TVIS

Traffic violation inquiry system : TVIS

سامانه استعلام تخلفات رانندگی : ساتر

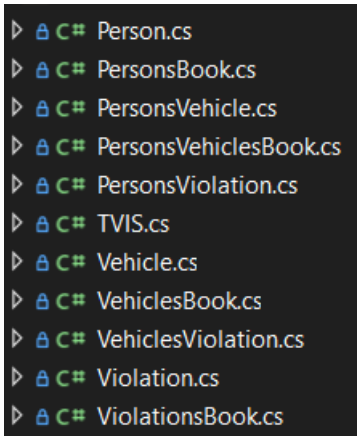
علی سلیمانی درچه 996203168

سهیل صالحی 996203195

پروژه پایگاه داده استاد شمس

در این پروژه از فریمورک **WPF** و دیزاین پترن **MVVM** و پایگاه داده **SQLServer** استفاده کردیم.

Model های ما را در روبرو میبینید:



برای هر Person یک مدل داریم که مجموعه ای از انها نیز توسط یک مدل به نام PersonsBook کنترل (اضافه،حذف،آپدیت و نمایش) میشوند.

برای هر Vehicle نیز یک مدل داریم که مجموعه ای از آنها نیز توسط یک مدل با نام VehiclesBook کنترل میشود.

برای هر Violation یک مدل داریم که مجموعه ای از آنها نیز توسط یک مدل به نام ViolationsBook کنترل میشود.

همچنین برای هرعضو از جدول رابطه چند به چند بین Person و Vehicle یک مدل نیاز داریم (PersonsVehicle) که مجموعه ای از آنها توسط یک مدل به نام PersonsVehiclesBook کنترل میشود.

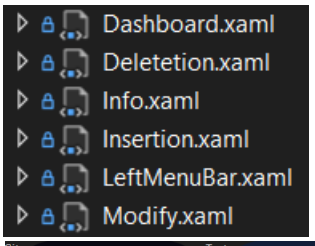
برای هر عضو از جدولی که عضو اول شماره پلاک و عضو دوم جریمه های یک شخص با آن پلاک کرده است از مدل PersonsViolation استفاده میکنیم(مربوط به قسمت گرفتن کوئری است).

برای هر عضو از جدولی که عضو اول نام , عضو دوم نام خانوادگی و عضو سوم جمع جریمه های شخص با شماره پلاک مشخص است از مدل VehiclesViolation استفاده میکنیم(مثل قبلی).

در نهایت میرسیم به مدل اصلی کل برنامه که قرار است کل عملیات نگهداری و کنترل را انجام دهد، مدل TVISModel در فایل TVIS.cs وظیفه کنترل کل عملیات ها را دارد.

