**Gdańsk, czerwiec 2025**

**PromoComparer**

Jakub Kapica

Julia Kusy

Wiktoria Wiśniewska

Spis treści

[1. Architektura systemu 4](#_Toc200721446)

[2. Funkcjonalności 5](#_Toc200721447)

[2.1. Scrapowanie gazetek promocyjnych 5](#_Toc200721448)

[2.2. Konwersja PDF do formatu graficznego 5](#_Toc200721449)

[2.3. Analiza treści obrazu z wykorzystaniem OpenAI 5](#_Toc200721450)

[2.4. Zarządzanie promocjami i ich filtrowanie 6](#_Toc200721451)

[2.5. Zarządzanie strukturą sklepów i kategorii 6](#_Toc200721452)

[2.6. Personalizacja użytkownika – ulubione promocje 7](#_Toc200721453)

[2.7. Automatyzacja zadań cyklicznych 7](#_Toc200721454)

[3. Konfiguracja i środowisko uruchomieniowe 7](#_Toc200721455)

[3.1. Inicjalizacja hosta i wczytanie konfiguracji 7](#_Toc200721456)

[3.2. Rejestracja usług i zależności (Dependency Injection) 8](#_Toc200721457)

[3.3. Konfiguracja CORS, autoryzacji i tożsamości 9](#_Toc200721458)

[3.3.1. CORS 9](#_Toc200721459)

[3.3.2. Uwierzytelnianie i autoryzacja 9](#_Toc200721460)

[3.3.3. Rejestracja ról i użytkowników 10](#_Toc200721461)

[3.4. Budowanie aplikacji i konfiguracja pipeline’u HTTP 10](#_Toc200721462)

[3.5. Tworzenie ról przy starcie aplikacji 12](#_Toc200721463)

[4. Przepływ danych 12](#_Toc200721464)

[4.1. Pozyskanie danych źródłowych 12](#_Toc200721465)

[4.2. Konwersja plików PDF do obrazów 13](#_Toc200721466)

[4.3. Analiza semantyczna obrazu (OpenAI) 14](#_Toc200721467)

[4.4. Transformacja danych do modeli i zapis w bazie 15](#_Toc200721468)

[4.5. Udostępnianie danych przez API 15](#_Toc200721469)

[4.6. Personalizacja i interakcje użytkownika 16](#_Toc200721470)

[5. Struktura bazy danych 16](#_Toc200721471)

[5.1. Tabela Store – Sklep 16](#_Toc200721472)

[5.2. Tabela Leaflet – Gazetka promocyjna 17](#_Toc200721473)

[5.3. Tabela Promotion – Promocja 17](#_Toc200721474)

[5.4. Tabela Category – Kategoria 18](#_Toc200721475)

[5.5. Tabela Favourite – Ulubiona promocja 18](#_Toc200721476)

[5.6. Tabela User – Użytkownik systemu 19](#_Toc200721477)

[6. Procedury przetwarzania danych 19](#_Toc200721478)

[6.1. GetPromotionsByStore 19](#_Toc200721479)

[6.2. GetTop10LargestPromotions 20](#_Toc200721480)

[6.3. GetPromotionsByCategory 21](#_Toc200721481)

[7. Kontrolery 21](#_Toc200721482)

[7.1. CategoryController 21](#_Toc200721483)

[7.2. LeafletController 23](#_Toc200721484)

[7.3. PromotionController 23](#_Toc200721485)

[7.4. StoreController 24](#_Toc200721486)

[7.5. OpenAIController 25](#_Toc200721487)

[7.6. ScrapingController 26](#_Toc200721488)

[7.7. UserPanelController 27](#_Toc200721489)

[8. Usługi 28](#_Toc200721490)

[8.1. CategoryService 28](#_Toc200721491)

[8.2. LeafletService 28](#_Toc200721492)

[8.3. PromotionService 29](#_Toc200721493)

[8.4. StoreService 31](#_Toc200721494)

[8.5. PdfHandlerService 32](#_Toc200721495)

