**CAR RENTAL IN C#**

*Transylvania University of Brasov*

*Faculty of Electrical Engineering and Computer Science*

**-VISUAL STUDIO 2019 WINDOWS FORMS APP-**

**Cuvânt de început**

În această documentație, doresc să vă prezint aplicația CAR RENTAL realizată pe parcursul anului 3 la Facultatea de Inginerie Electrică și Știința Calculatoarelor, departamentul de Calculatoare.

Aceasta a fost dezvoltată cu ajutorul mediului de dezvoltare integrat Visual Studio 2019, fiind o aplicație de tip Windows Forms, hard-codată în C#.

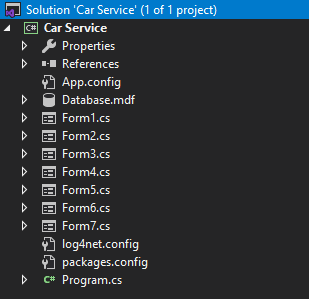
**Descriere pe scurt**

Aplicația în sine trebuie să fie capabilă de a realiza un sistem de închiriere a mașinilor, returnând o evidență a acestor înregistrări.

Se va prezenta folosirea principiilor de tip SOLID, ORM, șabloanelor FACTORY, urmând ca în continuare să descriem împreună aplicația.

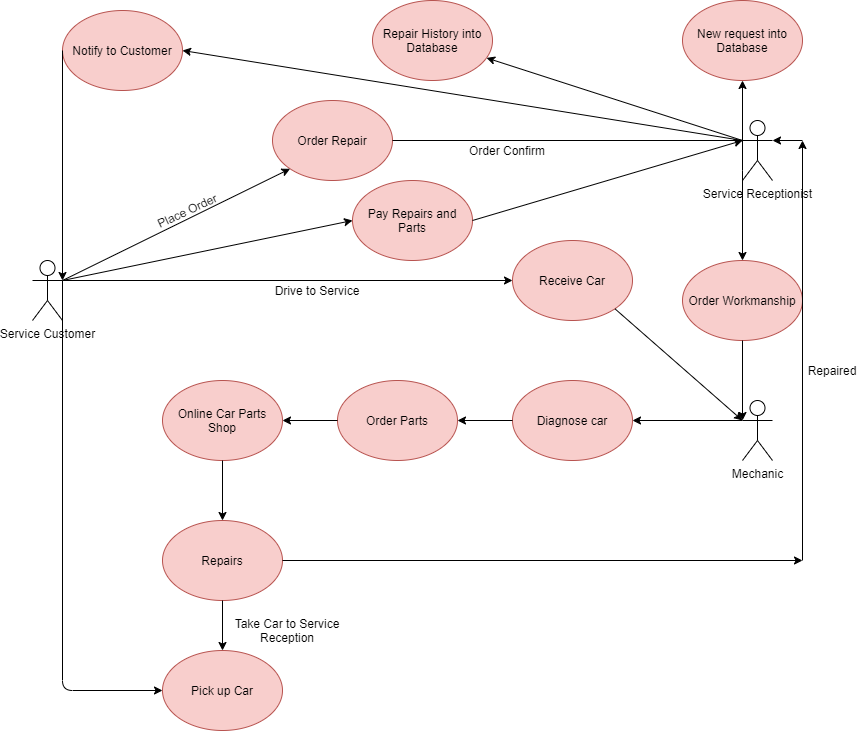
**Structură**

Aplicația dată prezintă următoarea structură:



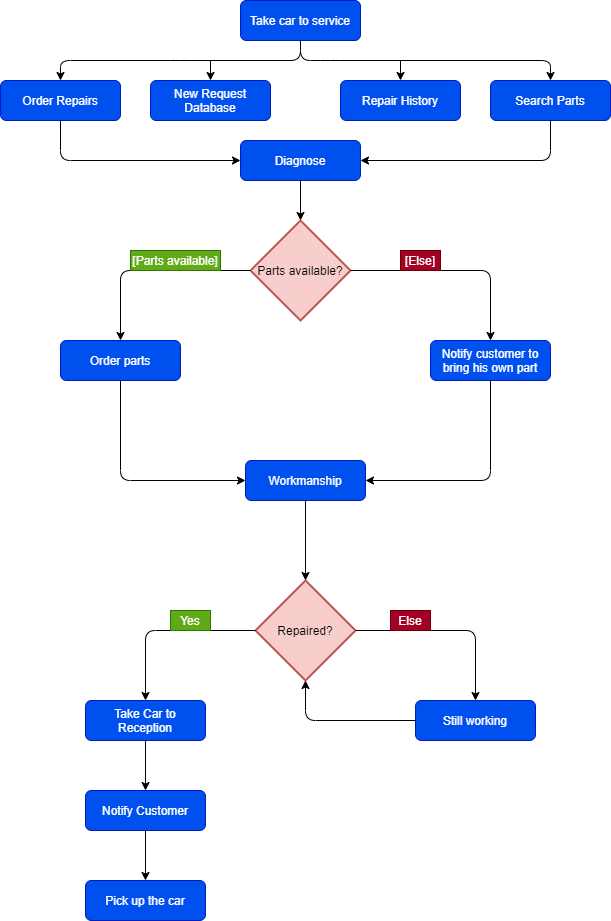
**Șapte Forme cu diferite hard-codări, o bază de date locală SQL ce deține 2 tabele, Login și Rent, și un fișier de log realizat prin utilizarea extensiei log4net.**