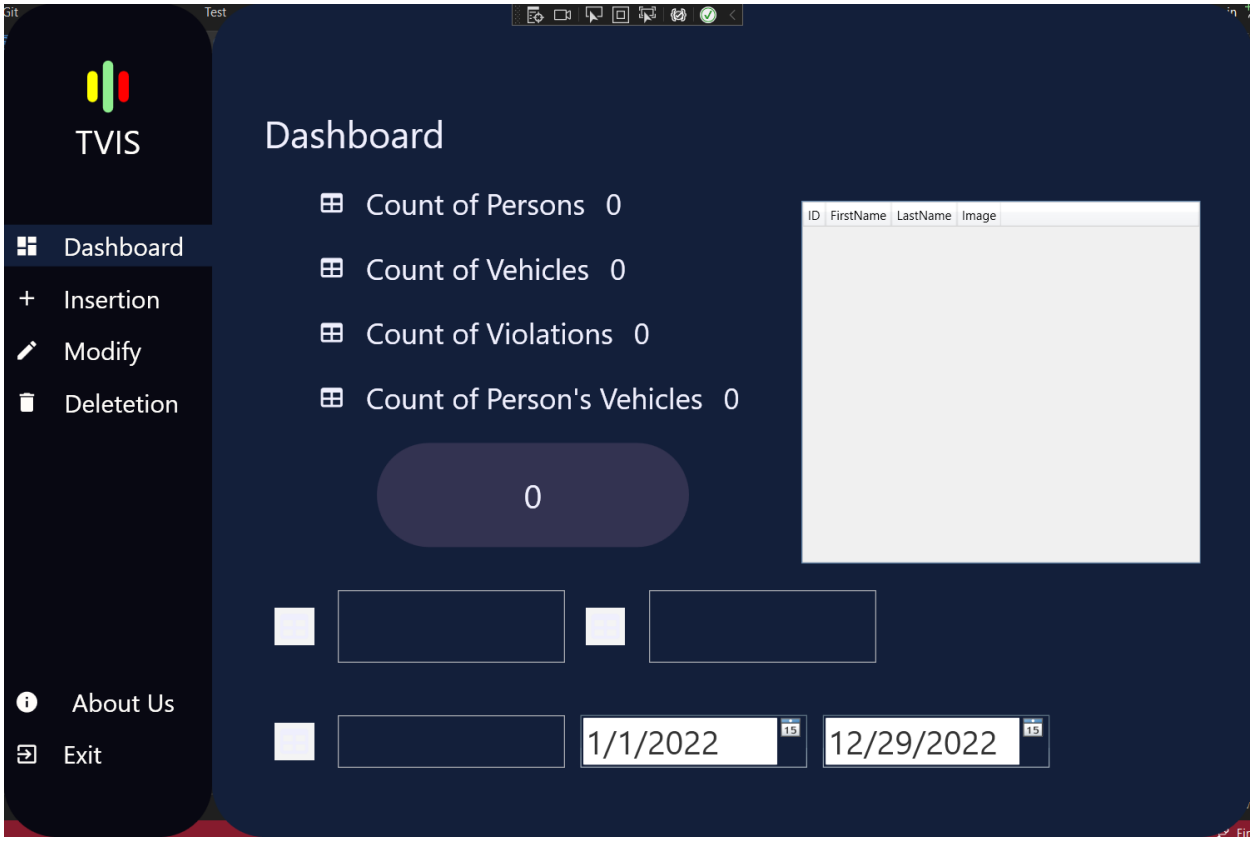
همچنین در این بخش عملیات های دریافت ، تغییر ، ارسال و حذف از دیتابیس نیز انجام میشود که توسط آبجکتی از نوع Database در فولدر Services که این ابجکت در کلاس App ساخته شده است.

نکته ای که هست ما در این برنامه چون قصد داشتیم کامل پیاده سازی شود یکسری کار اضافه که هیچ موقع در برنامه نیاز به استفاده از آنها نبوده با linq پیاده کردیم (کل قسمت های خواسته شده در فایل پروژه بصورت sql پیاده سازی شده و از آنها در برنامه استفاده میکنیم، جلوتر به قسمت دیتابیس میرسیم.)



View ها را در روبرو میبینیم:

در برنامه ما با توجه به دیزاینی که انتخاب کردیم قرار بر این بود که یک سایدبار منو داشته باشیم و در راست آن یک پنل که View های برنامه در آن قسمت نمایش داده میشوند و کل اتفاقات برنامه در یک Window رخ میدهد.



در بالا یک نمای کلی از برنامه ما و ویو داشبورد داریم.

در ویو داشبورد در ابتدا 4 سطر است که هر سطر تعداد اعضای هر جدول را نمایش میدهند همچنین در سمت چپشان یک باتن وجود دارد که اگر هر کدام از آن ها را بزنیم در جدول روبرو اطلاعات مربوط به آن جدول نمایش داده میشود.

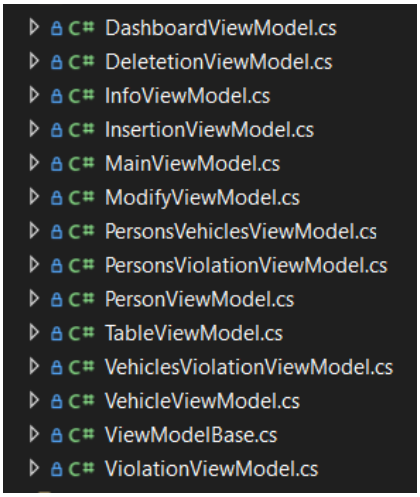
در ادامه یک سطر دیگر داریم که یک عدد را نمایش میدهد ، این عدد جمع کل جریمه ها را نمایش میدهد.(این قسمت نیز با دستورات SQL انجام میشود)

در ادامه دو سطر دیگر داریم که نیاز به توضیح دارد: سطر اول حاوی 3 بخش است، بخش اول یک باتن و یک تکست باکس است که درواقع در این قسمت با وارد کردن یک کدملی و زدن باتن در همین سطر ستون سوم که الان چیزی وجود ندارد (بعد از اخرین تکست باکس سمت راست در این سطر) عکس آن شخص نمایش داده میشود یه همچنین در جدول نیز پلاک های ماشین هایی که این شخص بوسیله انها مرتکب خلاف شده است به همراه مجموع جریمه هایی که آن ماشین کرده است نمایش داده میشود.