[8.6. OpenAIService 33](#_Toc200721496)

[8.7. Scheduler 33](#_Toc200721497)

[8.8. UserPanelService 34](#_Toc200721498)

# Architektura systemu

Aplikacja **PromoComparer** została zaprojektowana z zachowaniem bieżących zasada inżynierii oprogramowania, wykorzystując architekturę trójwarstwową, która wspiera modularność, łatwość utrzymania oraz rozszerzalność systemu. Całość oparta jest na platformie ASP.NET Core, co zapewnia wysoką wydajność, wsparcie dla RESTful API oraz szeroką integrację z ekosystemem .NET.

Pierwszą warstwę systemu stanowi **interfejs API**, czyli zestaw kontrolerów odpowiedzialnych za odbieranie i przetwarzanie żądań HTTP. Kontrolery odpowiadają za mapowanie przychodzących zapytań do odpowiednich metod logiki biznesowej oraz za serializację wyników do formatu JSON, który jest standardowym sposobem komunikacji z frontendem aplikacji. Każdy z kontrolerów pełni określoną rolę tematyczną – przykładowo, osobne kontrolery odpowiadają za operacje związane z promocjami, gazetkami, sklepami, kategoriami czy interakcją z systemem OpenAI. Dzięki temu separacja odpowiedzialności w tej warstwie jest zachowana, co ułatwia zarówno testowanie, jak i dalszy rozwój aplikacji.

Kolejnym filarem architektury jest **warstwa logiki biznesowej**, która obejmuje zestaw serwisów implementujących kluczowe funkcjonalności aplikacji. Serwisy odpowiadają za realizację procesów takich jak analiza obrazów przy użyciu modeli językowych OpenAI, harmonogramowanie pobierania gazetek promocyjnych, transformację danych z formatu JSON do struktur domenowych, zarządzanie danymi o promocjach oraz logikę personalizacji użytkownika. Każdy serwis jest projektowany jako samodzielna jednostka funkcjonalna, co pozwala na niezależne testowanie, reużywalność kodu oraz łatwe wprowadzanie modyfikacji bez wpływu na inne obszary systemu.

Trzecią warstwą systemu jest **warstwa dostępu do danych**, która obejmuje zarówno definicje modeli domenowych (encje), jak i klasę kontekstu bazy danych ApplicationDbContext. Modele reprezentują poszczególne jednostki danych, takie jak promocje, kategorie, sklepy czy gazetki, i zawierają informacje o relacjach między nimi. Kontekst bazy danych jest odpowiedzialny za konfigurację dostępu do SQL Server oraz zarządzanie cyklem życia danych z wykorzystaniem technologii Entity Framework Core. Migracje definiowane w tej warstwie umożliwiają ewolucję schematu bazy danych w sposób kontrolowany i spójny z logiką aplikacyjną.

Cała struktura aplikacji została zorganizowana w sposób wspierający zasadę pojedynczej odpowiedzialności (Single Responsibility Principle) oraz rozdzielenie zależności (Separation of Concerns). Dzięki temu system charakteryzuje się wysokim poziomem czytelności, łatwością testowania jednostkowego, a także odpornością na błędy wynikające ze sprzężenia między komponentami.

# 2. Funkcjonalności

## 2.1. Scrapowanie gazetek promocyjnych

Jednym z głównych zadań systemu PromoComparer jest automatyczne pozyskiwanie treści promocyjnych w postaci gazetek reklamowych publikowanych na stronach internetowych sieci handlowych. Proces ten jest inicjowany przez ScrapingController i realizowany w PdfHandlerService. Scrapowanie obejmuje lokalizację plików PDF na stronach docelowych, pobieranie ich do systemu plików aplikacji oraz przygotowanie do dalszego przetwarzania.

Pliki zapisywane są w katalogach przypisanych do poszczególnych sklepów (Stem jako identyfikator). Operacja ta jest zautomatyzowana i może być wyzwalana ręcznie lub cyklicznie przy pomocy harmonogramu.