**Diagrama Use-Case**

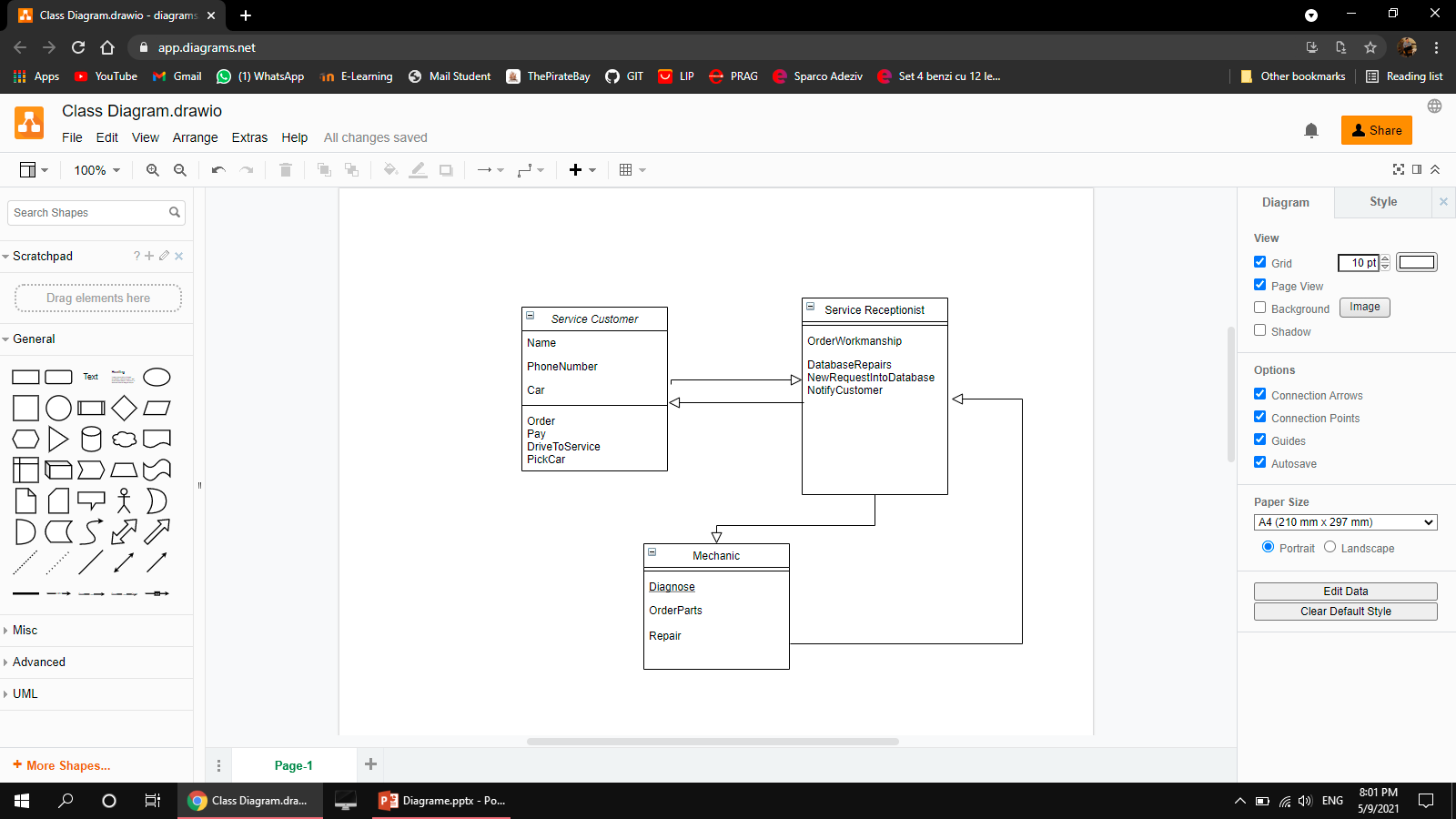
****

**Se poate observa structura unei aplicatii de tip service, dar pe parcurs am ales să îmi modific tema de proiect, realizând că utilitatea aplicației ar fi mai potrivită pe o tematică asemănătoare celei de închirieri mașini.**