در ستون بعنی نیس به همین شکل فقط در تکست باکس ما شماره پلاک ماشینی را میگیریم و با زدن باتن کناریش نام و نام خانوادگی و جمع کل جریمه های آن شخص با این ماشین را نمایش میدهیم.

در سطر اخر هم یک کد ملی و دو تاریخ میگیریم و تمامی خلاف های ان شخص در آن بازه زمانی را با کلیک بر روی باتن کناری در جدول نمایش میدهیم.

در ویو های Isertion,Modify,Deletetion به ترتیب از چپ به راست عملیات های اضافه ، تغییر (آپدیت) و حذف داده ها را انجام میدهیم. در ویو Info هم فقط اطلاعات من و هم تیمی به همراه لینک گیتهاب کد پروژه قرار دارد . ویو LeftMenuBar هم همان یوزر کنترل منو است که از آن در صفحه اصلی استفاده میکنیم.



ViewModel ها را در روبرو میبینیم:

ما یک viewmodel برای اینکه همه ویو مادل ها از آن اث بری کنند نیاز داشتیم زیرا قرار است همه viewmodel ها در یک فیلد نگهداری شود ، زیرا برای تغییر ویو مادل در یک صفحه ما یک ویو مادل را در یک فیلد نگهداری میکنیم و هربار که در منو شخص ویو دیگری را انتخاب کرد ، درهمان فیلد مربوطه ویو مادل مربوط به ویو جدید باید قرار داده شود پس برای همین همه ویو مادل ها را از یک کلاس باید ارث بری کنیم. همچنین کلاس مادر از اینترفیس INotifyPropertyChanged ارث بری میکند زیرا میخواهیم عملیات های ما با تغییر اطلاعات در ویو ها تغییرات در ویو مادل ما و همینطور با تغییر Property مربوطه در قسمت کد ما ، تغییر در روی ایتم ها در ویو اعمال شود و این binding ها به درستی کار کند و سینک باشد. این کلاس را ViewModelBase نام گذاری میکنیم و دیگر همه کلاس هایمان در ویو مادل ها از این کلاس اث بری میکند.

ویو مادل های DashboardViewModel, DeletetionViewModel, InsertionViewModel, ModifyViewModel همگی برای کنترل ویو های اصلی یعنی داشبورد ، اضافه ، حذف و تغییر هستند. ویو مادل InfoViewModel یک کلاس خالی است و هیچ عملیاتی نیاز به انجام ندارد زیرا در ویو Info ما داده ای دریافت و ست نمیکنیم بلکه یک صفحه ثابت داریم ولی چون شیوه تغییر صفحه ما به شکلیست که با تغییر ویو مادل ، ویو تغییر میکند پس یک کلاس خالی نیز برای این بخش ساختیم.