## 2.2. Konwersja PDF do formatu graficznego

Po pobraniu plików PDF, kolejnym krokiem przetwarzania jest ich konwersja do formatu rastrowego (JPEG, PNG). Operacja ta realizowana jest przez metodę ConvertAllPdfsToImagesAndDelete() w serwisie PdfHandlerService. Proces konwersji jest niezbędny, ponieważ analiza treści odbywa się na poziomie obrazu, a nie pliku PDF.

Konwersja realizowana jest z wykorzystaniem narzędzi zewnętrznych, które transformują każdą stronę PDF na osobny plik graficzny. Pliki te są następnie przetwarzane przez usługę OpenAI. Oryginalne pliki PDF są usuwane po zakończeniu konwersji, co pozwala na optymalizację przestrzeni dyskowej.

## 2.3. Analiza treści obrazu z wykorzystaniem OpenAI

Najbardziej zaawansowaną funkcjonalnością systemu jest semantyczna analiza obrazów gazetek promocyjnych przy użyciu modeli językowych OpenAI. Proces ten realizowany jest w serwisie OpenAIService, który implementuje interfejs IOpenAIService.

Metoda ParseImagesToFunction() iteruje po folderach zawierających obrazy gazet promocyjnych, a następnie dla każdego obrazu wywołuje metodę GetJsonPromotions(). Ta z kolei konstruuje odpowiedni prompt, wysyła obraz do API OpenAI i oczekuje odpowiedzi w formacie JSON zawierającej szczegóły oferty promocyjnej. Przykładowy zestaw danych pozyskanych z modelu może obejmować:

* nazwę produktu,
* cenę pierwotną i promocyjną,
* typ promocji (np. rabat procentowy, 2+1),
* jednostkę sprzedaży (np. kg, szt.),
* daty obowiązywania promocji.

Uzyskane dane są następnie przekształcane do modelu Promotion i zapisywane w bazie danych przy użyciu serwisu PromotionService.

## 2.4. Zarządzanie promocjami i ich filtrowanie

System pozwala na zarządzanie strukturą danych dotyczących promocji, w tym ich tworzenie, pobieranie, filtrowanie i udostępnianie przez API. Funkcje te obsługiwane są przez PromotionController oraz IPromotionService.

Użytkownik może pobierać:

* wszystkie promocje (/api/promotions),
* tylko aktywne promocje (/api/promotions/active),
* promocje w konkretnym sklepie (/api/promotions/store/{storeId}),
* promocje z wybranej kategorii (/api/promotions/category/{categoryId}),
* 10 największych promocji według wartości rabatu (/api/promotions/top).

Wszystkie operacje są optymalizowane poprzez DTO, które dostarczają jedynie istotne informacje z punktu widzenia frontendu. Ponadto system oblicza rabaty (kwotowo i procentowo) na etapie ekstrakcji danych.

## 2.5. Zarządzanie strukturą sklepów i kategorii

System umożliwia rejestrowanie sklepów (Store) oraz tworzenie i utrzymywanie listy kategorii produktów (Category). Operacje na tych jednostkach realizowane są przez kontrolery StoreController oraz CategoryController, a logikę obsługują odpowiednie serwisy implementujące interfejsy IStoreService i ICategoryService.

Sklepy identyfikowane są przez unikalną nazwę oraz atrybut Stem, który służy do identyfikacji URL źródłowych gazetek. Kategorie natomiast wykorzystywane są do klasyfikacji produktów promocyjnych w systemie.

Zarówno sklepy, jak i kategorie mogą być tworzone ręcznie (np. z listy w konfiguracji), jak również automatycznie, jeśli dane pojawią się w trakcie analizy obrazu przez OpenAI.

## 2.6. Personalizacja użytkownika – ulubione promocje

Aplikacja obsługuje funkcję dodawania promocji do listy ulubionych, co pozwala użytkownikowi na personalizację interfejsu oraz szybki dostęp do interesujących go ofert. Dane te są przechowywane w tabeli Favourites i powiązane bezpośrednio z użytkownikiem systemu (User).

