Hurtownia danych

# Projekt: Proces ETL

# Wstęp

W celu współdzielenia kodu pomiędzy członków zespołu, stworzone zostało repozytorium w systemie kontroli wesji GIT, który umożliwia korzystanie z repozytoriów na portalu GitHub.

Zdalne repozytorium projektu znajduję się pod adresem: <https://github.com/Cenarius1/HurtowniaDanych>. Projekt został wykonany w języku programowania Microsoft Visual C# z wykorzystaniem platformy .NET Framework w wersji 4.7.2. Na potrzeby projektu wybrano platformę bazodanową MS SQL Server w wersji 13.0.

# Technologie

W projekcie zastosowano technologie ściśle powiązane z wybranym językiem programowania, tj.

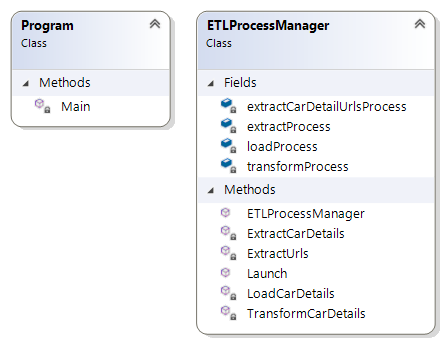
* *Windows Presentation Foundation (WPF)* – framework do tworzenia aplikacji desktopowych. Poprzez użycie znaczników XAML możemy tworzyć wygląd aplikacji, dodając w tle kod który odpowiada za ich zachowanie.
* *Json.Net* – framework do serializacji i deserializacji formatu tekstowego Json. Z jego pomocą można w wydajny sposób dokonywać przekształceń obiektów.
* *Entity Framework* – jeden z dostępnych ORM (Object-Relational Mapping), przeznaczony do budowania aplikacji opartych o bazy danych. W projekcie zastosowano podejście ‘Code First’ co oznacza tworzenie bazy danych za pomocą odwzorowania tabel poprzez klasy

Całościowe rozwiązanie nosi nazwę *HurtowniaDanych* i składa się z pięciu projektów:

* *DWH* – aplikacja konsolowa, która z użyciem odpowiednich parametrów uruchamia cały proces ETL jak również wybrane jego procesy
* *DWH.Domain* – projekt definiuje modele użyte w procesach ‘Extract’ oraz ‘Load’
* *DWH.ETL* – implementacja funkcjonalności procesu ‘Extract’ oraz ‘Transform’
* *DWH.GUI* – projekt interfejsu użytkownika wykonany w technologii WPF
* *DWH.Storage* – implementacja funkcjonalności procesu ‘Load’

# Projekt DWH

## Diagram klas



W skład projektu DWH wchodzą dwie klasy, *Program* oraz *ETLProcessManager*.

## Class Program

W klasie tej wyróżniamy główną metodę statyczną – *Main()*, która na wejściu przyjmuje argumenty podawane z linii poleceń. Następnie zostaje utworzony obiekt *etlManager* typu *ETLProcessManager*, który jest głównym elementem aplikacji. Na tak utworzonym obiekcie wywołujemy metodę *Launch()*.

## Class ETLProcessManager

W klasie zadeklarowane są cztery pola: *extractCarDetailUrlsProcess*, *extractProcess*, *transformProcess* oraz *loadProcess*. Nadanie im wartości następuje poprzez wywołanie konstruktora klasy, metody *ETLProcessManager()*.

Metoda *Launch()* na podstawie ustalonego adresu URL w zmiennej ‘baseUrl’, wywołuje metodę *ExtractUrls()* która przyjmuje ‘baseUrl’ jako parametr i zwraca listę adresów URL portalu OtoMoto. Kolejnym krokiem w procesie jest wyekstrahowanie danych z każdego ogłoszenia poprzez wywołanie metody *ExtractCarDetails()*. Na powstałej kolekcji obiektów, dokonujemy transformacji do docelowego modelu danych, który odpowiada schematowi bazy danych – metoda *TransformCarDetails()*. Finalnym etapem jest wykonanie metody *LoadCarDetails()*, która jak nazwa sugeruje służy do wprowadzenia danych do bazy.

