Projekt zaliczeniowy

Proces ETL

Grupa projektowa: Projekt24, WZISN2-1112

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Imię | Nazwisko | Numer albumu | Grupa dziekańska | Wkład w prace nad projektem[[1]](#footnote-1) | Udział procentowy |
| Mikołaj | Męderski | 217067 | WZISN2-1112 | Stworzenie API, pobieranie danych z OTODOM, transformacja wyżej wymienionych danych, oczyszczanie dysku po całym procesie, eksport danych do “.csv” i “.txt”, praca przy dokumentacji i instrukcji użycia. | 33% |
| Krzysztof | Kowalik | 216939 | WZISN2-1112 | Integracja z API po stronie frontendu, wyświetlanie danych w przeglądarce, praca przy dokumentacji i instrukcji użycia. | 33% |
| Tomasz | Nowakowski | 216907 | WZISN2-1112 | Obsługa bazy danych, wczytywanie danych do bazy i ich obróbka, praca przy dokumentacji i instrukcji użycia. | 33% |

|  |
| --- |
| \_\_/70 pkt |

# Dokumentacja

Celem projektu jest zaprojektowanie i stworzenie aplikacji pobierającej, konwertującej i zapisującej dane pobrane z wskazanej strony internetowej. Docelową stroną internetową, z której scraper pobierze dane jest otoDom (<https://www.otodom.pl/> ).

# Ogólne informacje:

Aplikacja ma za zadanie pobrać dane, przekonwertować je do czytelnego dla niej formatu i zapisać je w zdefiniowanej wcześniej bazie danych. Proces powinien być automatyczny, podzielony odpowiednio na 3 fazy. Faza pierwsza to pobranie interesujących użytkownika danych, w fazie drugiej dane pobrane w fazie pierwszej poddawane są obróbce i sprowadzane do stanu pozwalającego zapisać je do bazy danych, faza trzecia to eliminacja powtarzających się danych oraz ich zapis do bazy danych. Aplikacja powinna generować raporty przy każdej fazie i zależności od preferencji użytkownika powinna wykonać cały cykl pracy i wygenerować końcowy raport, lub wykonywać następujące po sobie fazy tylko po uprzednim zatwierdzeniu ich przez użytkownika oraz generować raport przy końcu każdej fazy.

# Wymagania funkcjonalne:

## Opis ogólny

Aplikacja powinna składać się z trzech modułów opisanych szczegółowo poniżej.

Pierwszy moduł wybiera stronę internetową, z której to pobierze dane i przygotuje je do następnego kroku. Wygeneruje on też raport ile danych udało mu się pobrać. Drugi moduł podda pobrane dane obróbce w celu konwersji ich do docelowego formatu, dzięki któremu możliwe będzie wprowadzenie danych do bazy danych. Moduł wygeneruje również raport ile danych uległo konwersji. Trzeci moduł usunie powtarzające się dane i zapisze pozostałe dane w bazie danych oraz wygeneruje końcowy raport oraz zestawienie wszystkich trzech raportów.

## Użyte technologie

Aplikacja została podzielona na trzy części technologiczne. Pierwsza z nich odpowiada za interfejs użytkownika, napisana jest przy użyciu technologii frontend’owych (JavaScript, HTML5, CSS3). Druga jest częścią wykonawczą. Jest to API napisane przy użyciu języka C#. Odpowiada za pobieranie danych ze strony OTODOM, transformuje dane na docelowy format, oraz komunikuje się z bazą danych. Ostatnia - trzecia to sama baza danych.

Technologie użyte na froncie

Aplikacja po stronie frontend’u została stworzona w JavaScript, HTML5 oraz CSS3. W celu optymalizacji oparta jest na bibliotece ReactJS w wersji 16.12.0. W celu budowania aplikacji w wersji produkcyjnej oraz developerskiej używamy WebPack 4.41.0 pozwalający tworzyć paczki JavaScript złożone z reużywalnych i łatwo zarządzalnych modułów, możliwych do odczytania przez przeglądarkę dopiero po ich skompilowaniu w jeden plik. Kolejną technologią używa w projekcie jest Babel 7.6.0 pozwalający pisać kod używając najnowszych feature’ow dodanych miedzy innymi w specyfikacji ES6 oraz nowszych. Kod jest kompliowany do standardów możliwych do odczytania nawet przez starsze przeglądarki. Do wykonania interfejsu użytkownika używamy Material-ui jednej z najpopularniejszych bibliotek UI dedykowanych dla ReactJS. Dodatkowo biblioteka material-ui pozawala na użycie material-table, która zapewnia obsługę wyświetlania tabel w przyjazny dla użytkownika sposób.

Technologie użyte w API

Aplikacja została napisana w języku C# 6.0 i używa .NET Framework w wersji 4.6.1. Głowną technologia odpowiadającą za komunikacje miedzy interfejsem użytkownika, a API jest ASP.NET Web API w wersji 2.0. Jest to technologia, która używa protokołu http, a dane wysyłane są w formacie JSON. W celu wykonywania operacji scrappingu, użyta została paczka Html Agility Pack (HAP) w wersji 1.11.17. Pomaga ona wyciągać potrzebne dane z dokumentu HTML zawierającego kod strony OTODOM. Po udanej transformacji danych, są one zapisywane do pliku .json, w tym celu użyta została paczka Newtonsof.Json w wersji 12.0.3. Pomaga ona serializować, oraz deserializować obiekty języka C#. Ostatnią technologią użytą w tej części aplikacji jest paczka CsvHelper w wersji 12.2.2, która pozwala eksportować obiekty języka C# do plików .csv.

Technologie użyte w bazie danych

Skrypty obsługujące bazę danych zostały napisane w języku Transact-SQL. Do poprawnego działania skryptów wymagany jest Microsoft SQL Server for Developer Edition 2019. Jest to dostarczany przez Microsoft darmowy system zarządzania bazą danych, który jak języka zapytań używa T-SQL. Dane dostarczane przez aplikację zapisywane są do bazy danych przy pomocy narzędzia BCP (bulk copy), używane jest ono też do zapisu danych z bazy do pliku. Jest to jeden z najszybszych sposobów przenoszenia plików między bazą danych a systemem plików.

## Instalacja aplikacji

Frontend

W celu zainstalowania części front-end’owej należy:

* Zainstalować NodeJs z podanego linku w najnowszej wersji rekomendowanej na oficjalnej stronie:

<https://nodejs.org/en/>

* Posiadać dowolny edytor tekstu - rekomendowany Visual Studio Code, który można pobrać z oficjalnej strony:

<https://code.visualstudio.com/>

* Korzystając z wiersza poleceń przechodzimy do katalogu:

*~\WebScrapper\webscrapper-frontend*

* Z wiersza poleceń uruchamiamy komendę

*npm install*

Po zakończeniu procesu instalacji wszystkie niezbędne do uruchomienia pakiety zostaną zainstalowane lokalnie na komputerze.