Interakcja z listą ulubionych odbywa się przez kontroler UserPanelController, który obsługuje dodawanie i usuwanie pozycji oraz pobieranie listy ulubionych ofert użytkownika. Funkcja ta wymaga uwierzytelnienia — aplikacja wykorzystuje tożsamość użytkownika na podstawie ClaimsPrincipal.

## 2.7. Automatyzacja zadań cyklicznych

System wyposażony jest w harmonogram zadań realizowany przy pomocy biblioteki **Coravel**, która umożliwia cykliczne uruchamianie określonych procesów — bez konieczności korzystania z zewnętrznego systemu CRON.

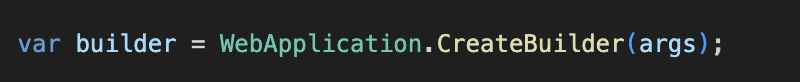
Harmonogram automatycznie uruchamia proces: pobierania gazetek, konwersji PDF oraz analizy przez OpenAI. Dzięki temu baza danych jest stale aktualizowana, a system może działać w trybie niemal bezobsługowym.

# 3. Konfiguracja i środowisko uruchomieniowe

Konfiguracja środowiska aplikacji PromoComparer odbywa się w pliku Program.cs, który wykorzystuje skrócony model hostowania dostępny w ASP.NET Core 6+. Wspomniany plik zawiera wszystkie niezbędne kroki uruchomienia aplikacji -począwszy od wczytania konfiguracji, przez rejestrację zależności i usług, aż po zdefiniowanie ścieżek HTTP i uruchomienie serwera.

## 3.1. Inicjalizacja hosta i wczytanie konfiguracji

Proces konfiguracji rozpoczyna się od utworzenia instancji WebApplicationBuilder



Rys. 3. 1 Inicjalizacja aplikacji ASP .NET Core w stylu “minimal hosting”

ASP.NET Core automatycznie ładuje:

* appsettings.json – bazowy plik konfiguracyjny,
* appsettings.{Environment}.json – plik środowiskowy, np. appsettings.Development.json,
* zmienne środowiskowe – np. zdefiniowane w systemie

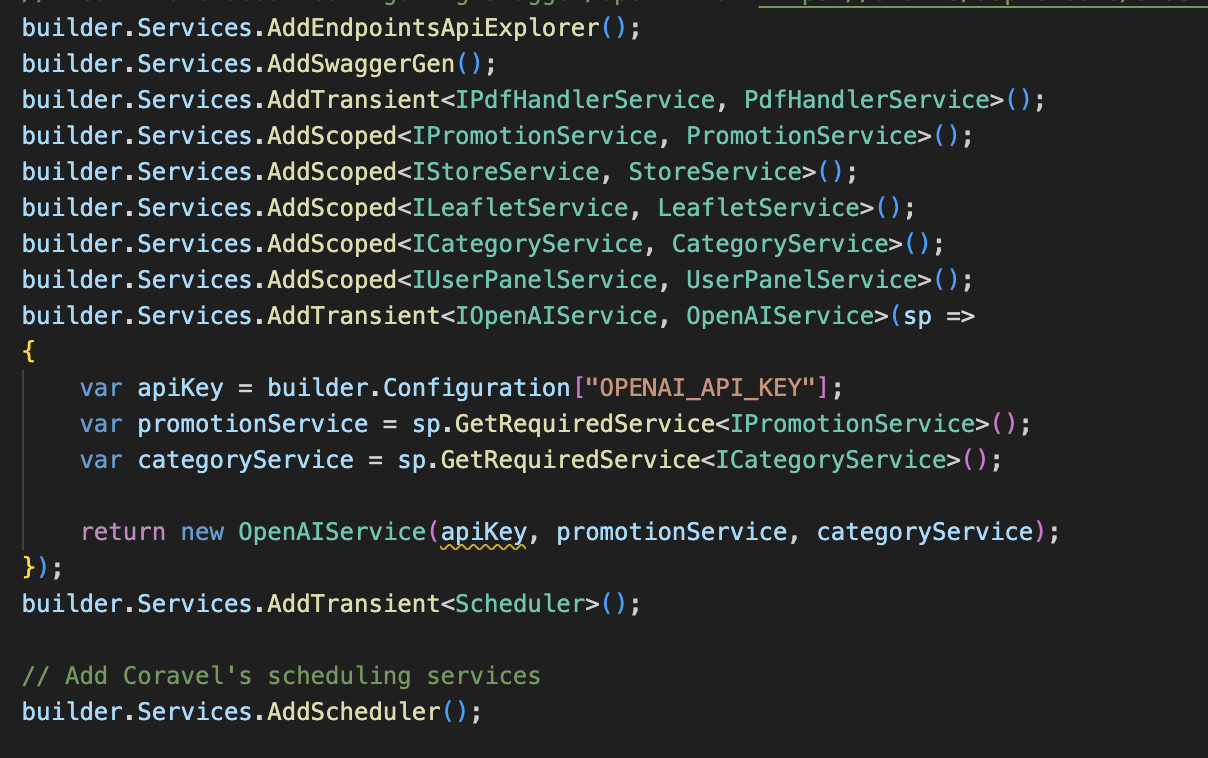
## 3.2. Rejestracja usług i zależności (Dependency Injection)