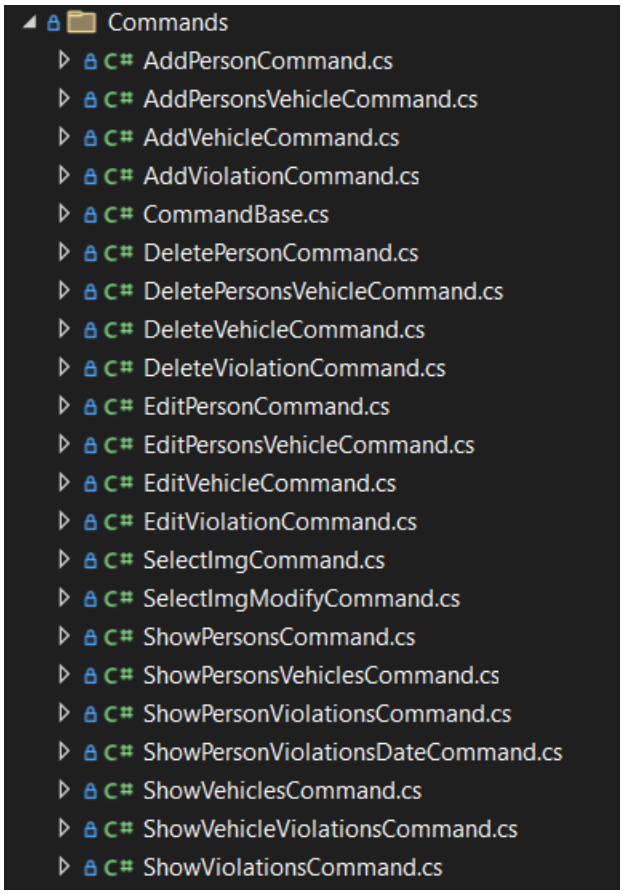
یک ویو مادل (MainViewModel) هم برای MainWindow نیاز داشتیم زیرا ما در صفحه اصلی عملیات Navigation را انجام میدهیم و نیاز بود که در این کلاس یک NavigationStore نگهداری کنیم که وظیفه آن نگهداری ویو مادل فعلی است که با تغییر آن ویو نیز تغییر کند. همچنین نیاز بود که با تغییر ویو مادل در نویگیشن استور ایونت OnPeropertyChanged برای پروپرتی ویومادل کنونی فراخوانی شود که با اینکار تغییرات در صفحه نیز انجام گردد.

یک ویو مادل نیاز داشتیم (TableViewModel) که همه اعضایی که باید در جدول نمایش داده شوند از آن نوع باشند زیرا قرار است هر دفعه لیست متفاوتی از آن ها در این جدول نمایش داده شود پس با

اینکار میتوانیم ما جدول را به لیستی از نوع پدرشان bind کنیم و هربار لیستی از نوع فرزندی که میخواهیم نمایش دهد در آن لیست بریزیم .

خب به همین ترتیب به تعداد نوع مدل هایی که قرار است داخل این جدول نمایش دهیم ViewModel میخواهیم که از Table ارث بری کند. که کل ویو مادل های دیگر میشوند که در Constractor خود مدل مربوطه خود را میگیرند و بر اساس داده های خود پروپرتی های مربوطه خود را دارند.

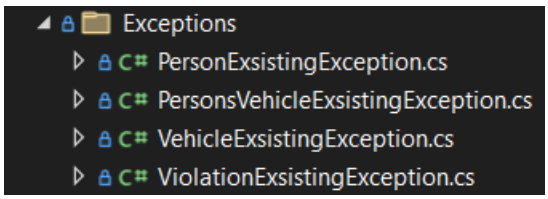
میرسیم به قسمت Command ها که در واقع عملیات های مربوط به باتن هارا انجام میدهند:



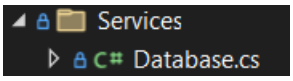
توضیح تک تک این قسمت ها زمانبر و خارج از توضیحات این فایل است برای همین بصورت کلی توضیح میدهیم:

در اینجا هم یک کلاس نیاز بود برای اینکه همه ابجکت ها از یک نوع شوند ، و ان هم کلاس CommandBase که خود از اینترفیس ICommand ارث بری میکند.

هر کلاس از این نوع باید دو کار بکند یکی مشخص کند آیا Executable است یا خیر که اگر اینچنین نبود باتن را دیزیبل کند، و اگر بود با کلیک بر روی آن کار بعدی که تابع Execute است را فراخوانی کند. در واقع دوتابع داریم که یکی کاری را انجام میدهد و دیگری اگه مقدارش true باشد باتن را اینیبل در غیر اینصورت دیزیبل میکند، لازم بذکر است برای اینکه با هر تغییر در پروپرتی ها بتوان این تابع را چک کرد باید تابعی به PropertyChanged ویومادل مربوطه اضافه کنیم که اگر آن پروپرتی مربوطه تغییر کرد تابع OnCanExecuteChanged نیز فراخوانی شود که این تابع مربوط به کلاس مادر است.



میرسیم به قسمت Exceptions که در این قسمت ارور های خود را پیاده سازی میکنیم که در هنگام مثلا اضافه کردن یک person با کد ملی ای که قبلا وجود داشته است این ارور throw شود، برای بهبود و بالابردن سطح برنامه میتوان ارور های بیشتری را پیشگیری و پیاده سازی کرد.



یک قسمت اضافه تر داریم برای سرویس ها :

که در اینجا فقط یک سرویس برای ساخت،ارتباط و کنترل دیتابیس نیاز داریم.