**Diagrama Activity**

****

**Diagrama Class**



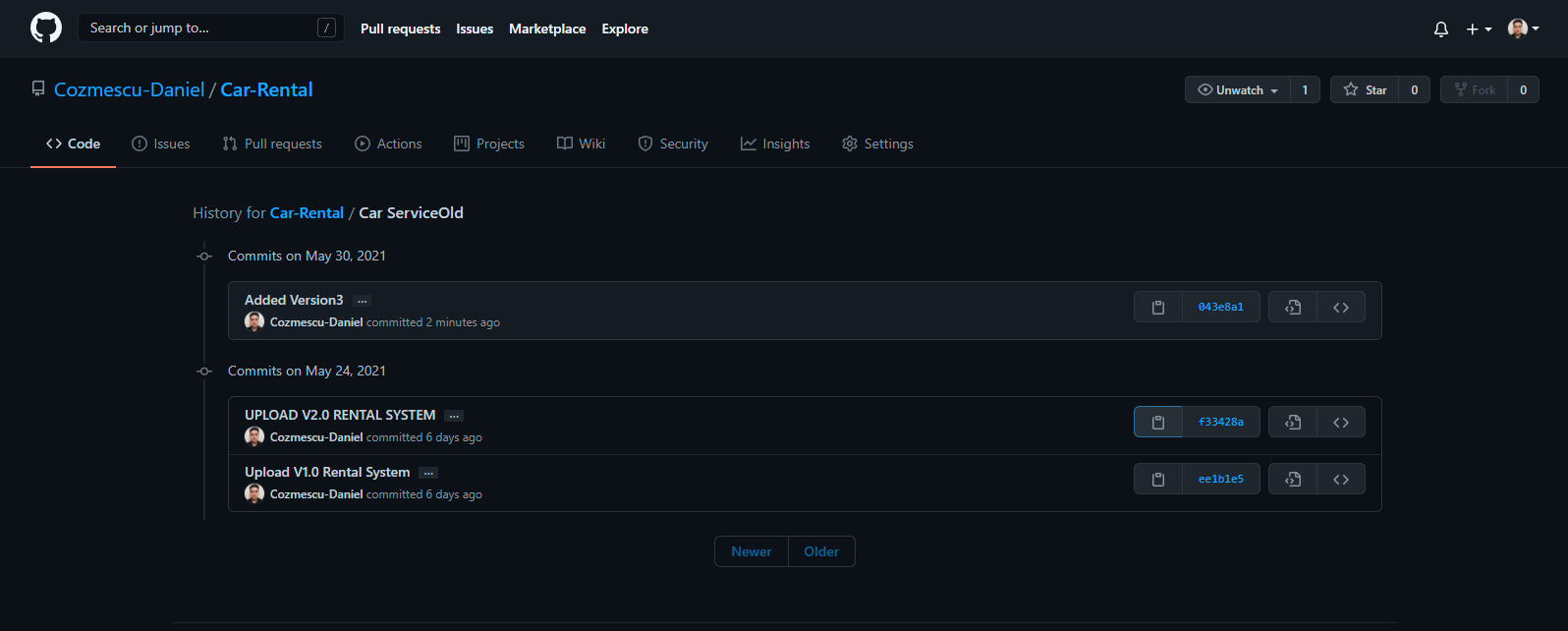
**Versionare**

După cum se poate observa în snippet-ul următor, am folosit un sistem de versionare bazat pe GITHUB, folosind upload-ul modificărilor aduse pe parcurs programului.

S-au utilizat mai multe comenzi de tip commit și push pentru versionarea aplicației cu un număr total de 3 versiuni.

Pentru mai multe detalii despre features implementate de-alungul proiectului , verificați descrierea fiecărui commit.

Repository -> [Car-Rental](https://github.com/Cozmescu-Daniel/Car-Rental)



Comenzile de tip clone ajută la extragerea unui repository remote din Github, acesta devenind unul de tip local, în care modificăm cu varianta noastră de cod și cerem privilegii creatorului de a urca la loc proiectul cu un **pull** request.

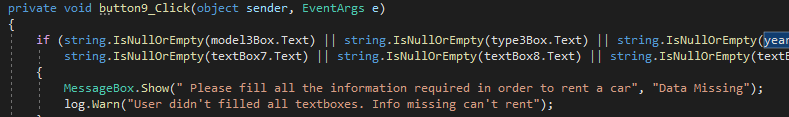
Pe de altă parte, când se lucrează la un proiect personal de către o singură persoană este îndeajuns să facem **push** de fiecare dată când modificăm ceva fără a mai cere permisiuni, noi fiind creatorii repository-ului.

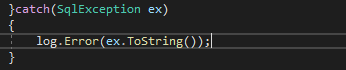
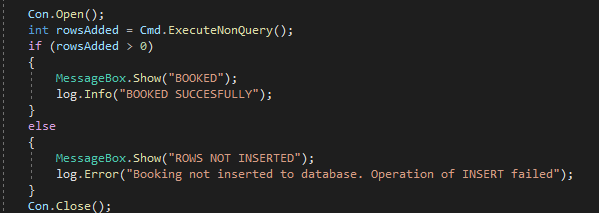
**Logs**

Pentru a ține evidența unor erori, bug-uri, warning-uri sau a info-urilor, am folosit biblioteca log4net.

Se poate observa cu ușurință prezența ei prezentată în snippet-ul de mai sus. (Vezi Structură ).

Totodată, în cod am stabilit mai multe tipuri de log files după cum se poate observa mai jos:

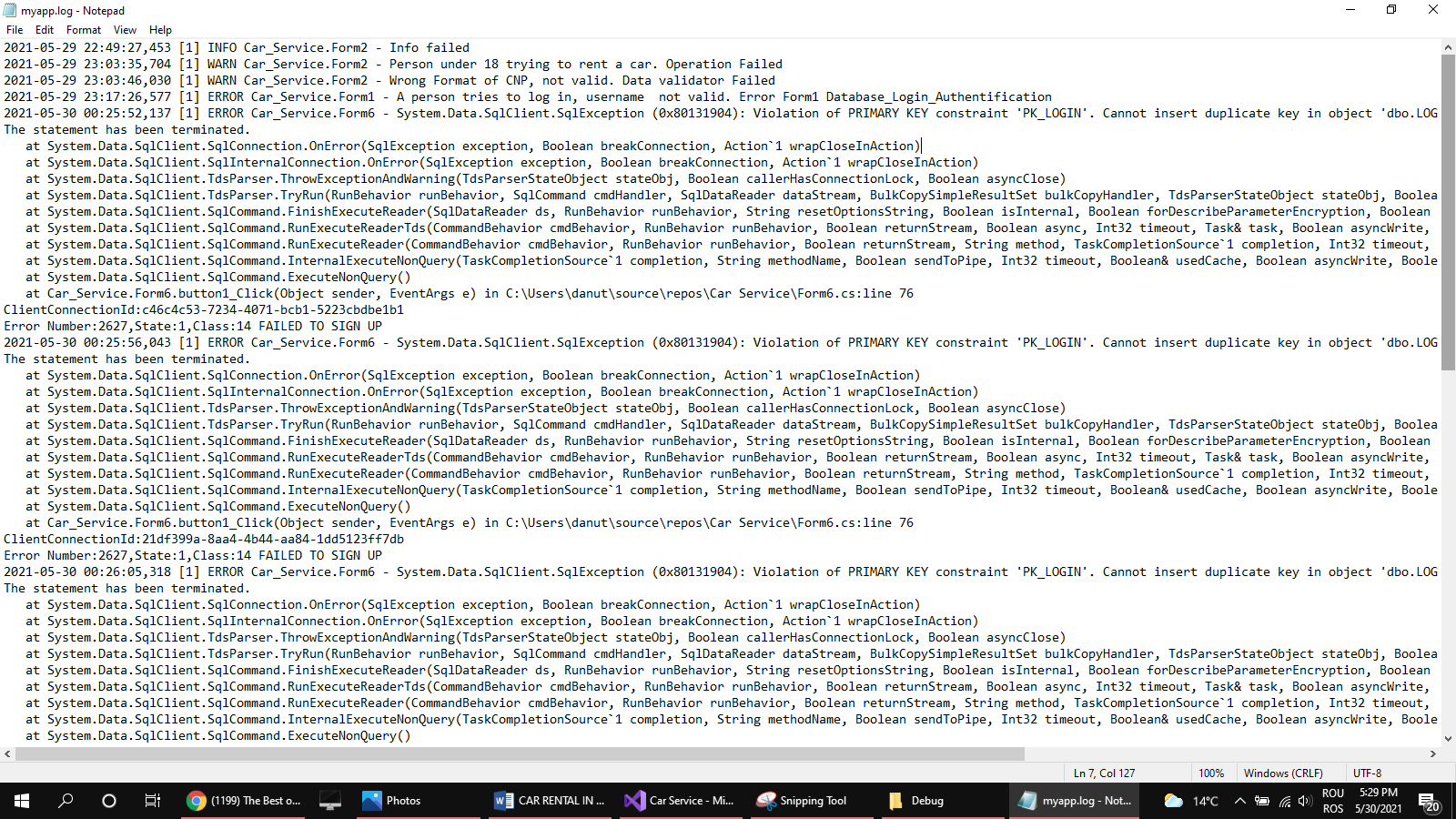


**Unități de testare**

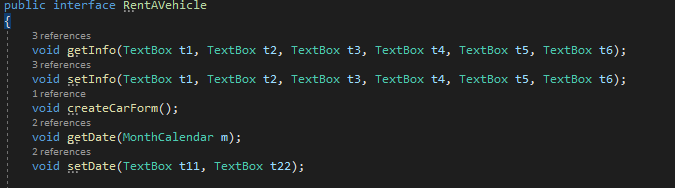
Personal, am ales să folosesc testarea manuală, deși există și anumite biblioteci pentru astfel de evenimente.