System korzysta z wbudowanego kontenera DI (Dependency Injection), do którego rejestrowane są wszystkie potrzebne komponenty: kontekst bazy danych, serwisy aplikacyjne oraz planista Coravel.



Rys. 3.2. Konfiguracja usług w ASP .NET Core

Następnie rejestrowane są konkretne serwisy, zgodnie z ich interfejsami.

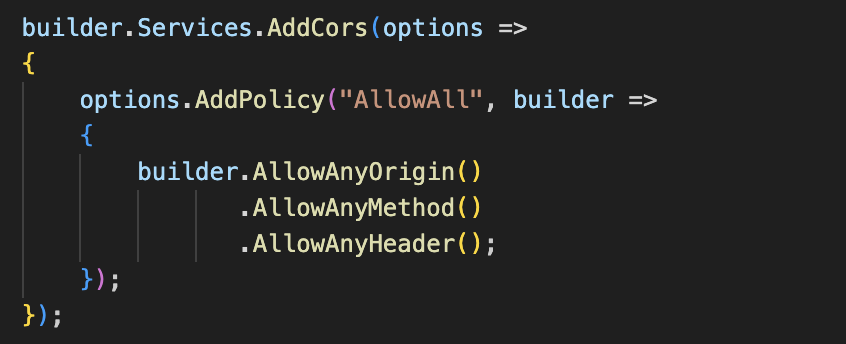


Rys. 3. 3 Rejestracja usług w kontenerze DI wraz z harmonogramowaniem

## 3.3. Konfiguracja CORS, autoryzacji i tożsamości

### 3.3.1. CORS

System umożliwia dostęp do API z dowolnej domeny (np. dla frontendowej części aplikacji)

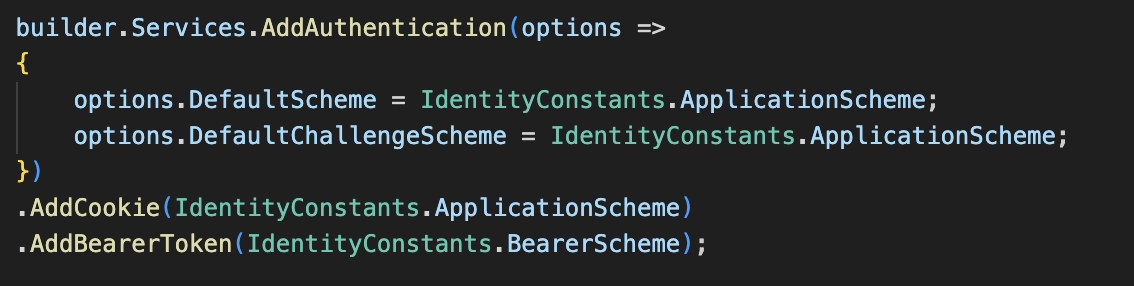


Rys. 3. 4 Konfiguracja polityki CORS

Powyższy fragment rejestruje w kontenerze usług prostą, otwartą politykę CORS o nazwie AllowAll. Dzięki niej każda domena (origin) może wysyłać żądania do Twojego API, używając dowolnych metod HTTP (GET, POST, PUT, DELETE itp.) oraz przesyłając dowolne nagłówki.