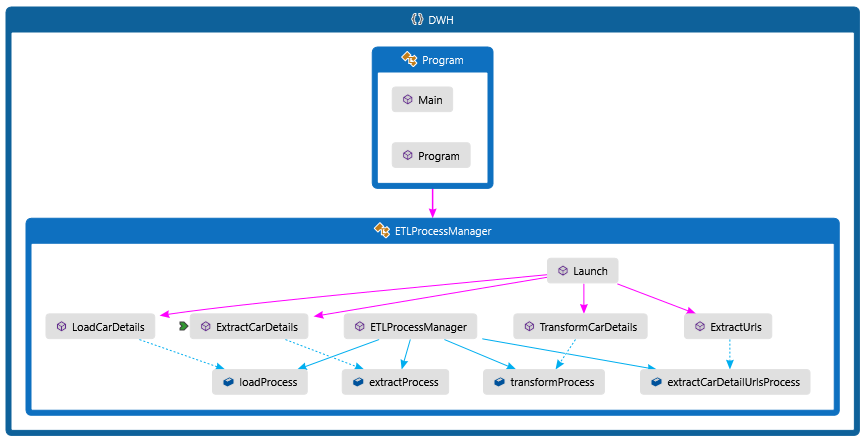
Metoda *ExtractUrls()* otrzymuję parametr wejściowy typu *string*. Poprzez wykorzystanie uprzednio zainicjowanego pola *extractCarDetailUrlsProcess,* wykonuje metodę *Process()* której wynikiem jest kolekcja ogłoszeń.

Metoda *ExtractCarDetails()* przyjmuje na wejściu listę adresów URL w postaci listy. Wewnątrz metody tworzona jest osobna lista, która zawiera listę obiektów typu *ExtractCarDetail* i jest wynikiem równoległego przetwarzania każdego pojedynczego adresu URL na ustalony model.

Metoda *TransformCarDetails()* otrzymuje jako parameter wejściowy listę obiektów typu *ExtractCarDetail*. Każdy pojedynczy obiekt odpowiada pojedynczemu ogłoszeniu portalu OtoMoto. Zainicjowana zostaje wynikowa lista obiektów typu *LoadCarDetail*, która zostaje wypełniona podczas procesu mapowania odpowiednich wartości do ustalonego modelu przechowywania danych.

Metoda *LoadCarDetails()* akceptuje na wejściu listę obiektów typu *LoadCarDetail*, a następnie wywołuje metodę *Process()* pola *loadProcess.* W przypadku poprawnego załadowania danych zostaje zwrócony komunikat treści ‘ETL Process finished succesfully’, w scenariuszu negatywnym komunikat ‘ETC Process encountered problems, check console logs for details.’

## Mapa zależności projektu DWH



# Projekt DWH.Domain

## Abstract Class Entity

W klasie abstrakcyjnej chodzi o wspólną implementację dla dziedziczonych klas. Zdefiniowane właściwości tworzą unikatowe parametry odróżnienia danych, tj:

* Id – numer ogłoszenia nadawany w procesie extract
* RowGuid – identyfikator rekordu tabeli typu GUID, wartość automatycznie ustawiana
* InsertedDttm – znacznik czasu użyty w procesie load

## Class LoadCarDetail

Klasa dziedziczy klasę abstrakcyjną *Entity*, w której skład wchodzą właściwości charakteryzujące pojedyncze ogłoszenie serwisu OtoMoto. Jest to kompletna lista elementów zawierająca podstawowe informacje ogłoszenia takie jak tytuł (‘*’Title‘’*), markę pojazdu (*‘’Mark’*’) czy cenę (*‘’PriceRaw’’*), jak również pełną listę wyposażenia pojazdu, tj. asystent parkowania (*‘’ParkAssist’’*), złącze IsoFix (*‘’Isofix’’*) – łącznie 93 właściwości.

## Class ExtractCarDetails

Klasa głównie składa się z skróconej deklaracji właściwości, które zapewniają dostęp do pól klasy posługując się akcesoriami get i set. Celem tej klasy jest mapowanie parsowanego łańcucha JSON do obiektu, który zostanie poddany kolejno transformacji. Mapowanie odbywa się przez ustawienie atrybutu *JsonProperty* dla każdej właściwości, np.