Am introdus intenționat anumite variabile greșite sau am lăsat câmpuri de tipul celor textBox goale, pentru a testa funcționalitatea, iar cu ajutorul log file-ului am scris într-un fișier de tip log tot output-ul corespunzător erorilor și warning-urilor afișate pe ecran, așa cum se poate observa mai jos:



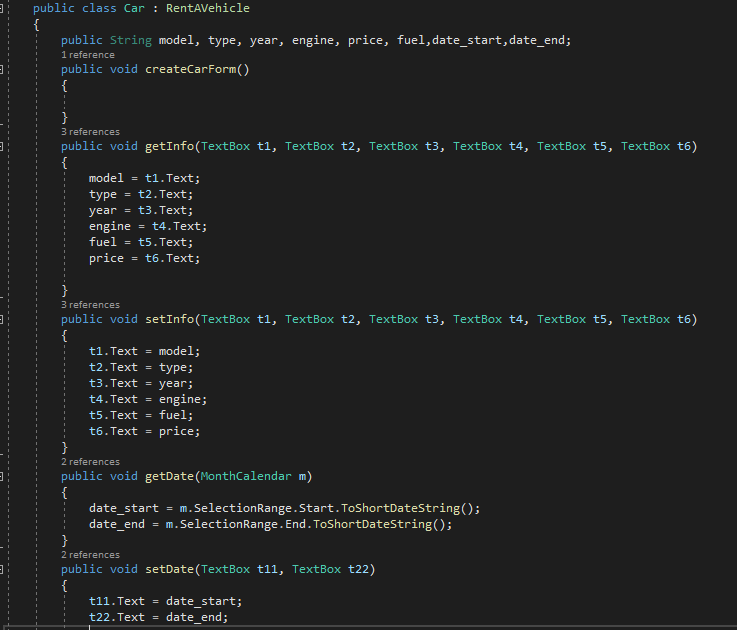
**Șablon de tip Factory**

Pentru realizarea preluării informațiilor din textbox-uri și inserarea acestora în forma de Booking, am folosit un șablon de tip Factory.



În această poză se poate observa interfața RentAVehicle ce poate fi implementată de orice tip de clasă care pune la dispoziție închirierea unui vehicul, cum ar fi o barcă, o bicicletă, o mașină ș.a.m.d.

Personal am avut nevoie doar de implementarea clasei Car, deoarece aplicația prin enunț poate închiria doar mașini, dar existând o interfață și un cod modular, putem adăuga cu ușurință alt tip de vehicul.



După cum se poate observa, am folosit clasa Car pentru a implementa definițiile de funcții din interfața RentAVehicle.

Am implementat metoda getInfo ce va prelua date din textbox-urile unde se găsesc mașinile și le va seta in textbox-urile formei de Booking, astfel procesându-le.

Totodată am folosit această clasă pentru a prelua și seta data din calendarul adăugat, pentru fiecare mașină în parte.

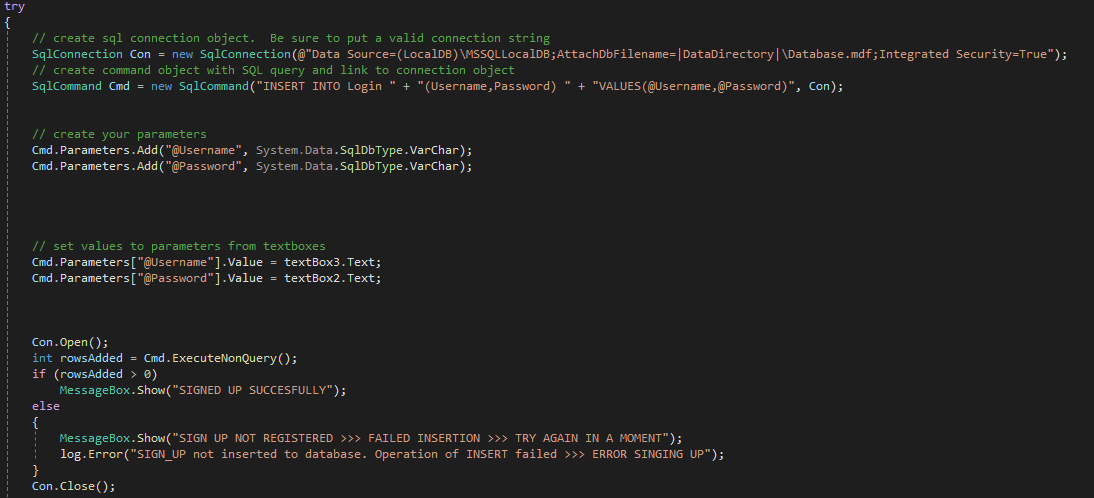
ORM

Personal am renunțat la folosirea unui ORM asemenea EntityFramework în .NET, deoarece nevoia de a folosi un Object-Relational Mapping a fost scăzută.