### 3.3.2. Uwierzytelnianie i autoryzacja

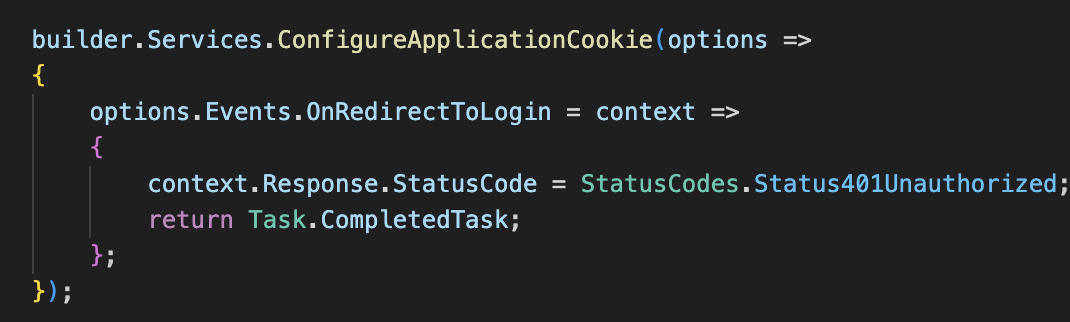
Aplikacja korzysta z ASP.NET Core Identity z obsługą ciasteczek i tokenów JWT



Rys. 3.5 Konfiguracja uwierzytelniania w ASP .NET Core

Powyższy fragment kodu rejestruje w kontenerze usług mechanizmy uwierzytelniania aplikacji. Najpierw za pomocą AddAuthentication ustalane są domyślne schematy: DefaultScheme oraz DefaultChallengeScheme wskazują na własny, zdefiniowany w IdentityConstants.ApplicationScheme. Następnie metoda AddCookie wprowadza uwierzytelnianie oparte na ciasteczkach dla mechanizmu „Application”, co pozwala na sesyjną obsługę logowania użytkowników. Dzięki wywołaniu AddBearerToken (najczęściej mapowanemu na JwtBearer) do aplikacji trafiają także żądania z nagłówkiem Authorization: Bearer <token>, umożliwiając bezstanowe uwierzytelnianie tokenami JWT.

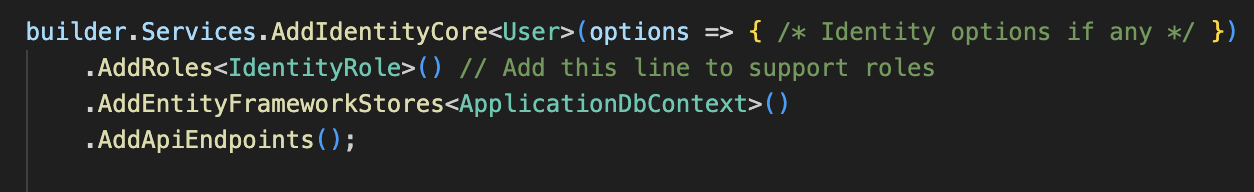
Dodatkowo zdefiniowane jest zachowanie w przypadku nieautoryzowanego żądania



Rys. 3. 6 Dostosowanie zachowania przekierowania do logowania

Powyższy fragment kodu konfiguruje opcje cookie-based authentication, aby w przypadku braku uwierzytelnienia nie wykonywać domyślnego przekierowania na stronę logowania, lecz zwrócić od razu status HTTP 401 Unauthorized. Dzięki podpięciu zdarzenia OnRedirectToLogin i ustawieniu context.Response.StatusCode = StatusCodes.Status401Unauthorized, klienci API otrzymują jednoznaczną odpowiedź o braku uprawnień zamiast HTML-owego przekierowania