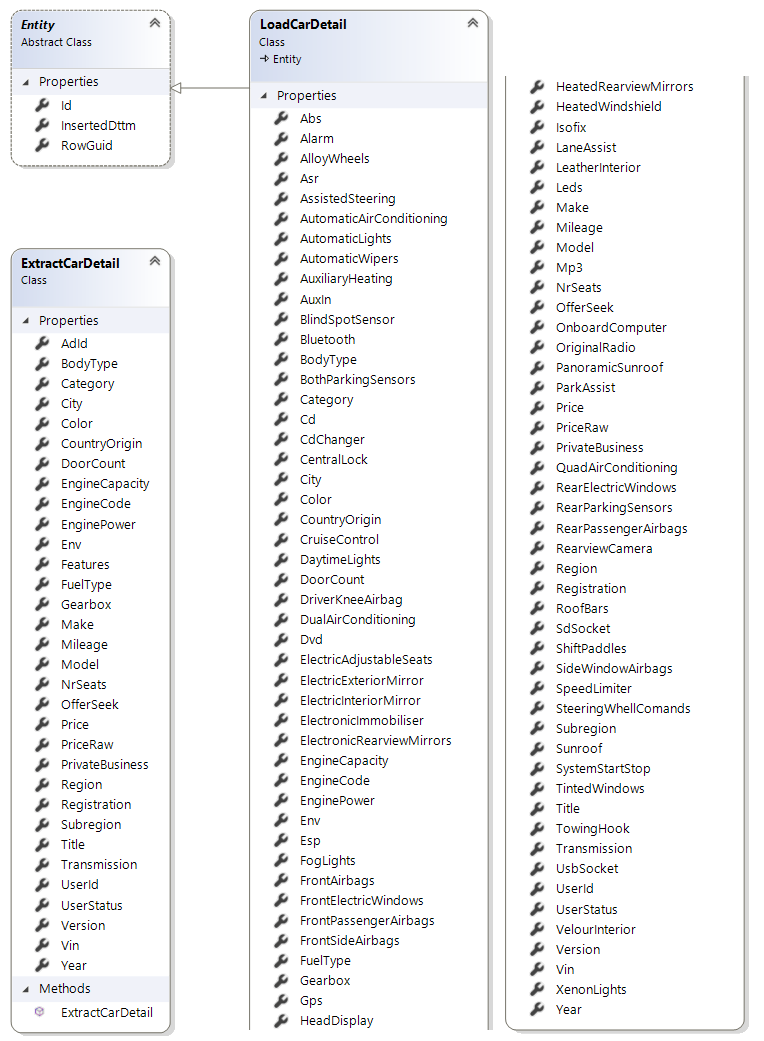
*JsonProperty(‘’title’’) 🡪 string Title*

*JsonProperty(‘’engine\_code’’) 🡪 string EngineCode*

W celu identyfikacji każdego utworzonego obiektu, użyta została właściwość *AdId* – jest to unikalny numer pojedynczego ogłoszenia serwisu.

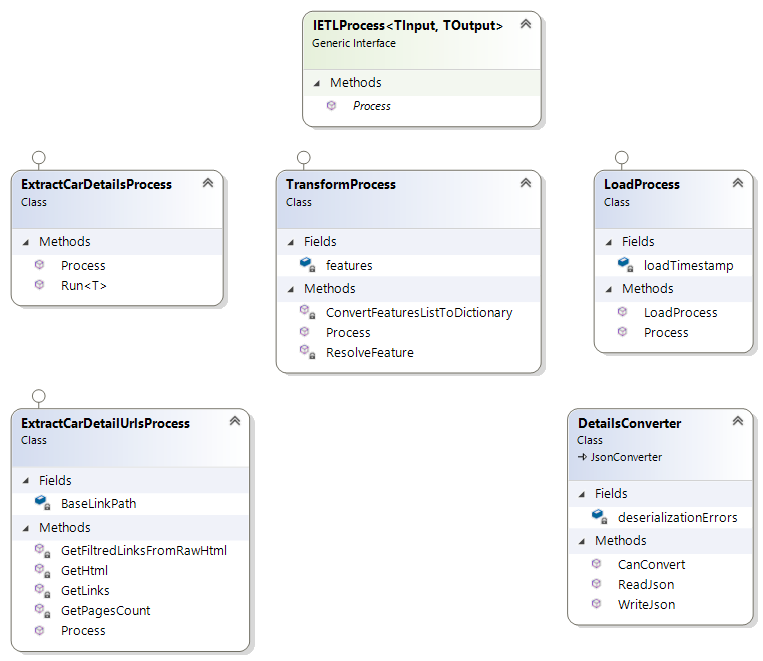
Użyty konstruktor ExtractCarDetails inicjują pustą kolekcję *Features*, która zostaje wypełniona elementami wyposażenia auta w trakcie parsowania.

## Diagram klas



# Projekt DWH.ETL

## Diagram klas



## Interface IETLProcess<TInput, TOutput>

Deklaruje składową metodę *Process()* wspomagającą proces pozyskiwania, przekształcenia oraz załadowania danych do bazy danych.

## Class ExtractCarDetailUrlsProcess

Klasa implementuje interfejs *IETLProcess* i jej głównym zadaniem jest pozyskanie wszystkich ogłoszeń serwisu OtoMoto dla wybranej marki pojazdu. Składa się z pola *BaseLinkPath* oraz metod *Process()*, *GetLinks()*, *GetPagesCount()*, *GetFilteredLinksFromRawHtml()* oraz *GetHtml()*.

