Documentație proiect 1 CarService

Subsemnata Bîrsan Ioana, declar pe proprie răspundere că acest cod nu a fost copiat din Internet sau din alte surse.

Bibliografie:

- https://blog.devart.com/set-identity-and-computed-properties-in-entity-framework-without-triggers.html
- https://stackoverflow.com/questions/25894587/how-to-update-record-using-entity-framework-6
- https://stackoverflow.com/questions/35799017/adding-attributes-to-ef6-generated-entity-classes
- http://alexwolfthoughts.com/adding-validation-metadata-to-entity-framework-generated-classes/
- https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/database-first-development/enhancing-data-validation
- https://dev.to/kylegalbraith/getting-familiar-with-the-awesome-repository-pattern--1ao3
- https://deviq.com/repository-pattern/

Noutăți:

Deoarece aveam nevoie să adaug reguli de validare folosind atribute la nivelul proprietăților din clasele parțiale generate, iar acestea s-ar fi rescris la următoarea regenerare, am folosit următoarea abordare:

- pentru fiecare clasă parțială generată (e.g. `Auto`), am creat o clasă nouă (e.g. `AutoMetadata`), în care am adăugat reguli de validare la nivel de proprietate
- pentru fiecare clasă parțială generată am creat o nouă clasă parțială (e.g. `Auto`) și am utilizat atributul MetadataType (e.g. [`MetadataType(typeof(AutoMetadata))`])

Aceste reguli sunt acum impuse în proiect și nu vor fi șterse în momentul în care se regenerează clasele. Clasele create se află în folderul **Models**.

Am implementat șablonul Repository pentru a oferi un nivel de abstractizare între nivelul de persistență (baza de date AUTO) și nivelul de business (`ICarService`, `CarService`). Am creat o interfață generică `IRepository` în care expun următoarele metode:

```
public interface IRepository<T>
      {
      T GetById(int id);
      IReadOnlyList<T> GetAll();
      void Create(T item);
      void Delete(int id);
      void Update(T item);
      bool Exists(int id);
      void SaveChanges();
      }
```

Pentru fiecare model, am implementat această interfață, folosind ca dependență contextul generat `CarServiceModelContainer`. Unde a fost cazul am expus și alte metode necesare pentru logica aplicației. Nivelul acesta se regăsește în folderul **Repository**.

Nivelul de logică al aplicației, care expune API-ul cerut, se află în folderul **Service**. Am creat o interfață: `ICarService` și o clasă care o implementează: `CarService`. Dependențele folosite sunt cele din nivelul de repository.

Pentru testare, am creat în cadrul aceleiași soluții un proiect de tip Console App (TestCarService) căruia i-am adăugat referință către proiectul de tip Class Library (CarService). În arhivă voi adăuga doar fișierul numit `Program.cs`.

Actualizare Proiect 1

1. Am eliminat folderul **Repository,** eliminând astfel redundanța (Entity Framework deja implementează Repository pattern, așa cum ați menționat și dumneavoastră).
2. Am eliminat folderul **Service**, în locul lui am introdus folderul **Api** în care am adăugat doar clasa care expune metodele publice necesare (CarServiceApi)
3. Am corectat erorile de logică din modelul creat cu ADO.Net, am actualizat și relațiile de asociere dintre modele, am adăugat o nouă valoare enum-ului StareComandă.