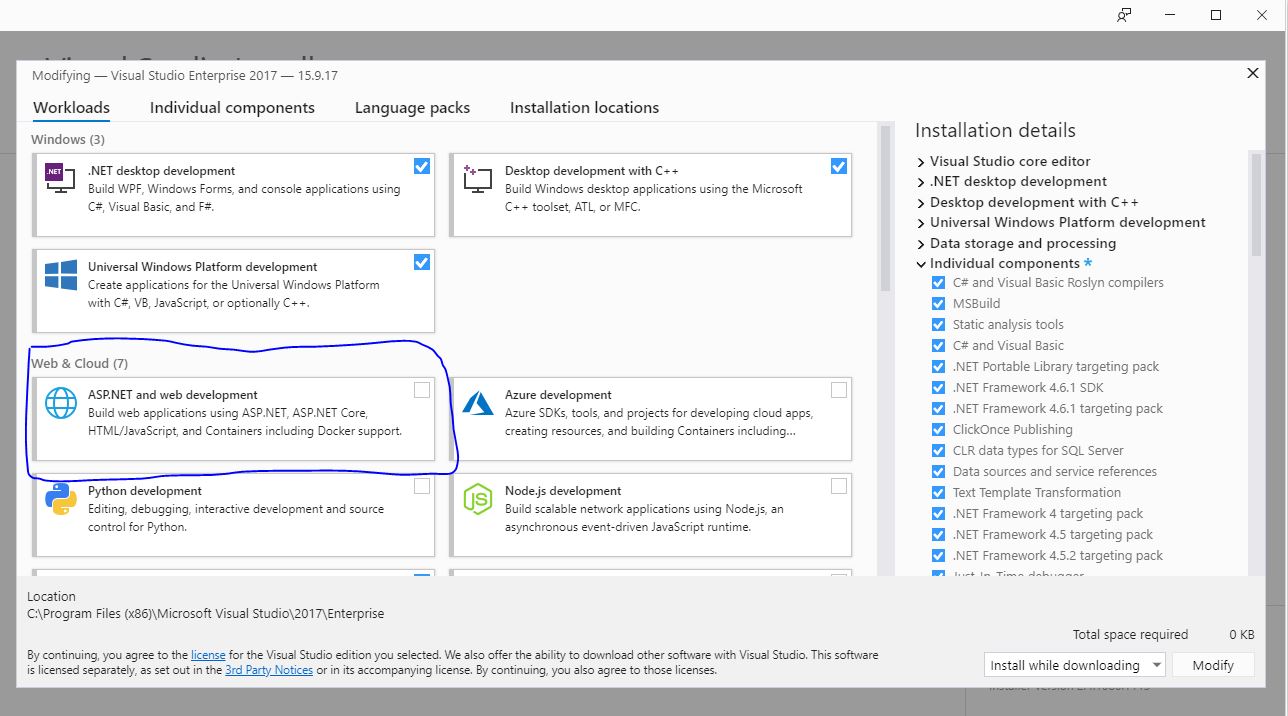
Backend

W celu zainstalowania API obsługującego front-end aplikacji należy:

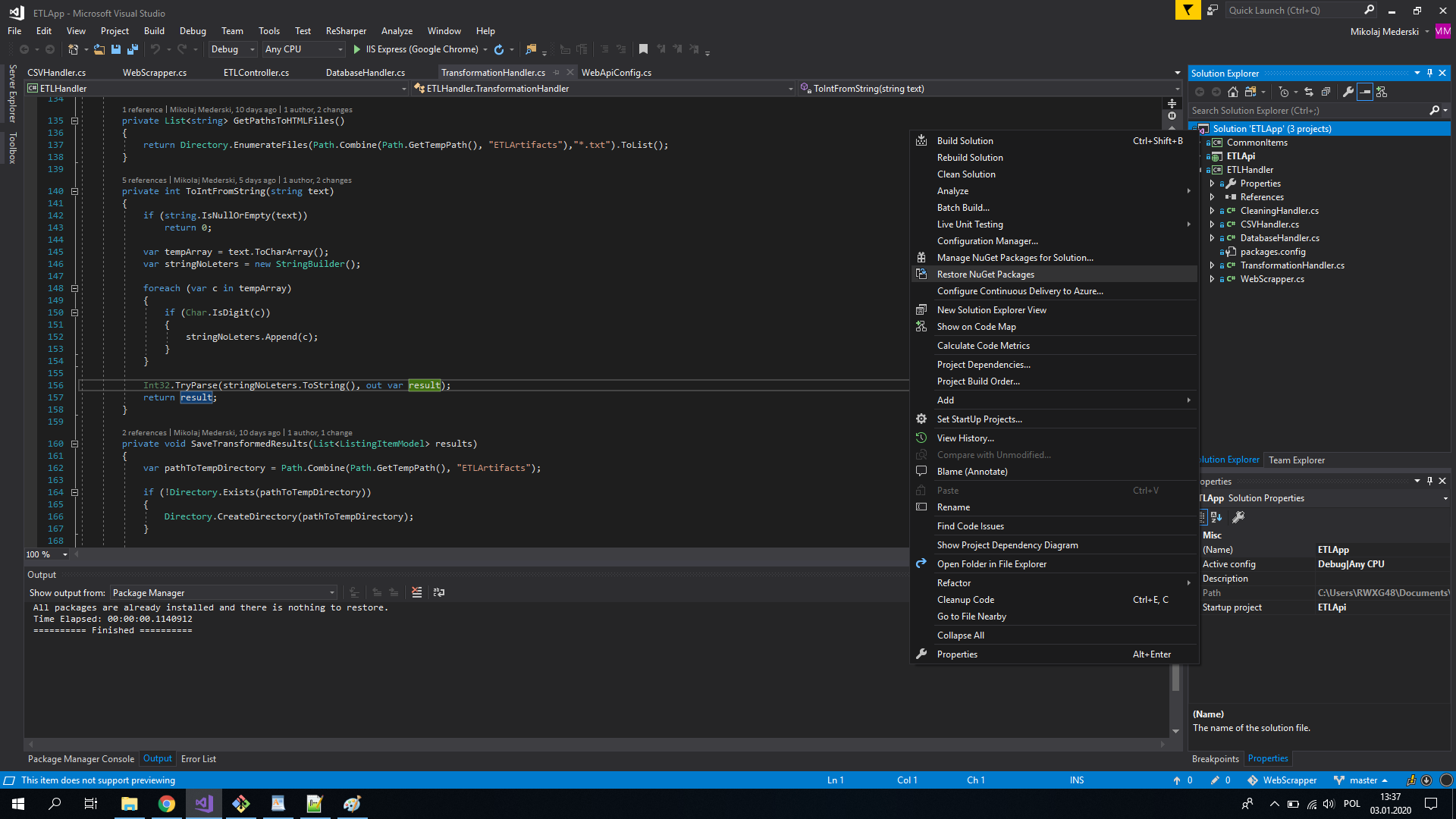
* Pobrać installer Visual Studio wchodząc w link:

<https://visualstudio.microsoft.com/pl/downloads/>

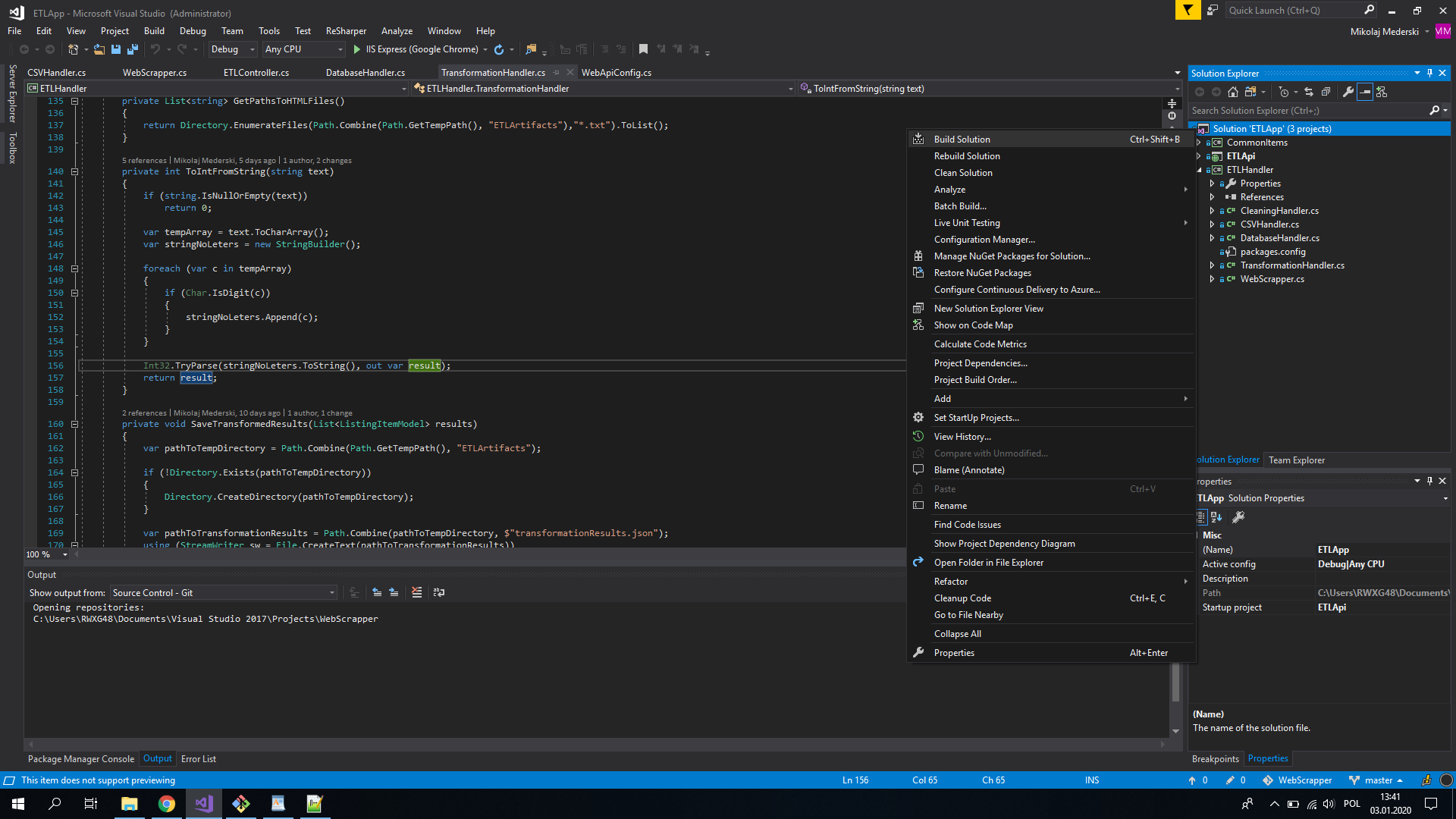
* Zainstalować Visual Studio 2017 wybierając dodatkowo ASP.NET and Web Development w zakładce Workload