در این سرویس عملیات هایی چون ساخت دیتابیس TVIS درصورت نبود، ساخت جداول مربوطه درصورت نبود ، دریافت کل رکورد های همه جداول ، دریافت مقدار کل جریمه ها ، دریافت رکورد های مربوط به 3 کوئری خواسته شده ، اضافه کردن به هر جدول، آپدیت کردن اطلاعات هر رکورد ، حذف کردن هر رکورد و در نهایت دو تابع استاتیک کمکی برای تبدیل []Byte به BitmapImage و برعکس در هنگام خواندن و نوشتن در دیتابیس.

خب چون این قسمت قسمت اصلی این درس هست با جزییات بیشتری توضیح میدهیم:

در ابتدا برای سازنده ما اطلاعات مربوط به اتصال به دیتابیس را دریافت میکنیم سپس به دیتابیس master در آن پایگاه داده متصل میشویم و بعد با دستور زیر دیتابیس TVIS را درصورت نبود میسازیم:

```
IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = 'TVIS') BEGIN CREATE
DATABASE TVIS; END;
```

سپس به کانکشن قبلی را میبندیم و به دیتابیس TVIS کانکت میشویم و با دستورات زیر جداول مربوطه را میسازیم:

```
if not exists (select * from sysobjects where name='Persons' and xtype='U') create
table Persons (ID char(10),FirstName varchar(50),LastName varchar(50),Image
image,primary key (ID));
if not exists (select * from sysobjects where name='Vehicles' and xtype='U'
create table Vehicles (Pelak char(8),ConstractionYear smallint,Type tinyint,primary
key (Pelak));
if not exists (select * from sysobjects where name='Violations' and xtype='U')
create table Violations (ID char(10),Pelak char(8),Time Datetime,Type tinyint,Cost
int,primary key (ID,Pelak,Time),FOREIGN KEY (ID) REFERENCES Persons(ID),FOREIGN KEY
(Pelak) REFERENCES Vehicles(Pelak));
if not exists (select * from sysobjects where name='PersonsVehicles' and xtype='U')
create table PersonsVehicles (ID char(10),Pelak char(8),primary key
(ID,Pelak),FOREIGN KEY (ID) REFERENCES Persons(ID),FOREIGN KEY (Pelak) REFERENCES
Vehicles(Pelak));
```

نکته ای که در سازنده هست این است که ما کانکشن مان را در یک فیلد نگهداری میکنیم و هربار در توابع کلاس از ان فیلد استفاده میکنیم تا به دیتابیس متصل شویم.

در توابع InsertTo ما آبجکت مربوط به هر مدل را دریافت میکنیم و در صورت نال بودن هر پروپرتی از آن ابجکت ها مقدار DBNull و در غیر اینصورت مقدار خود انها را در پارامتر های مربوط به کوئری قرار میدهیم، کوئری های اضافه کردن برای همه توابع مربوطه به شکل زیر است:

```
INSERT INTO Persons values('{person.ID}',@FirstName,@LastName,@ImageArray)

INSERT INTO Vehicles values('{vehicle.Pelak}',@YearOfConstraction,@TypeOfVehicle)

INSERT INTO Violations
values('{violation.Person.ID}','{violation.Vehicle.Pelak}','{violation.ViolationDate
Time}',@TypeOfViolation,@Cost)

INSERT INTO PersonsVehicles
values('{personsVehicle.Person.ID}','{personsVehicle.Vehicle.Pelak}')
```

در بالا در کوئری ها میبینیم به 2 صورت مقادیر را در کوئری قرار دادیم یکی به شکل {value} و یکی به شکل @Property که تفاوت اینجا بود که وقتی میخواستیم مقدار نال را در مقادیری مثل نام که میتواند نال باشد قرار دهیم به شکل دوم امکان پذیر تر بود زیرا میتوانستیم بصورت DBNull مقدار را در پارامتر بریزیم.