Documentație proiect 2 CarService

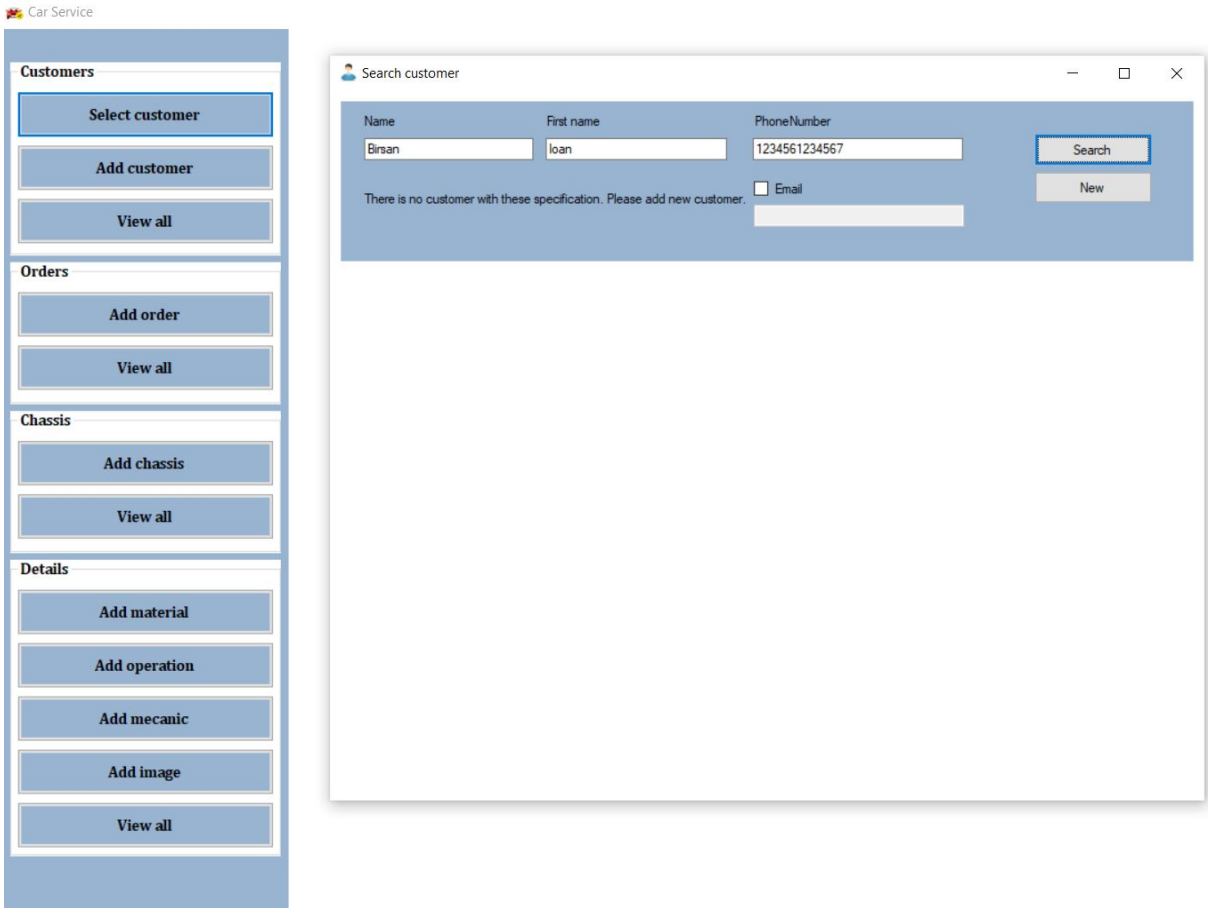
Bibliografie:

- https://docs.microsoft.com/en-us/dotnet/framework/winforms/
- https://docs.microsoft.com/en-us/dotnet/api/system.drawing.image?view=netframework-4.7.2
- https://www.youtube.com/watch?v=sGP6u68k2hc
- https://stackoverflow.com/questions/3801275/how-to-convert-image-to-byte-array
- https://www.youtube.com/watch?v=C9s0H6yeFLQ