* Otworzyć plik "ETLApp.sln" w programie Visual Studio 2017 wybierając kolejno w lewym górnym rogu ekranu File -> Open -> Project/Solution
* W oknie "Solution Explorer" kliknąć prawym przyciskiem myszy "Solution 'ELTApp'" i wybrać z menu opcję "Restore NuGet Packages"



* Ponownie kliknąć prawym przyciskiem myszy "Solution 'ELTApp'" i wybrać opcje Build Solution

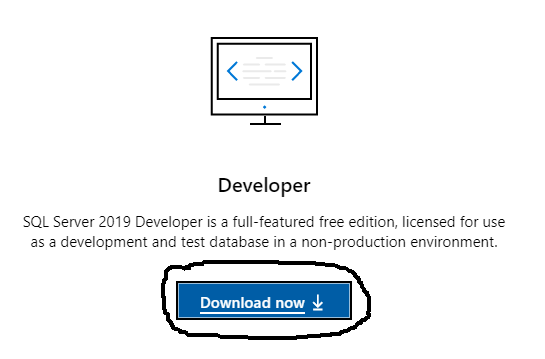


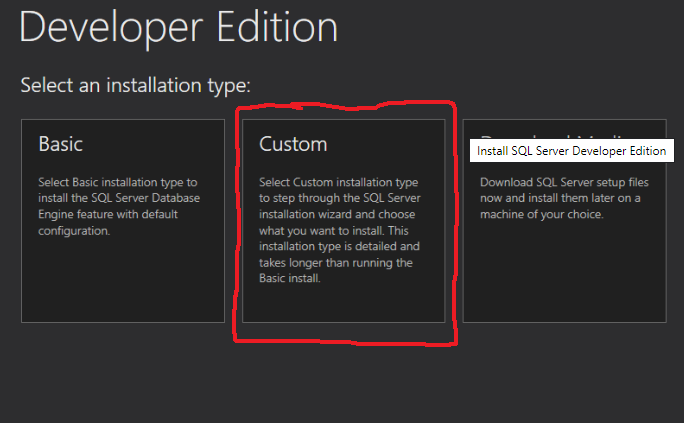
Baza danych

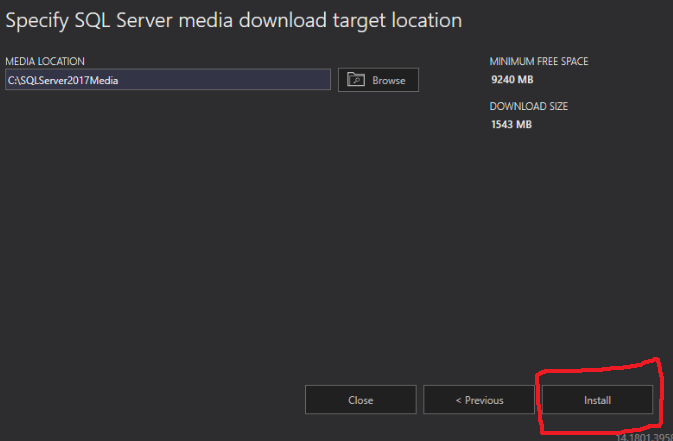
W celu Bazy danych należy:

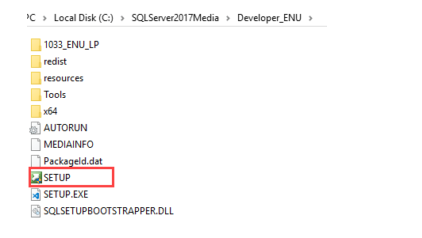
* Zainstalować Microsoft SQL Database z oficjalnej strony:

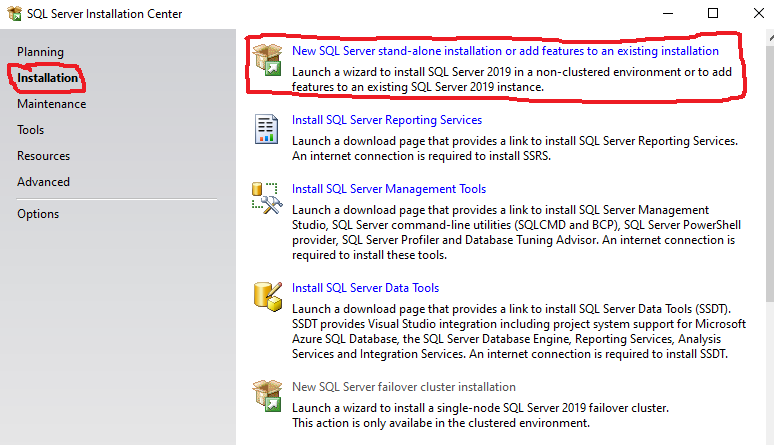
<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