در توابع DeleteFrom ما کلید های اصلی را به عنوان آرگومان های تابع دریافت میکردیم و با دادن آنها به کوئری های مربوطه رکورد را حذف میکردیم، کوئری های مربوطه را میتوانیم ببینیم:

```
DELETE FROM Persons WHERE ID='{ID}'

DELETE FROM Vehicles WHERE Pelak='{Pelak}'

DELETE FROM Violations WHERE ID='{ID}' and Pelak='{Pelak}' and Time='{time}'

DELETE FROM PersonsVehicles WHERE ID='{ID}' and Pelak='{Pelak}'
```

در توابع ModifyFrom مثل قسمت InsertTo عمل کردیم فقط اینجا آبجکت جدید که تغییر کرده بود را به تابع دادیم که آن هم تغییر را در دیتابیس اعمال کند:

```
UPDATE Persons SET FirstName=@FirstName, LastName=@LastName, Image=@ImageArray where
ID='{person.ID}';

UPDATE Vehicles SET Type=@TypeOfVehicle , ConstractionYear=@YearOfConstraction where
Pelak='{vehicle.Pelak}';

UPDATE Violations SET Type=@TypeOfViolation , Cost=@Cost where
ID='{violation.Person.ID}' and Pelak='{violation.Vehicle.Pelak}' and
Time='{violation.ViolationDateTime}';
```

نکته ای که هست برای PersonsVehicle ما تابع تغییر نداریم زیرا هر دو کلید هستندو صفت رابطه ندارد.

در توابع Get ما برای 4 جدول اصلی ما یک ابجکت از مدل Book مربوطه که داشتیم ساختیم و با Select همه اعضای این جداول همه اعضا را خواندیم و تک تک ابجکت مربوطه را از مدل خود ساختیم و به Book خود اضافه کردیم

در تابع GetSumOfCost ما مقدار کل جریمه هارو با کوئری `SELECT sum(Cost) FROM Violations` بدست میآوریم.

در تابع `GetPersonsViolations` ما یک تاپل که عضو اول آن BitmapImage برای برگرداندن عکس شخص مربوطه (کوئری 3) و عضو دوم آن لیستی از مدل PersonsViolation است که با کوئری زیر میگیریم :

ابتدا عکس شخص: `SELECT Image FROM Persons WHERE ID='{ID}'`

سپس اطلاعات مربوط به جدول خواسته شده:

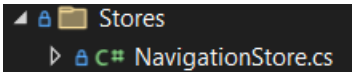
```
select Violations.Pelak,sum(cost) Violations,(select Pelak from Violations where
ID='{ID}' group by Pelak) as [Pelaks] where Violations.Pelak=Pelaks.Pelak group by
Violations.Pelak
```

در تابع GetPersonsViolationsTime (کوئری 4) ما با دریافت کد ملی و بازه زمانی مربوطه با کوئری زیر اطلاعات خواسته شده را در قالب لیستی ازین مدل بر میگردانیم:

```
select * from Violations where ID = '{ID}' and time between '{StartDate}' and
'{EndDate}'
```

در تابع GetVehiclesViolations (کوئری 5) ما با دریافت شماره پلاک با کوئری زیر اطلاعات خواسته شده را در قالب لیستی از مدل مربوطه بر میگردانیم:

```
select Firstname,Lastname,PersonsCost.Cost from Persons,(select
Persons.ID,sum(cost)as 'Cost' from Violations,Persons where Violations.ID=Persons.ID
and Pelak='{Pelak}' group by Persons.ID) as [PersonsCost] where
Persons.ID=PersonsCost.ID
```



قسمت بعدی قسمت Stores است که اینجا فقط یک مدل Store داریم: NavigationStore وظیفه نگهداری نویگیشن ها را دارد که در اینجا viewmodel کنونی را نگهداری میکند و با تغییر آن یک ایوند فراخوانی میکند که این ایونت بدین منظور است که در قسمت mainviewmodel توضیح دادیم که موجب فراخوانی OnPropertyChanged برای پروپرتی مربوط به ویو مادل کنونی شود و در نهایت در قسمت view ها تغییر کند.

میرسیم به تنها Window ما ، در این Window علاوه بر پیاده سازی منو و تغییر ویو مادل با کلیک هر ایتم منو ، نمایش ویو مربوطه نیز بر اساس ویومادل کنونی انجام میگردد. که کد های زیر این کار را انجام میدهد:

```
<Grid.Resources>
	<DataTemplate DataType="{x:Type vms:DashboardViewModel}">
		<views:Dashboard/>
	</DataTemplate>
	<DataTemplate DataType="{x:Type vms:InsertionViewModel}">
		<views:Insertion/>
	</DataTemplate>
	<DataTemplate DataType="{x:Type vms:ModifyViewModel}">
		<views:Modify/>
	</DataTemplate>
	<DataTemplate DataType="{x:Type vms:DeletetionViewModel}">
		<views:Deletetion/>
	</DataTemplate>
	<DataTemplate DataType="{x:Type vms:InfoViewModel}">
		<views:Info/>
	</DataTemplate>
</Grid.Resources>
<ContentControl Grid.Row="1" Grid.Column="1" Content="{Binding CurentViewModel}"/>
```