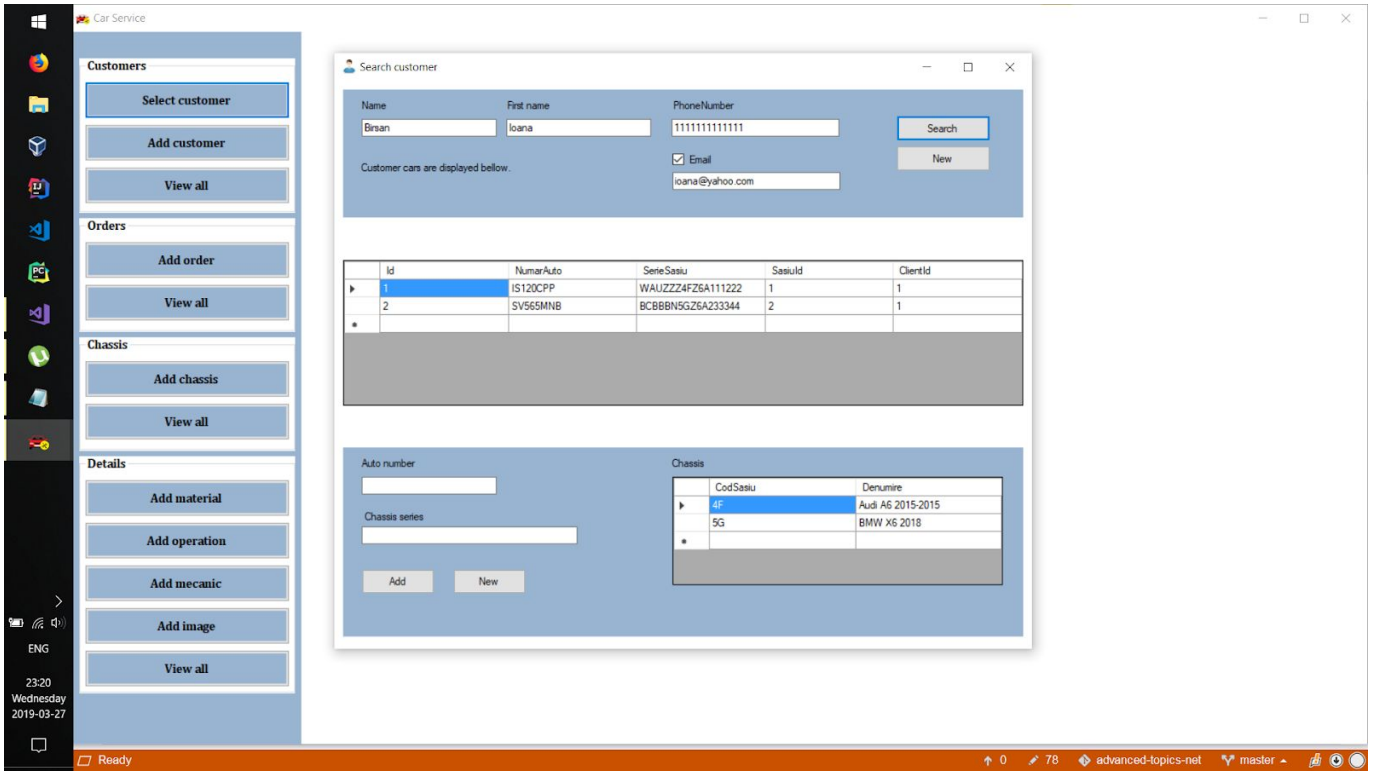
Interfață:

Pentru această secțiune voi adăuga niște screenshot-uri, însoțite de explicații.