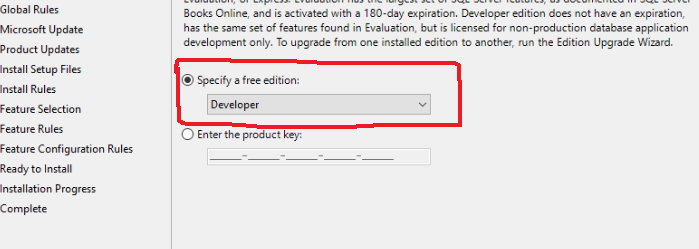


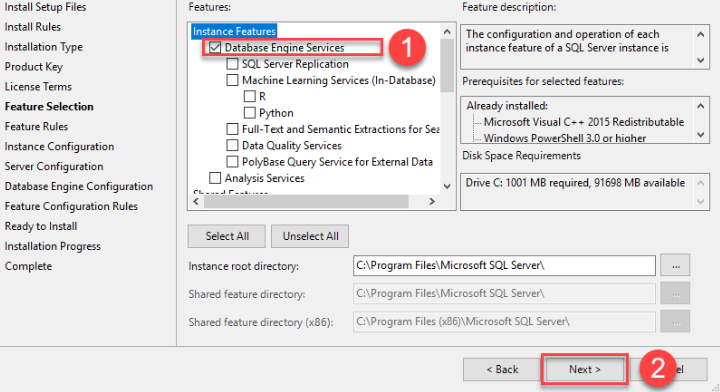


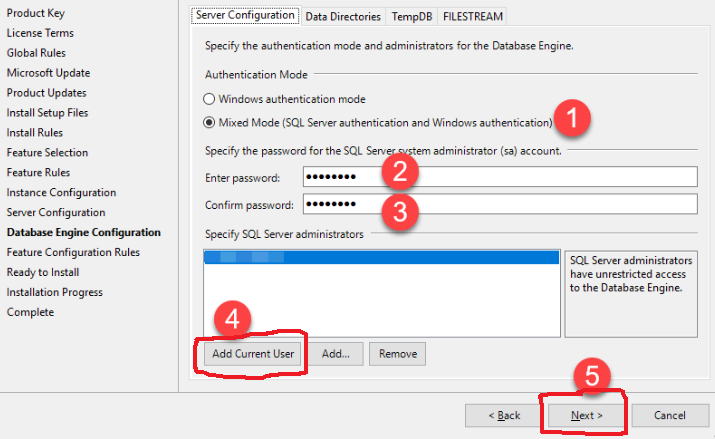






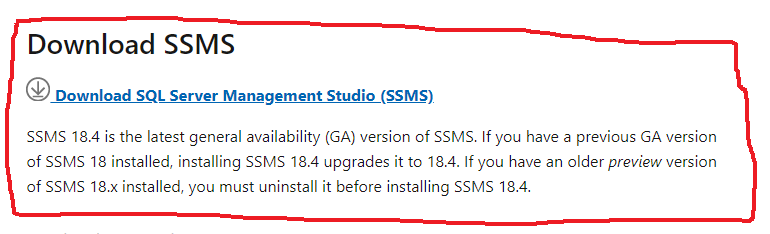


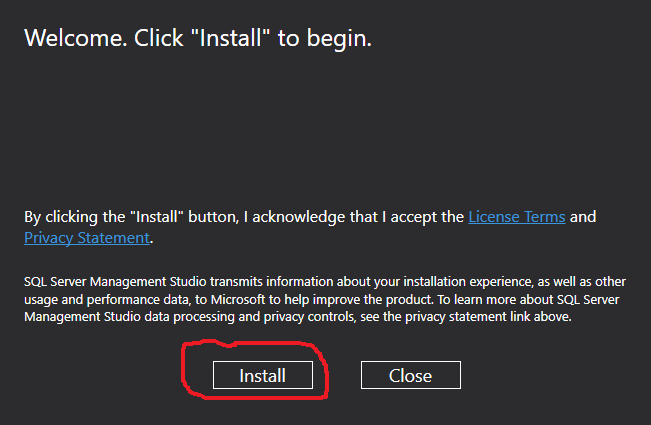




* Zainstalować SQL Server Management Studio (SSMS) z oficjalnej strony:

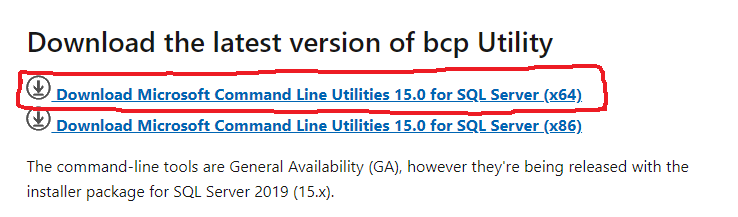
<https://docs.microsoft.com/pl-pl/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>





* Zainstalować BCP Utilities z oficjalnej storny:

<https://docs.microsoft.com/en-us/sql/tools/bcp-utility?view=sql-server-ver15>



* Dodać wyjątek do Firewall dla bazy danych:

<https://docs.microsoft.com/en-us/sql/sql-server/install/configure-the-windows-firewall-to-allow-sql-server-access?view=sql-server-ver15#to-add-a-program-exception-to-the-firewall-using-windows-firewall-with-advanced-security>

* Pliki w podanych lokalizacjach umieścić według wskazań plików README!!.txt:

"\WebScrapper\Data Base\Funkcje Bazy Danych\"

"\WebScrapper\Data Base\Tworzenie Tabeli Dane\"

# ***Dokładny opis modułów***

Moduł pierwszy - jest on odpowiedzialny za scrappowanie. Aktywowany jest po wysłaniu prośby przez interfejs użytkownika. Aby moduł mógł zadziałać poprawnie, wymagane jest podanie poprawnego adresu URL strony OTODOM, zawierającej wyniki wyszukiwania. Adres ten zapisywany jest jako pole w klasie scrappującej. Następnym krokiem jest wysłanie prośby do serwisu OTODOM, o kod źródłowy strony, do której prowadzi wcześniej wspomniany link URL. Po otrzymaniu odpowiedzi, kod ten przekazywany jest do modułu drugiego (opisanego niżej w tym dokumencie), w celu uzyskania linków URL do pojedynczych ofert, znajdujących się na stronie wyszukiwarki ogłoszeń. Po otrzymaniu listy linków, klasa scrappująca wysyła prośby o kody źródłowe konkretnych ogłoszeń. Próśb tych wysłanych jest tyle, ile ogłoszeń znajdowało się na stronie wyszukiwarki. Ostatnią funkcjonalnością tego modułu jest zapisanie otrzymanych wyników do plików tekstowych, znajdujących się w folderze C:\Users\USER\AppData\Local\Temp (gdzie USER to nazwa użytkownika komputera). Zapis ten odbywa się jedynie wtedy gdy użytkownik wybierze funkcje Extract. Podczas funkcji ETL nie zachodzi potrzeba zapisywania danych do zewnętrznych plików. Dane zapisane w plikach tekstowych mogą następnie zostać odczytane przez moduł drugi, podczas transformacji.

Moduł drugi - odpowiedzialny za transformacje. Posiada on dwie główne funkcjonalności:

* Transformacje danych na format, który pozwala na zapis do bazy danych
* Wyciąganie adresów URL poszczególnych ogłoszeń zawartych na stronie wyszukiwani serwisu OTODOM.

Pierwsza funkcjonalność działa na dwa sposoby, w zależności od ścieżki która wybrał użytkownik. Jeśli wybrana została ścieżka pojedynczych kroków (osobno Extract, osobno Transform i osobno Load) wtedy dane potrzebne do transformacji (kod HTML ogłoszeń) odczytywany jest z plików powstałych podczas działania modułu pierwszego. Jeśli wybrana została ścieżka automatyczna (wszystkie kroki ETL wykonywane są jeden po drugim, bez ingerencji użytkownika) dane do transformacji przekazywane są bezpośrednio z poprzedniego modułu.

Gdy moduł ma już dostęp do danych, kod źródłowy ogłoszeń jest odczytywany przez Html Agility Pack, a następnie wyciągane są z niego części kodu odpowiedzialne za część tytułową ogłoszenia, oraz część posiadającą dane szczegółowe (np. rodzaj ogrzewania, rok budowy budynku). Te dwie sekcje są następnie przekazywane do funkcji odpowiedzialnej za wyciągnięcie konkretnych wartości liczbowych, bądź informacji tekstowych i stworzenie obiektu C#, posiadającego interesujące nas pola.

Kończąc swe działanie, na podstawie ścieżki wybranej przez użytkownika moduł zapisuje dane po transformacji, jeśli użytkownik wybrał ścieżkę pojedynczych kroków. Dane zapisywane są w pliku o formacie „.json”.

Druga funkcjonalność wykorzystywana jest przez moduł pierwszy. Funkcja otrzymuje kod źródłowy strony, na której wylistowane są ogłoszenia. Następnie tworzy listę pojedynczych ogłoszeń i z każdego ogłoszenia wyciąga jego adres URL. Lista adresów zwracana jest do modułu pierwszego.

Moduł trzeci - odpowiedzialny za komunikację z bazą danych. Posiada on trzy główne funkcjonalności :

* wczytywanie pobranych danych do Bazy Danych
* sprawdzenie i eliminacja dublujących się danych

Dostęp do Bazy danych inicjalizowany jest przez Microsoft SQL Server Management, podłączając się do dostępnego servera. Następnie uruchamiając Tabela\_DANE.exe tworzona jest Baza Danych oraz tabela główna.

Pierwsza funkcjonalność działa wczytując dane z pliku transformationResults.json, który jest efektem transformacji ogłoszeń przez moduł drugi. Dane z pliku json zostają rozdzielone do odpowiednich kolumn w tabeli tymczasowej tworzonej w bazie danych.

Gdy wszystkie dane zostaną poprawnie załadowana do tabeli, następuje porównanie danych znajdujących się w tabeli głównej z danymi znajdującymi się w tabeli tymczasowej. Gdy sprawdzane dane okażą się identyczne, następuje ich pominięcie i przejście do następnych. Wszystkie unikalne dane zostają zapisane w tabeli głównej.

Moduł pomocniczy - wykonuje on operacje nie związane z procesem ETL. Zawiera on

* Eksport danych do pliku o formacie „.csv”
* Eksport pojedynczego ogłoszenia do pliku o formacie „.txt”
* Usuwanie plików powstałych podczas procesu ETL.

