}

private String password;

public String Password

{

get

{

return password;

}

set

{

password = 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));

}

}

}

}

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;

}

}

class InsuranceViewModel : Window, INotifyPropertyChanged

{ AutoDBEntities ctx = new AutoDBEntities();

public InsuranceViewModel()

{

initListBox();

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

}

private void InsuranceButton(object sender)

{

if (InsuranceDate == null || InsuranceDate == null || YearsOwner == null || Period == null || YearsCar== null)

{

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

return;

}

INSURANCE i = new INSURANCE();

i.DATE\_EXPIRE = insuranceDate;

i.ENGINE\_LITERS = engine;

i.OWNERAGE = yearsOwner;

i.PERIOD = period;

i.PRICE = finalPrice;

i.YEAROFCAR = yearsCar;

INSURANCE\_INFO info = new INSURANCE\_INFO();

if(SelectedCar == null){

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

return;

}

info.CAR = SelectedCar;

info.INSURANCE = i;

ctx.INSURANCE\_INFO.Add(info);

ctx.SaveChanges();

MessageBox.Show("Гражданската отговорност е създадена!");

}

private ICommand insuranceButtonCommand;

public ICommand InsuranceButtonCommand

{

get

{

return insuranceButtonCommand;

}

set

{

insuranceButtonCommand = value;

}

}

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;

FinalPrice="0";

NotifyPropertyChanged();

}

}

private float finalPrice;

public String FinalPrice

{

get

{

return finalPrice.ToString();

}

set

{

finalPrice = (engine\*(float)0.07)+(100-yearsOwner);

Console.WriteLine(finalPrice);

NotifyPropertyChanged();

}

}

private String insuranceDate;

public String InsuranceDate

{

get

{

return insuranceDate;

}

set

{

insuranceDate = value;

NotifyPropertyChanged();

}

}

private int yearsCar;

public String YearsCar

{

get

{

return yearsCar.ToString();

}

set

{

int res;

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

if (!isFloat)

{

MessageBox.Show("грешен формат година на автомобила");

return;

}

yearsCar = res;

NotifyPropertyChanged();

}

}

private int yearsOwner;

public String YearsOwner

{

get

{

return yearsOwner.ToString();

}

set

{

int res;

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

if (!isINT)

{

MessageBox.Show("грешен формат възраст на потребителя");

return;

}

if (res < 18 || res > 70)

{

MessageBox.Show("грешен формат възраст на потребителя");

return;

}

yearsOwner = res;

FinalPrice = "0";

NotifyPropertyChanged();

}

}

private int period;

public String Period

{

get

{

return period.ToString();

}

set

{

int res;

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

if (!isFloat)

{

MessageBox.Show("грешен формат период в години");

return;

}

period = res;

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));

}

}

}

public class PrintViewModel: Window,INotifyPropertyChanged

{

AutoDBEntities ctx = new AutoDBEntities();

public PrintViewModel()

{

initListBox();

PrintToFileButtonCommand = new RelayCommand(PrintFileButton, param => this.canExecute);

}

private void PrintFileButton(object sender)

{

if (SelectedCar == null)

{

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

return;

}

INSURANCE\_INFO i= GetInfo();

INFO\_COMPREHENSIVE\_COVER c = GetCover();

if (i == null && c == null)

{

MessageBox.Show("no data!");

}

String result = "car: " + SelectedCar.MARK + " registration" + SelectedCar.REGISTRATION + " model: " + SelectedCar.MODEL + " date: " + SelectedCar.DATA + " engine L: " + SelectedCar.ENGINE\_LITERS + " \n"

+ "Owner name " + SelectedCar.OWNER.NAME + " Owner adress " + SelectedCar.OWNER.ADDRESS + " \n";

if (i != null)

{

result+="Insurance date: " + i.INSURANCE.DATE\_EXPIRE + " period " + i.INSURANCE.PERIOD + " price " + i.INSURANCE.PRICE + " \n";

}

if(c != null)

{

result+= "Comprehensice date" + c.COMPREHENSIVE\_COVER.DATE\_EXPIRE + " final price " + c.COMPREHENSIVE\_COVER.FINAL\_PRICE + " \n";

}

System.IO.File.WriteAllText("test.txt", result);

}

public INFO\_COMPREHENSIVE\_COVER GetCover()

{

var q = (from a in ctx.INFO\_COMPREHENSIVE\_COVER

select a).ToList();

ObservableCollection<INFO\_COMPREHENSIVE\_COVER> covers = new ObservableCollection<INFO\_COMPREHENSIVE\_COVER>(q);

foreach(INFO\_COMPREHENSIVE\_COVER c in covers){

if (c.CAR.ID == SelectedCar.ID)

{

return c;

}

}

return null;

}

public INSURANCE\_INFO GetInfo()

{

var q = (from a in ctx.INSURANCE\_INFO

select a).ToList();

ObservableCollection<INSURANCE\_INFO> infos = new ObservableCollection<INSURANCE\_INFO>(q);

foreach (INSURANCE\_INFO c in infos)

{

if (c.CAR.ID == SelectedCar.ID)

{

return c;

}

}

return null;

}

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 ICollection<CAR> \_cars;

public ICollection<CAR> cars

{

get { return this.\_cars; }

set

{

this.\_cars = value;

NotifyPropertyChanged();

}

}

private CAR \_selectedCar;

public CAR SelectedCar

{

get

{

return this.\_selectedCar;

}

set

{

this.\_selectedCar = value;

NotifyPropertyChanged();

}

}

private ICommand printToFileButtonCommand;

public ICommand PrintToFileButtonCommand

{

get

{

return printToFileButtonCommand;

}

set

{

printToFileButtonCommand = value;

}

}