Metoda *Process()* przyjmuje jako parametr adres URL, który zawiera w sobie wskaźnik do marki pojazdu. Na jego podstawie oraz ustalonej granicznej wartości która wynosi 10, utworzona zostaje kolekcja wszystkich ogłoszeń marki do 10 podstrony serwisu (*paginacja*). Każde ogłoszenie jest adresem URL.

Metoda *GetLinks()* na podstawie wywołania metody *GetHtml()* oraz *GetFilteredLinksFromRawHtml()* zwraca listę adresów URL dla pojedynczej podstrony

Metoda *GetPagesCount()* zwraca maksymalną ilość podstron ogłoszeń danej marki.

Metoda *GetFilteredLinksFromRawHtml()* otrzymuje na wejściu dokument HTML w formie tekstowej. Parsując treść, która stanowi pojedynczą stronę wraz z kilkoma ogłoszeniami, pozyskuje adres URL ogłoszenia każdego pojazdu.

Metoda *GetHtml()* na podstawie otrzymanego parametru *pageLink*, nawiązuje połączenie ‘HttpWebRequest’ i zwraca całą zawartość strony HTML w formie tekstowej.

## Class ExtractCarDetailsProcess

Klasa implementuje interfejs *IETLProcess* i odpowiada za pozyskanie kluczowych danych które jednoznacznie określają oraz opisują pojazd.

Metoda *Process()* otrzymuje w parametrze adres URL ogłoszenia – pojedynczy pojazd. Następuje pozyskanie treści dokumentu HTML, procesowanie drzewa znaczników i wyciągnięcie zagnieżdżonego obiektu JSON z danymi pojazdu.

Metoda *Run()* została utworzona w celu ponowienia procesu pozyskania treści HTML.

## Class DetailsConverter

Klasa dziedziczy po klasie *JsonConverter* i napisuje metody *CanConvert()*, *ReadJson()* oraz *WriteJson()*. Jej głównym zadaniem jest serializacja obiektu JSON (dane pojazdu w ogłoszeniu) do obiektu *ExtractCarDetail*.

Metoda *CanConvert()* determinuje czy obiekt może zostać poddany dalszej konwersji.

Metoda *ReadJson()* zawiera ustawienia niezbędne w procesie serializacji, jak również tworzy listę napotkanych błędów (np. nowa właściwość nie uwzględniona w modelu) podczas konwersji do obiektu *ExtractCarDetail*.

## Class TranformProcess

Klasa implementuje interfejs *IELTProcess* i jest odpowiedzialna za transformację obiektu *ExtractCarDetail* (obiekt posiada dane z ogłoszenia) do obiektu *LoadCarDetail* (obiekt który zostaje zapisany do bazy).

Metoda *Process()* otrzymuje ciąg znaków w formacie Json, który zostaje zamieniony w obiekt *ExtractCarDetail.* Kolejnym etapem jest odpowiednie przekształcenie utworzonego obiektu do docelowego typu *LoadCarDetail*. Ważnym krokiem jest proces transformacji każdego elementu wyposażenia pojazdu do typu bool – wyróżniamy ponad 70 różnych elementów.

Metoda *ResolveFeature()* służy do identyfikacji wyposażenia w procesowanym pojeździe.

## Class LoadProcess

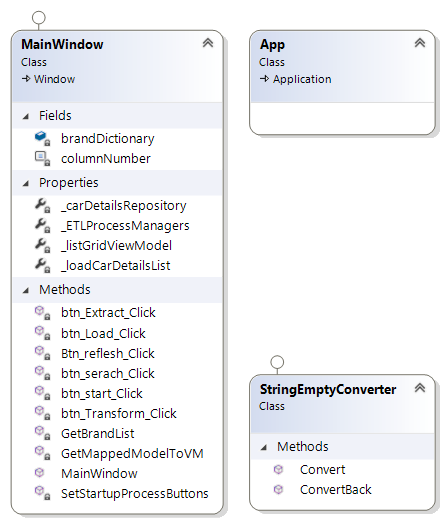
Klasa implementuje interfejs *IETLProcess* i jest wykorzystywana do wstawiana nowych ogłoszeń do bazy danych.

Konstruktor klasy inicjuje pole *loadTimestamp*, nadając mu wartość bieżącej daty oraz czasu. Znacznik jest użyty w procesie wpisania ogłoszeń.

Metoda *Process()* przyjmuje jako parametr listę ogłoszeń które zostały poddane uprzednio transformacji do obiektu *LoadCarDetail*. Każdemu obiektowi zostaje przypisana znacznik czasu i następuje wpis do bazy. W przypadku wystąpienia błędu, odpowiednia treść komunikatu zostanie wyświetlona w konsoli.