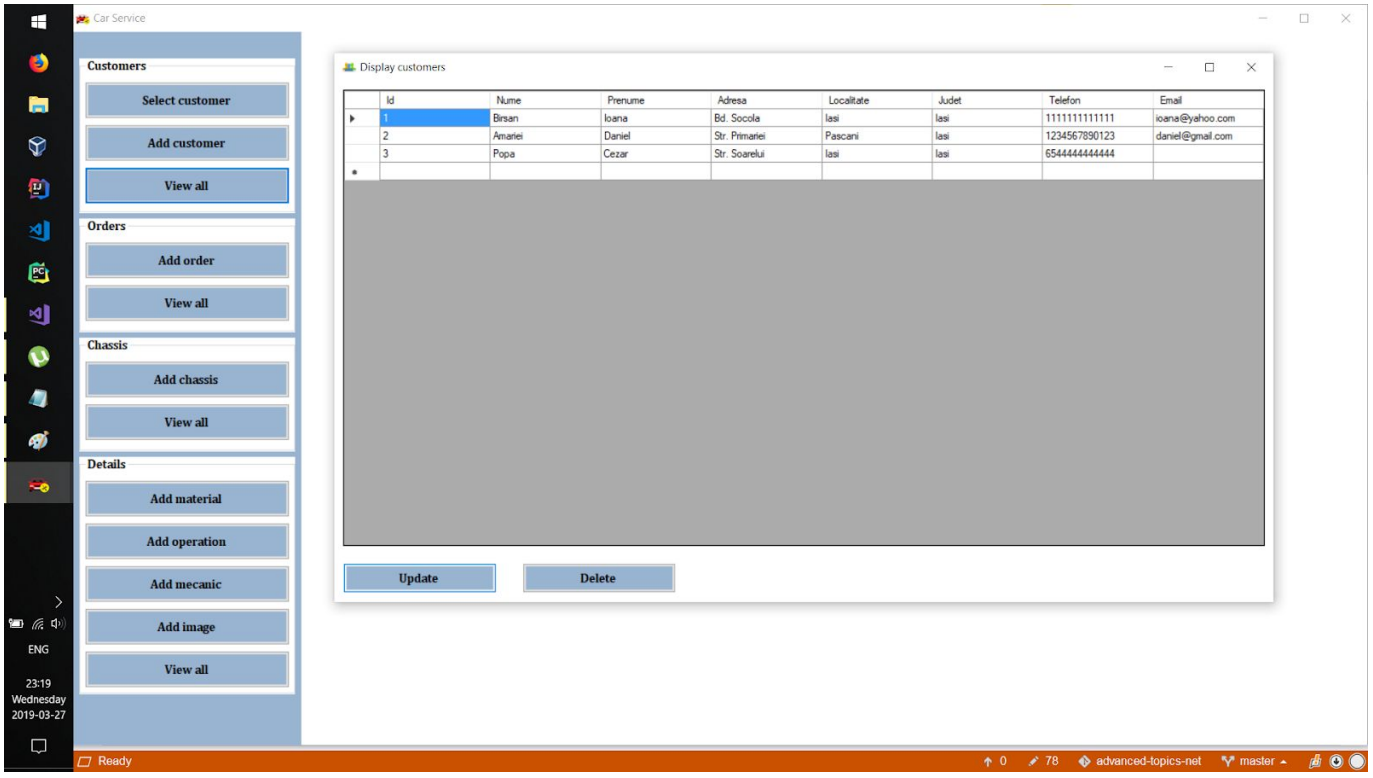
În prima etapă, se caută existența unui client. În cazul în care acesta nu există se va afișa un mesaj corespunzător. Pentru fiecare câmp ce trebuie completat există validări corespunzătoare, iar butonul de *Search* nu este activat decât în momentul în care toate validările trec și sunt completate toate câmpurile obligatorii.



Dacă există clientul căutat, atunci se vor afișa mașinile sale, precum și opțiunea de a adăuga o nouă mașina. La fel ca în cazul anterior, are loc validarea câmpurilor necesare.