# Tabele klas i atrybutów

## Backend

### ListingItemModel

Atrybuty:

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| Title | string | Pole przechowujące tytuł ogłoszenia |
| Rooms | int | Pole przechowujące ilość pokoi w nieruchomości |
| Area | string | Pole przechowujące powierzchnie nieruchomości |
| Price | string | Pole przechowujące cenę nieruchomości |
| Bond | int | Pole przechowujące kaucję za nieruchomość |
| BuildingType | string | Pole przechowujące rodzaj nieruchomości (np. Blok) |
| Windows | string | Pole przechowujące typ okien w nieruchomości (np. plastikowe) |
| BuiltIn | int | Pole przechowujące rok budowy nieruchomości |
| HeatingType | string | Pole przechowujące rodzaj ogrzewania (np. miejskie) |
| Materials | string | Pole przechowujące rodzaj materiałów budulcowych |
| Floor | int | Pole przechowujące piętro na którym znajduje się nieruchomość |
| Address | string | Pole przechowujące lokalizacje nieruchomości |
| FloorsInBuilding | int | Pole przechowujące ilość pięter w budynku |

### CleaningHandler

Funkcje Składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| DeleteArtifacts | Void | Brak | Funkcja usuwająca pliki powstałe podczas procesu ETL |

### CSVHandler

Funkcje Składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| ExportToCSV | Void | List<ListingItemModel> - Lista ofert | Tworzy plik “.csv” z danymi, które zostały przekazane w parametrze |
| ExportSingleToTxt | Void | ListingItemModel - pojedyncza oferta | Tworzy plik “.txt” z danymi oferty przekazanej w parametrze |
| ValidateOfferTitle | string | string title | Zamienia znaki typu “,” na znak “\_”, w celu stworzenia nazwy pliku na podstawie tytułu oferty |
| GetPathToFile | string | string fileName | Zwraca ścieżkę, w której powstanie plik z wyeksportowanymi danymi |

### DatabaseHandler

Funkcje składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| Load | List<ListingItemModel> | Brak | Funkcja uruchamiająca skrypt zasilający bazę danych, następnie używająca metody GetAllOffersFromDB. |
| CleanDB | Void | Brak | Funkcja uruchamiająca skrypt czyszczący bazę danych. |
| GetAllOffersFromDB | List<ListingItemModel> | Brak | Funkcja wyciągająca wszystkie oferty z bazy danych |

### *TransformationHandler*

Atrybuty:

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| \_polishChars | static readonly char[] | przechowuje wszystkie polskie znaki |
| \_englishChars | static readonly char[] | przechowuje angielskie odpowiedniki polskich znaków |

Funkcje składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| GetListOfProducts | List<ListingItemModel> | bool needSave,  List<string> rawHtmlList | Zwraca listę ofert pod postacią obiektów ListingItemModel |
| GetAdsUrls | List<string> | string rawHtml | Zwraca wszystkie URLe ogłoszeń zawartych w HTMLu przekazanym w parametrze |
| GetOfferDetails | ListingItemModel | HtmlNode details  HtmlNode title | Tworzy obiekt ListingItemModel przypisując wartości pól na podstawie sekcji HTML przekazanych w parametrze |
| GetPathsToHTMLFiles | List<string> | Brak | Zwraca lokalizacje plików zawierających kod HTML ofert |
| ToIntFromString | int | string text | Przerabia string w którym znajdują się cyfry na int ignorując litery. Np “30 zł” -> “30” |
| SaveTransformedResults | Void | List<ListingItemModel> results | Tworzy plik formatu “.json” zawierający listę ofert w postaci ListingItemModel |
| RemoveDiacritics | string | string text | Zamienia wszystkie polskie znaki na ich angielskie odpowiedniki |

### WebScrapper

Atrybuty:

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| \_url | readonly string | Zawiera adres witryny OTODOM |
| \_httpClient | HttpClient | Instancja klasy HttpClient, wykorzystywana do komunikacji internetowej |
| \_transformer | TransformationHandler | Instancja klasy TransformationHandler opisanej powyżej |

Funkcje składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| KONSTRUKTOR:  WebScrapper | Brak | String url | Przypisuje wartosc pola \_url, oraz tworzy nowe instancje HttpClient i TransformationHandler |
| GetRawHtmls | List<string> | bool needSave | Pobiera kod źródłowy znajdujący się pod adresem z pola \_url, a nastepnie pobiera kod źródłowy pojedynczych ofert |
| SaveRawHtmlToText | Void | string result  int documentNumber | Tworzy plik o nazwie “rawHtmlX.txt” gdzie X to liczba ze zmiennej documentNumber. Zawartością pliku jest ciąg literalny result |

### ETLController

Funkcje składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| Get | IHttpActionResult | JsonBodyModel model | Wywołuje akcje Extract |
| Transform | IHttpActionResult | Brak | Wywołuje akcje Transform |
| Load | IHttpActionResult | Brak | Wywołuje akcje Load |
| FullETL | IHttpActionResult | JsonBodyModel model | Wywołuje po kolei akcje Extract, Transform, Load |
| ExportToCSV | IHttpActionResult | List<ListingItemModel> offers | Wywołuje eksport do “.csv” danych przekazanych w parametrze offers |
| CleanDb | IHttpActionResult | Brak | Wywołuje czyszczenie bazy danych |
| ExportSingleToTxt | IHttpActionResult | ListingItemModel offer | Wywołuje eksport do “.txt” oferty przekazanej w parametrze offers |
| GetAllRecords | IHttpActionResult | Brak | Wywołuje metodę wyciągającą całą zawartość bazy danych |

### JsonBodyModel

Atrybuty:

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| url\_adress | string | Adres witryny OTODOM |

## Frontend

Funkcje składowe:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nazwa** | **Wartość Zwracana** | **Parametry** | **Opis** |
| handleSubmit | IHttpActionRequest | Submit Event | Wysyłanie formularza z podanym adresem url, zapobieżenie domyślnej akcji przeglądarki po wysłaniu formularza. |
| postUrl | function | String url | Wysyła adres url do API w celu uruchomienia procesu Extract. |
| getAfterTransform | IHttpActionRequest | Brak | Uruchomienie metody odpowiedzialnej za pobieranie danych z domeny otodom. |
| fetchData | IHttpActionRequest | Brak | Uruchomienie procesu Transform, wyświetlenie statusu zapytania po wysłaniu zapytania. |
| getAfterLoad | IHttpActionRequest | Brak | Uruchomienie metody Load zwracającej dane otrzymane po scrapowanieu i wyświetlenie ich w przeglądarce. |
| fullETLProcess | IHttpActionRequest | Brak | Uruchomienie metod odpowiedzialnych za Export, Transform, Load automatycznie jedna po drugiej jeśli poprzenia zakończyła się sukcesem i wyświetlenie rezultatów. |
| exportToCsv | IHttpActionRequest | Brak | Export tabeli do pliku .csv. |
| exportSingleElement | IHttpActionRequest | rowData | Export pojedynczego wiersza z tabeli do pliku .txt. |
| cleadDb | IHttpActionRequest | Brak | Czyszczenie bazy danych |

## Baza danych

Atrybuty:

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Typ** | **Opis** |
| Title | NVARCHAR(255) | Kolumna przechowująca tytuł ogłoszenia |
| Rooms | INT | Kolumna przechowująca ilość pokoi w nieruchomości |
| Area | NVARCHAR(255) | Kolumna przechowująca powierzchnie nieruchomości |
| Price | NVARCHAR(255) | Kolumna przechowująca cenę nieruchomości |
| Bond | INT | Kolumna przechowująca kaucję za nieruchomość |
| BuildingType | NVARCHAR(255) | Kolumna przechowująca rodzaj nieruchomości (np. Blok) |
| Windows | NVARCHAR(255) | Kolumna przechowująca typ okien w nieruchomości (np. plastikowe) |
| BuiltIn | INT | Kolumna przechowująca rok budowy nieruchomości |
| HeatingType | NVARCHAR(255) | Kolumna przechowująca rodzaj ogrzewania (np. miejskie) |
| Materials | NVARCHAR(255) | Kolumna przechowująca rodzaj materiałów budulcowych |
| Floor | INT | Kolumna przechowująca piętro na którym znajduje się nieruchomość |
| Address | NVARCHAR(255) | Kolumna przechowująca lokalizacje nieruchomości |
| FloorsInBuilding | INT | Kolumna przechowująca ilość pięter w budynku |