Aș putea menționa totuși ca și exemplu faptul că un ORM este un mid-software, o legătură pentru a efectua operații simple de query asupra unei baze de date prin intermediul codului, pentru a ușura munca programatorului și a nu efectua mereu o conexiune cu baza de date.

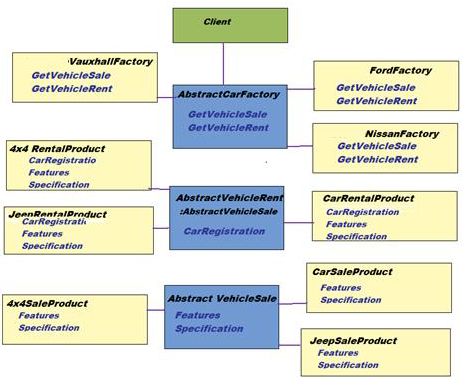
Eu unul am ales să deschid conexiunea de câte ori am avut nevoie și să setez parametrii, un string connection, un string query și operațiuni de executare query, deoarece tot ce a fost nevoie era să aplic comenzi de insert, delete sau select asupra a două tabele din baza de date locală, nefiind un proiect de amplitudine.

Mai jos puteți observa „simularea” unui ORM creeat de mine și incorporat într-un bloc de tip try-catch.



**ISP - Principiul de segregare a interfeței (S.O.L.I.D)**

Așa cum am menționat și mai sus, am folosit o intefață RentAVehicle pentru a generaliza închirierea de vehicule și pentru ca pe viitor să putem implementa altă clasă în mod particular numită Bike sau PrivateBoat.



După cum se poate observa în diagrama de mai sus, o interfață se poate folosi pentru implementarea mai multor tipuri de clase.

**Șabloane de tip Creațional, Structural și Behavioral**

În cadrul șablonului de tip structural, se poate observa cu ușurință faptul că pentru a muta informațiile dintr-o formă în alta am creeat o clasă Car pentru manipularea datelor, aceasta trebuind doar instanțiată cu valori e tip textBox pentru a muta datele.

Aceasta este dovada principiului de tip DO NOT REPEAT.

Totodată am folosit principiul SINGLE RESPONSABILITY asupra butoanelor.

Butoanele au unul dintre cele două roluri atribuite în acest program: de afișare a unor panouri/forme noi sau de efectuare a transportului datelor între textBox-uri.

În cadrul șablonului de tip creațional, am creeat un șablon de tip Factory pentru a nu dezvălui clientului logica din spatele aplicației.

De asemenea au fost folosite și SINGLETON-uri, astfel încât am instanțiat anumite obiecte doar o singură dată cum ar fi SpeechSynthetizer sau Assistant.  
În cadrul șablonului de tip behavioral, ne folosim de ITERATOR pentru a accesa baza de date și a căuta în aceasta înregistrări cu matching între ID introdus și ID existent.

Totodată am folosit VISITOR pentru a adăuga o nouă operație fără a modifica ierarhia clasei, cum ar fi metoda log.Warning(ex.toString()).

În încheiere, aș dori să le multumesc profesorilor îndrumători Lupașcu Teodor și Danciu Gabriel pentru ajutorul oferit și totodată să vă multumesc pentru atenție.

Biblioteci folosite:

Log4Net, System.Speech, System.Web, System.Net, System.Sql, System.Collections, System.Data, System.Data.SqlClient, System.Threading.

Webografie:

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/csharp-programming-language/>

<https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/>

<https://docs.microsoft.com/en-us/troubleshoot/dotnet/csharp/create-sql-server-database-programmatically>