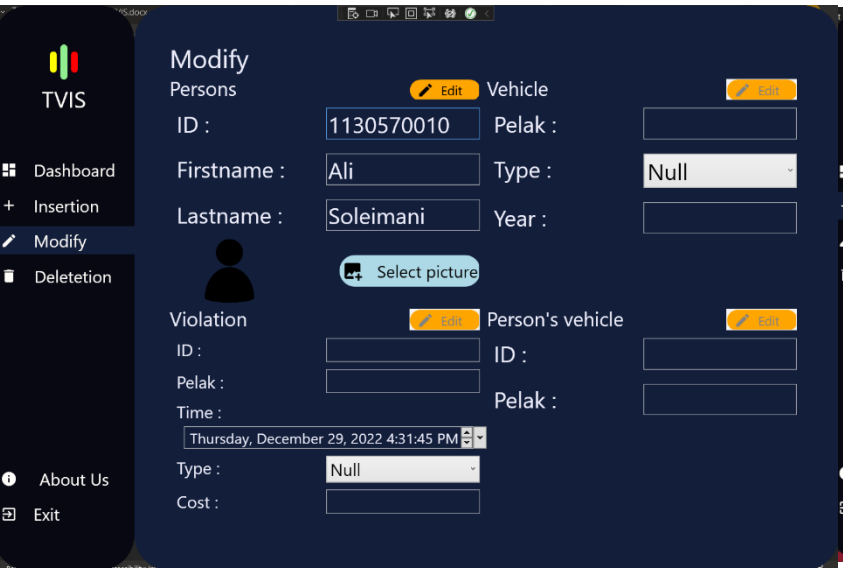
در واقع در اینجا ما مقدار کانتنت ContentControl را به ویو مادل کنونی بایند کردیم و برای این بایند مشخص کردیم که هر کدام از ویو مادل ها چه ویویی باشد.

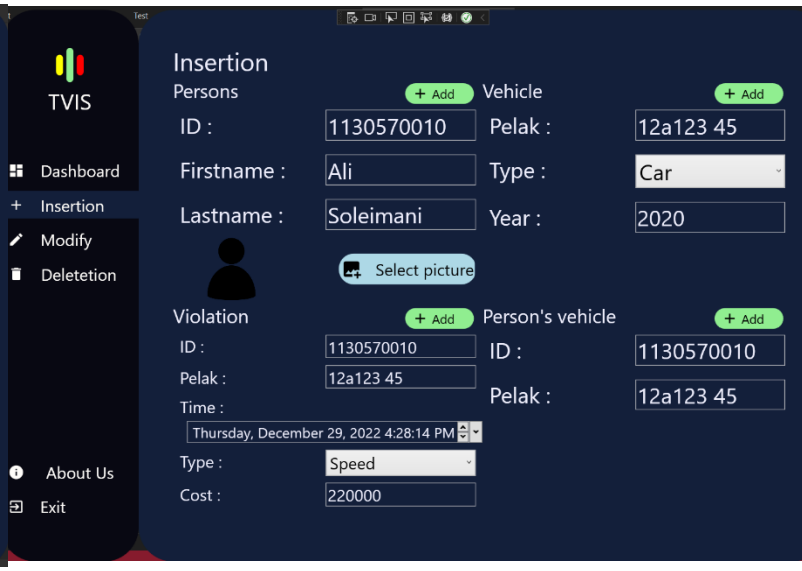
در قسمت منو فقط کافی بود با کلیک بر روی هر کدام از بخش ها مقدار ویومادل کنونی را از ابجکت navigationstore به ویومادلی که دوست داریم باشد تغییر دهیم.