Instrukcja obsługi aplikacji:

# Instrukcja uruchomienia aplikacji:

## Uruchomienie części front-end’owej:

Aby uruchomić wersie deweloperską aplikacji front-end’owej najlepiej skorzystać z wiersza poleceń ( domyślny dostępny w systemie operacyjnym Windows ). Powinniśmy znajdować się w katalogu aplikacji front-end’owej:

*~\WebScrapper\webscrapper-frontend*

Z tego miejsca uruchamiamy polecenie:

*npm start*

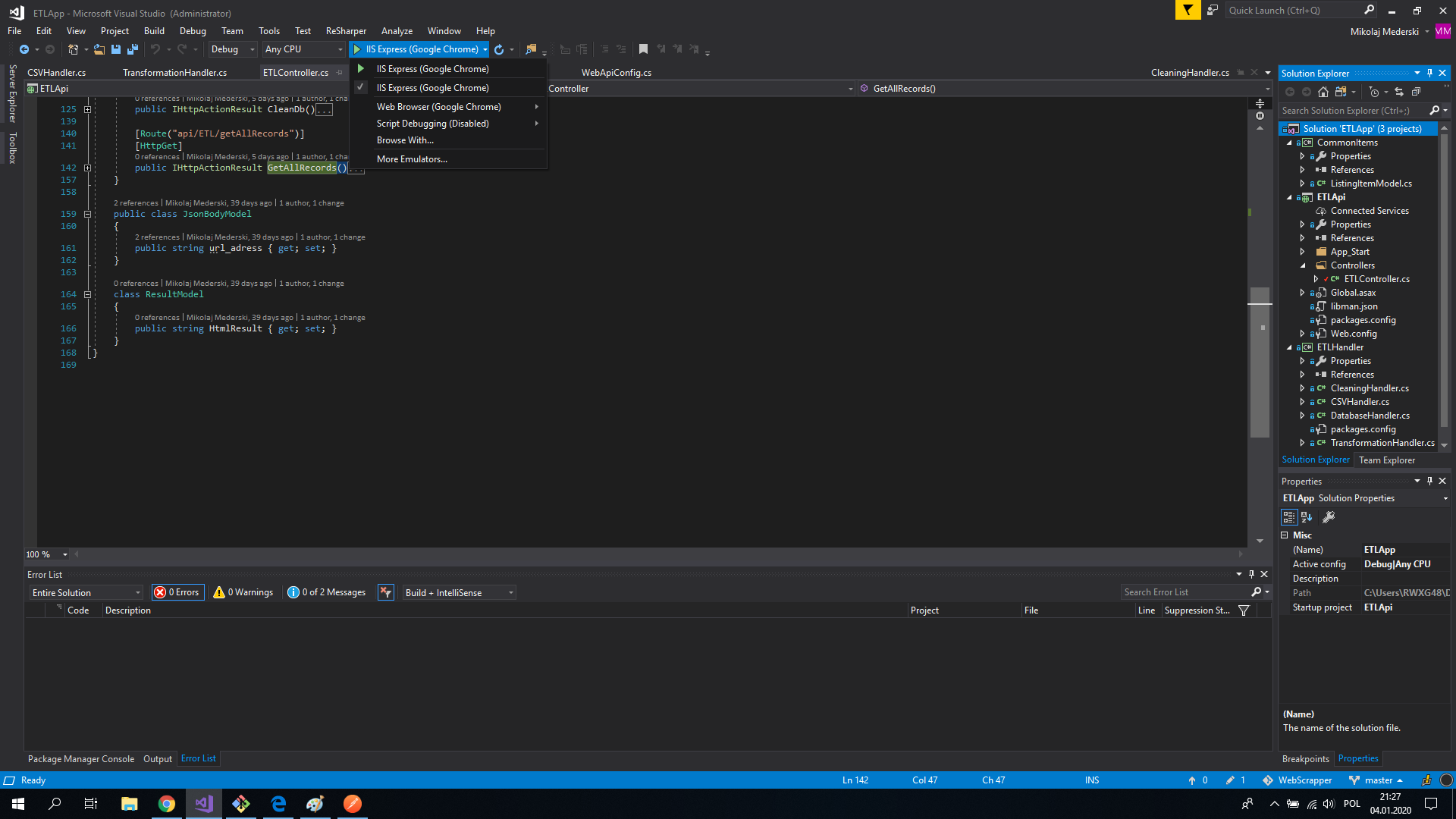
Powyższa komenda automatycznie otworzy domyślną przeglądarkę i uruchomi aplikację pod portem:

<http://localhost:3000/>

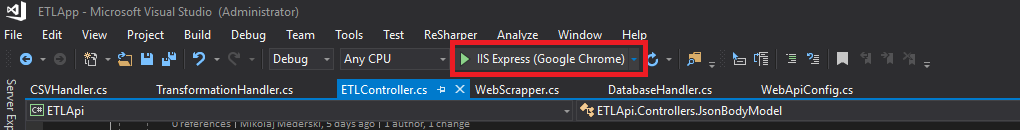
## Uruchomienie części back-end’owej:

Aby uruchomić API obsługujące front-end aplikacji należy:

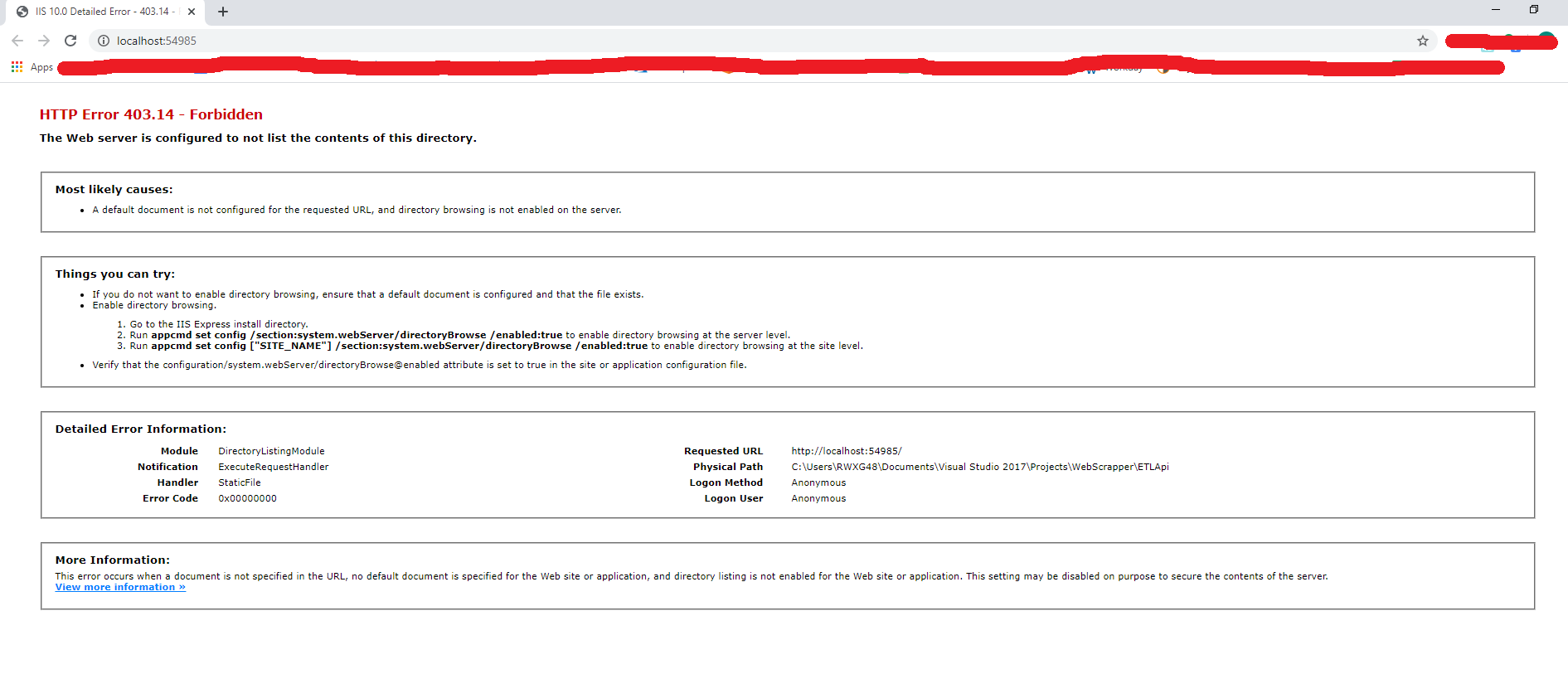
* Uruchomić program Visual Studio 2017
* Otworzyć plik "ETLApp.sln" w programie Visual Studio 2017 wybierając kolejno w lewym górnym rogu ekranu File -> Open -> Project/Solution
* W menu podświetlonym na zdjęciu poniżej wybrać opcje IIS Express