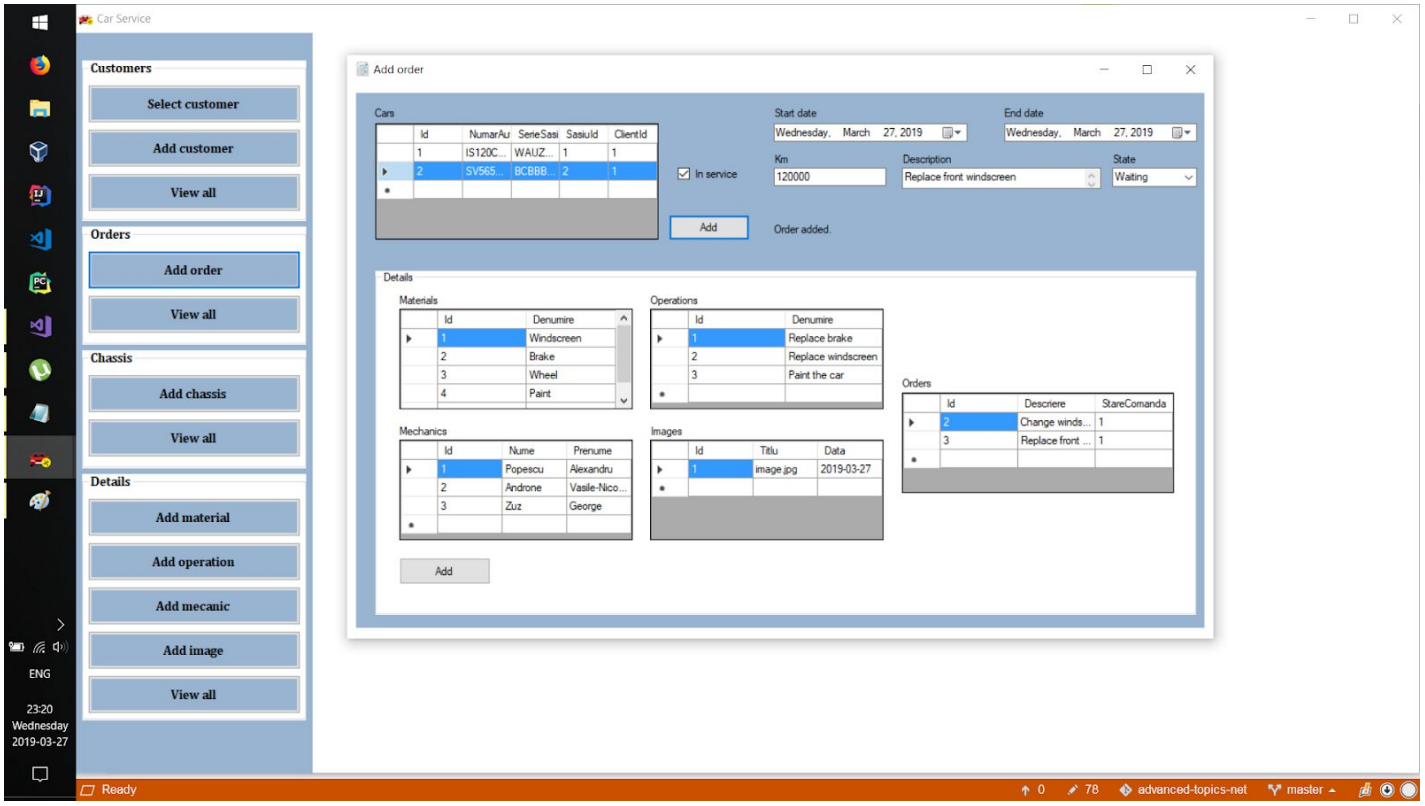


Pentru fiecare grup (Customers, Orders, Chassis, Details) există posibilitatea de a vizualiza toate înregistrările existente. La selectarea unui rând, avem posibilitatea de a actualiza informațiile existente, dar și de ștergere a unei înregistrări. Am exemplificat mai jos doar pentru Customers.