# Projekt DWH.GUI

## Diagram klas



## Class App (Xaml)

Klasa pozwala na uruchomienie głównego okna *MainWindow.xaml*. Elementem root tego pliku jest element ‘Application’ – początek aplikacji. Użyty atrybut ‘StartupUri’ określa, które okno ma być otwarte w dalszym kroku. Domyślnie jest to plik *MainWindow.xaml.*

## Class MainWindow (Xaml)

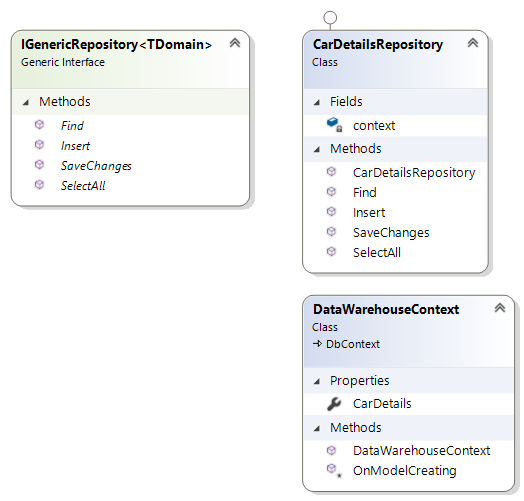
Klasa dziedziczy po klasie *System.Windows.Window*. Elementem root tego pliku jest element ‘Window’. Wewnątrz znajduje się element ‘Grid’, który tworzy zawartość okna poprzez zagnieżdżenie odpowiednich kontrolek UI tj. przyciski, tabela, pole tekstowe.

## Class StringEmptyConverter

Klasa implementuje interfejs *IValueConverter* i jest klasą pomocniczą. Zaimplementowana metoda *Covert()* pozwala na przypisanie w miejscu pustej wartości napisu ‘N/A’ (*not available*) oraz transformację występującej wartości do ‘Title Case’ notacji.

# Projekt DWH.Storage

## Diagram klas



## Interface IGenericRepository<TDomain>

Deklaruje składowe metody użyte do współpracy z funkcjami które udostępnia Entity Framework.

## Class CarDetailsRepository

Klasa implementuje interfejs *IGenericRepository* typu *LoadCarDetail*. Konstruktor *CarDetailsRepository* inicjuje pole *context*, które jest obiektem bazy.

Metoda *Insert()* jest odpowiedzialna za dodanie nowego ogłoszenia do bazy MS SQL. Przed dodaniem nowego wpisu, weryfikowana jest obecność ogłoszenia w bazie. W przypadku pojawienia się duplikatu informacji, ogłoszenie zostaje zingnorowane.

Metoda *SelectAll()* służy do pobrania wszystkich rekordów tabeli ‘CarDetails’. Otrzymane dane tworzą kolekcję typu *LoadCarDetail*, co pozwala w łatwy sposób na iterację wszystkich ogłoszeń.

Metoda *Find()* pozwala za pomocą parametru wejściowego *prediction* na zwrócenie podzbioru ogłoszeń według określonego kryterium.

Metoda *SaveChanges()* zapisuje zmiany które zostały dokonane na obiekcie bazy – *context*.

## Class DataWarehouseContext

Klasa dziedziczy po klasie *DBContext* i utworzone w niej obiekty odpowiadają fizycznej bazie danych. Konstruktor *DataWarehouseContext()*, zawiera w sobie właściwość z klasy bazowej. Parametr *base* w konstruktorze klasy nawiązuje połączenie z bazą danych o nazwie ETL\_DWH. Ciąg połączenia jest ustawiony w pliku *App.config* projektu DWH.

Właściwość *CarDetails* typu DbSet<LoadCarDetail> reprezentuje tabelę ‘dbo.CarDetails’ w bazie. Pozwala na wykonanie wszystkich operacji CRUD na zdefiniowanych modelach.

Projekt ‘DWH.Storage’ zawiera również katalog Migrations. Jego zawartość jest automatycznie generowana po uprzednim włączeniu migracji komendą „Enable-Migrations”. Następnie pozostaje wpisać „Update-Database” aby utworzyć bazę danych o nazwie ETL\_DWH w lokalnej instancji SQL Server.

# Źródła

1. WPF 🡪 https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2017
2. Json.Net 🡪 https://www.newtonsoft.com/json
3. Entity Framework 🡪 http://www.entityframeworktutorial.net/
4. HTML Agility Pack 🡪 https://html-agility-pack.net