* Kliknąć w przycisk Startu



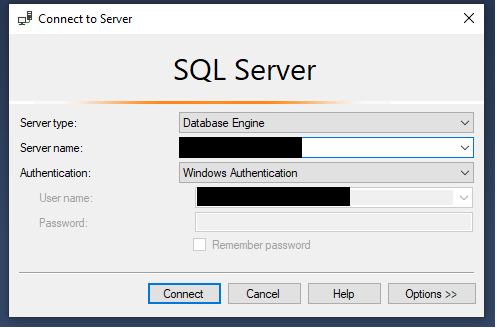
Po kliknięciu w przycisk Start, otworzy się nowe okno przeglądarki. Komunikat ***„403 Forbidden”*** wskazuje, na poprawne działanie serwisu.



## Uruchomienie bazy danych

Aby uruchomić Bazę Danych należy:

* Uruchomić Microsoft SQL Server Management Studio 18
* Kliknąć przycisk Connect i zaczekać na połączenie z serwerem



Uruchomić skrypt Tabela\_Dane.exe.

# Opis funkcjonalności aplikacji:

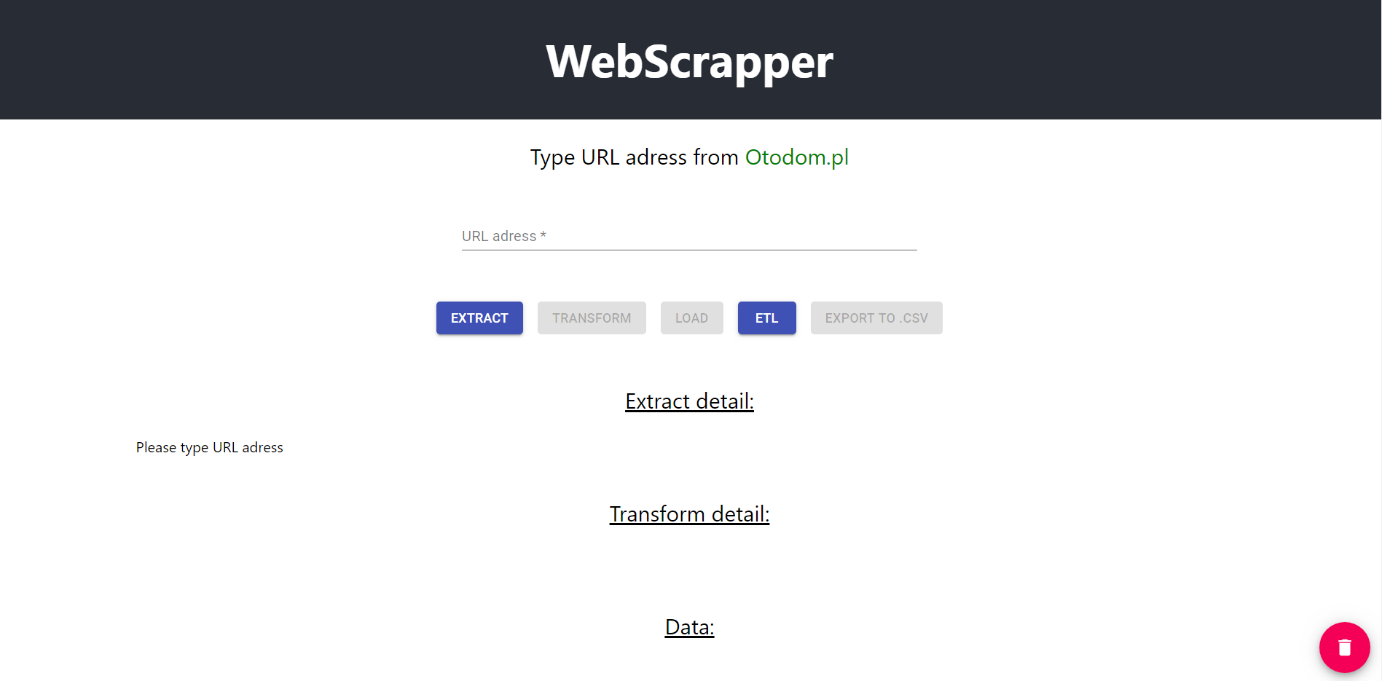
Aplikacja ma na celu pobranie danych ze źródła (otodom.pl), odpowiedniego ich przekształcenia i zasilenia nimi bazy danych. Są to składowe procesu ETL, który główną funkcjonalnością naszej aplikacji. Ponadto istnieje możliwość exportu danych do piku .csv lub .txt. Dane wyświetlane w tabeli możemy filtrować, sortować w sposób umożliwiający użytkownikowi jak najlepsze ich przetwarzanie.

# Opis scenariuszy użycia aplikacji:

Aplikacja posiada dwa scenariusze użycia:

* W pierwszym musimy uruchomić wszystkie etapy manualnie pamiętając, że do uruchomienia kolejnego konieczne jest pozytywne zakończenie etapu poprzedniego.
* W drugim scenariuszu cały proces wykonywany jest automatycznie przy użyciu przycisku *„ETL”.*

## Opis menu i widoków okna aplikacji:

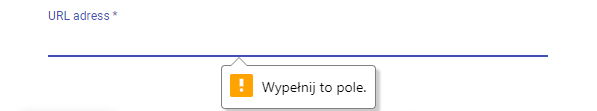


## Aplikacja podzielona została na kilka sekcji:

### Menu funkcyjne:

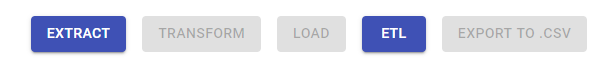
* Pole typu input – pozwalające wprowadzić adres URL strony, którą użytkownik będzie Scrapował wraz z walidacją wymagająca wpisania poprawnego adresu ze strony otodom.pl







* Przycisku funkcyjne odpowiedzialne za uruchamianie kolejnych etapów procesu ETL osobno - *‘EXTRACT’, ‘TRANSFORM’, ‘LOAD’*, przycisk *‘ETL’* wykonujący cały proces automatycznie oraz *‘EXPORT TO .CSV’* generujący plik z wszystkimi danymi znajdującymi się w tabeli w formacje .csv. W początkowej fazie działania aplikacji nie ma możliwości użycia przycisków wyszarzonych, ponieważ wymagają one ukończenia poprzedniego etapu procesu.

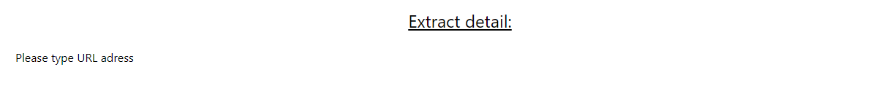


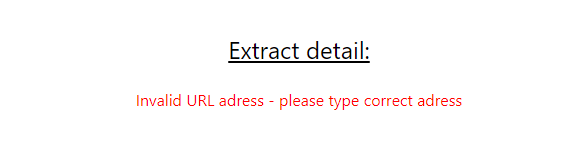
* Przycisk odpowiedzialny za czyszczenie bazy danych

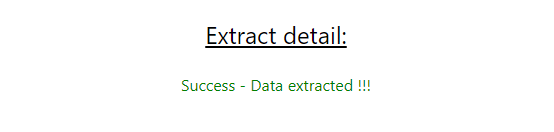


### Informacje o postępie i efektach procesu

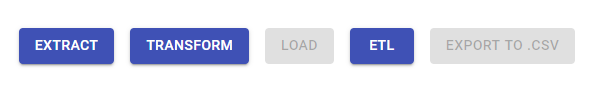
* *EXTRACT* – sekcja mówiąca o postępie etapu *EXTRACT*. Informuje nasz czy etap zakończył się sukcesem lub czy podany adres URL jest błędy, ponieważ nie zawiera żadnych ogłoszeń możliwych do pobrania (np. strona startowa domeny otodom.pl).