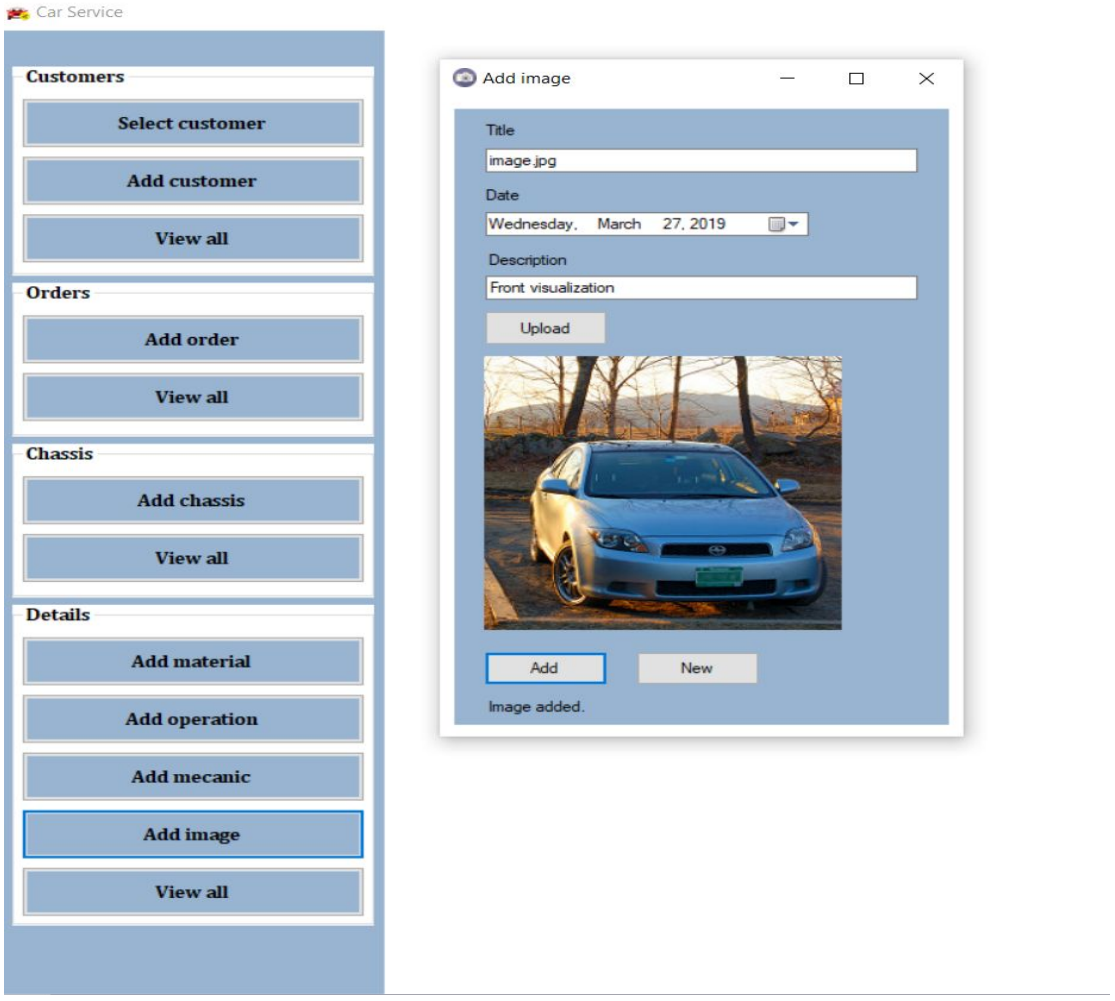


Pentru plasarea unei comenzi se alege o mașină, împreună cu toate datele necesare creării unei comenzi. De asemenea, datele sunt validate corespunzător. În funcție de opțiunea de a lăsa/nu mașina în service, se va activa/dezactiva posibilitatea de completare a câmpului Km.

După plasarea unei comenzi, se poate opta pentru adăugarea detaliilor necesare procesării comenzii selectate. Are loc calculul costului total pentru reparații (am considerat și un cost de service), împreună cu actualizarea stocului de materiale.



Aici este prezentată opțiunea de adăugare a unei imagini.



Vizualizare înregistrări detalii: materiale, mecanici, operațiuni și imagini. Pentru fiecare avem posibilitatea de editare sau ștergere.