در کلاس APP ما 3 فیلد داریم یکی برای نگهداری مدل کل برنامه ، یکی برای نگهداری مدل Database که البته نیاز ما فقط در مدل برنامه اصلی بود که میشد فیلدی بررای ان در نظر نگیریم، و

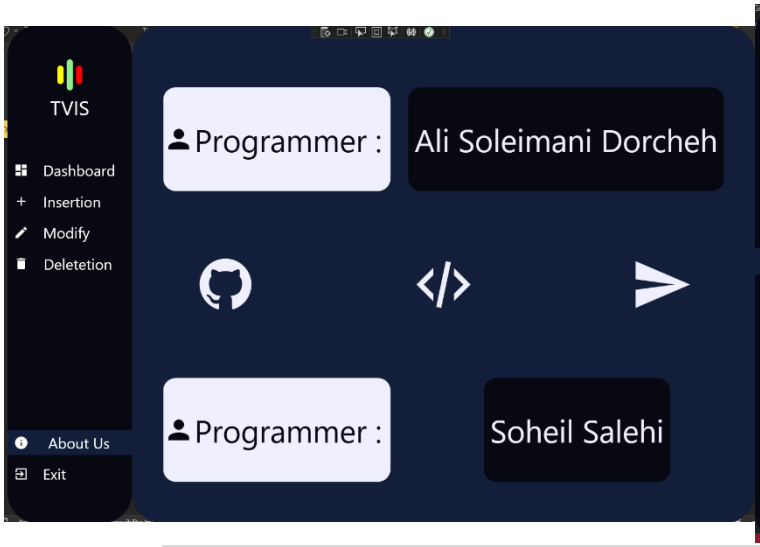
یکی هم برای نگهداری NavigationStore که عملیات تغییر صفحه و ویو مادل بر عهده این بخش بود.
اطلاعات کانکشن دیتا بیس هر بار که برنامه ران میشود از فایل App.Config خوانده میشود و انها برای سازنده دیتابیس ارسال میشود و دیتا بیس نیز برای سازنده مدل برنامه ارسال میشود و در ابتدا در ویو مادل کنونی در نویگیشن استور یک ابجکت از ویومادل داشبورد ریخته میشود و دیتا کانتکس ویندوز اصلی برنامه برابر با ویو مادل از mainviewmodel میشود که خود در کانستراکتور نویگیشن استور را دریافت میکند همچنین در کانستراکتور ویندوز اصلی نیز نویگیشن استور را میدهیم.

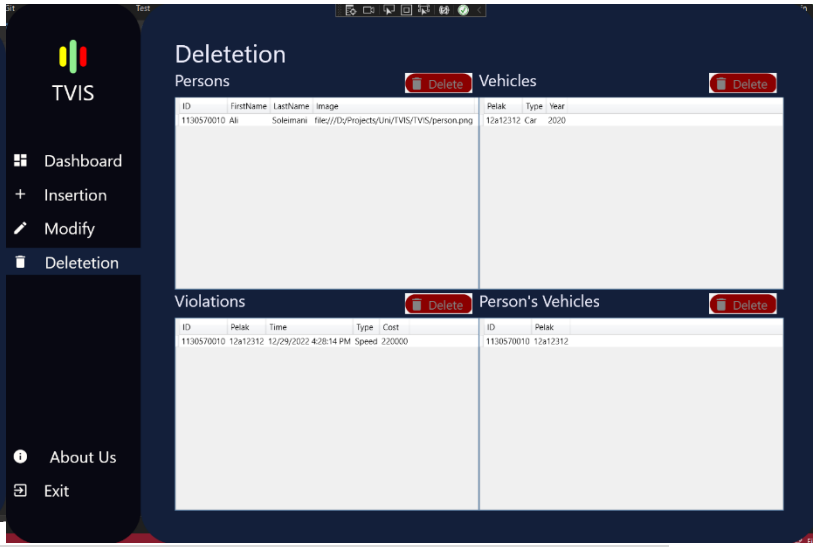
در ادامه کلیه صفحات برنامه را مشاهده میکنید:





نکته ای که درمورد صفحه modify هست ، در این قسمت با هربار وارد کردن یک کاراکتر در قسمت کلید های مربوط به هر بخش، چک میشود که اگر شخصی با آن ایدی وجود داشت در آن صورت بقیه اطلاعات آن شخص در تکست باکس ها و تصویر لود میشود و در این حالت باتن مربوطه Executable میشود ، حال میتوان هر قسمتی که میخواستیم را تغییر دهیم و ادیت را بزنیم.





کد کل پروژه در لینک زیر:

https://github.com/gameraliaz/TVIS