Po pozywanym zakończeniu etapu *EXTRACT* przycisk *TRANSFORM* staje się aktywny i możemy przejść do kolejnego procesu.

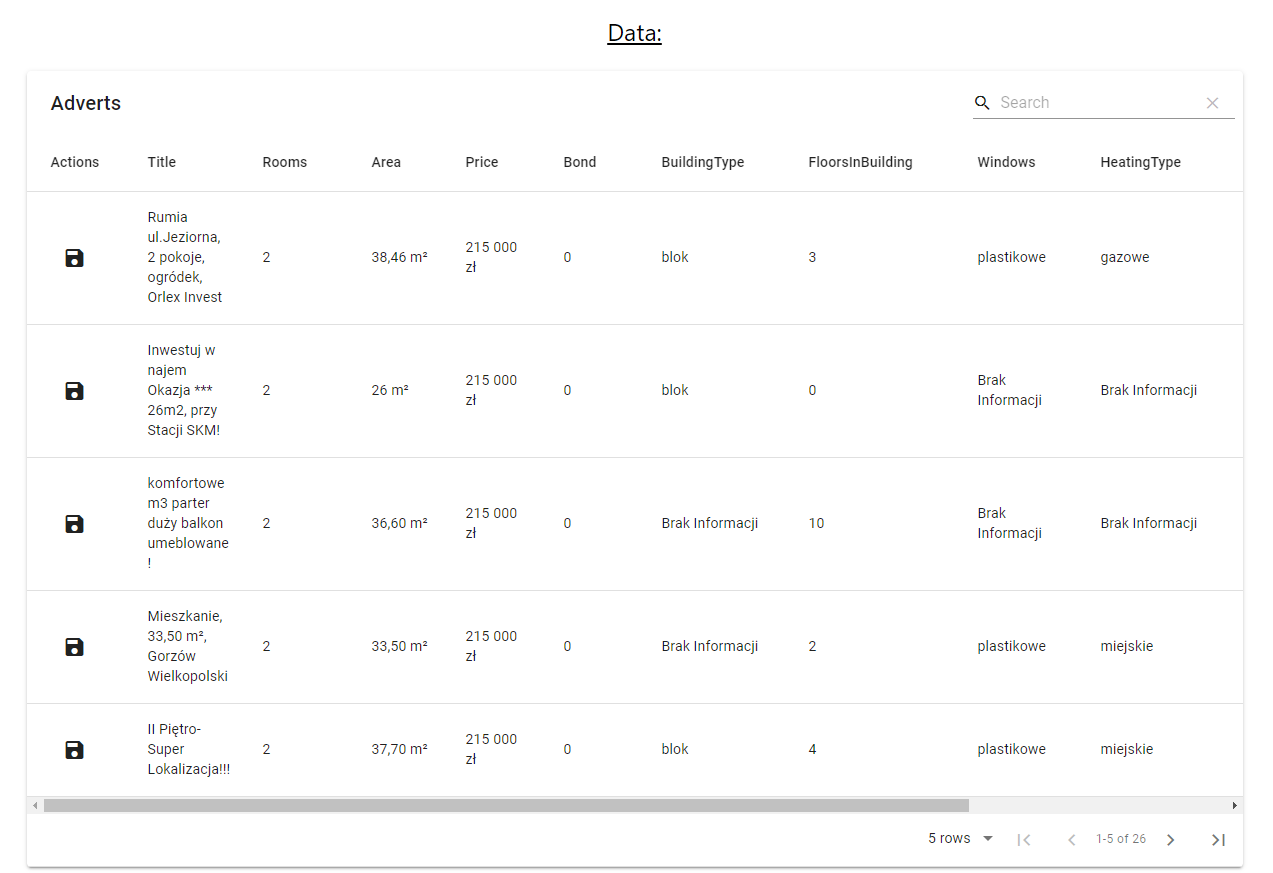


* *TRANSFORM* – informuje nasz ile ofert zostało pobranych i pozywanie przeszło etap *TRANSFORM* po czym aktywuje kolejny etap *LOAD:*

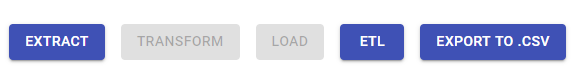
**

**

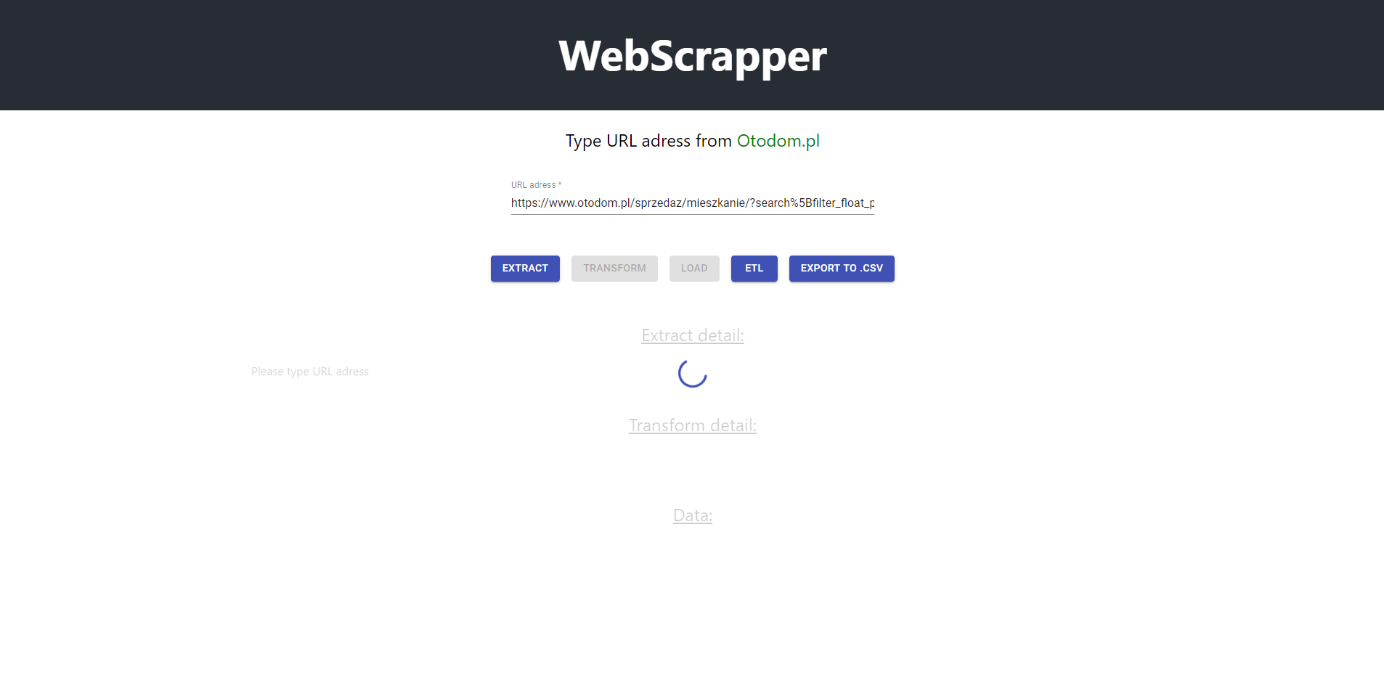
* *DATA* – sekcja w której załadowane zostają pobrane dane w formie tabeli. Tabela posiada sortowanie, wyszukiwarkę, możliwość przesuwania i decydowania o kolejności kolumn, możliwość zmiany ilości wierszy oraz w pierwszej kolumnie ‘Action’ przycisk do exportu pojedynczego wiersza i zapisania go w formacie .txt.



Po zakończeniu etapu *LOAD* możemy wyeksportować cała tabele do pliku .csv jednoczenie wyłączmy możliwość wykonania etapu *TRANSFORM* oraz *LOAD* umożliwiając jedynie rozpoczęcie procesu na nowo i kolejne zasilenie hurtowni danych.



Podczas działania każdego procesu uruchamiany jest loader w celu zablokowania interfejsu użytkownika na czas trwania wykonywania procesu:



1. proszę wymienić konkretne zadania [↑](#footnote-ref-1)