### 3.3.3. Rejestracja ról i użytkowników



Rys. 3. 7 Konfiguracja ASP .NET Core Identity z obsługą ról

W powyższym kodzie:

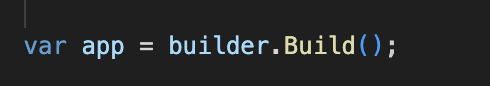
* User to klasa definiująca model użytkownika aplikacji,
* AddRoles<IdentityRole>() aktywuje obsługę ról,
* AddEntityFrameworkStores<ApplicationDbContext>() wskazuje, że dane

o użytkownikach będą przechowywane w bazie danych za pomocą EF Core,

* AddApiEndpoints() dodaje domyślne endpointy API dla zarządzania kontami.

## 3.4. Budowanie aplikacji i konfiguracja pipeline’u HTTP

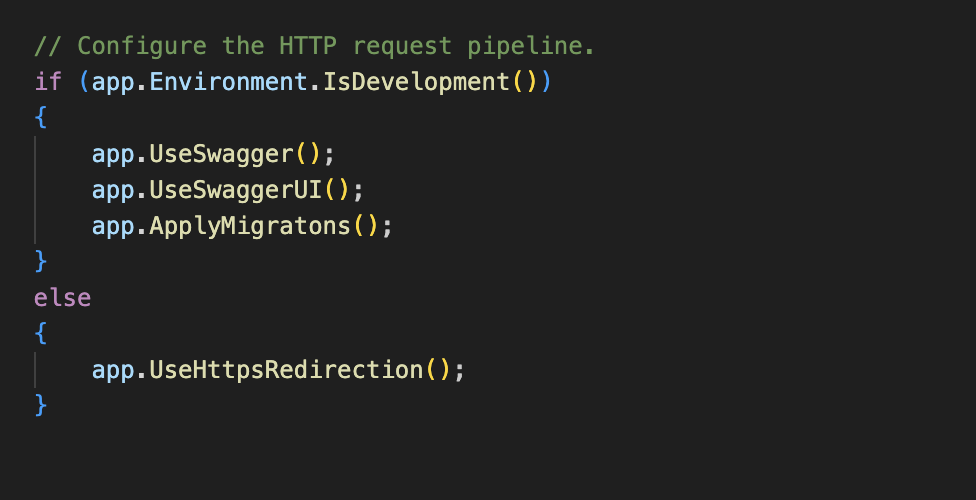
Po skonfigurowaniu usług, tworzona jest aplikacja webowa



Rys. 3. 8 Utworzenie aplikacji webowej

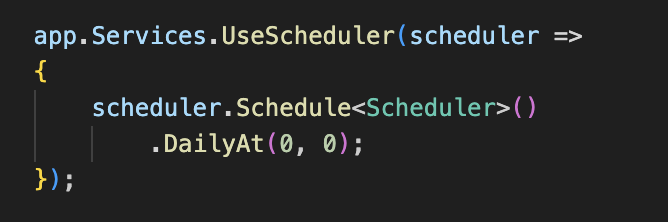
W środowisku deweloperskim aplikacja uruchamia Swagger oraz migracje.

Realizowana zostaje ścieżka przetwarzania przychodzących żądań w zależności od środowiska uruchomieniowego aplikacji. W trybie deweloperskim (*IsDevelopment())* aktywuje się dokumentacja **OpenAPI** (metody *UseSwagger()* *i UseSwaggerUI())* oraz automatyczne zastosowanie migracji bazy danych (*ApplyMigrations()),* co przyspiesza iterację nad zmianami w modelu. W środowisku produkcyjnym natomiast włącza się przekierowanie ruchu HTTP na HTTPS *(UseHttpsRedirection())*



Rys. 3. 9 Konfiguracja potoku żądań HTTP

Uruchamiany jest harmonogram

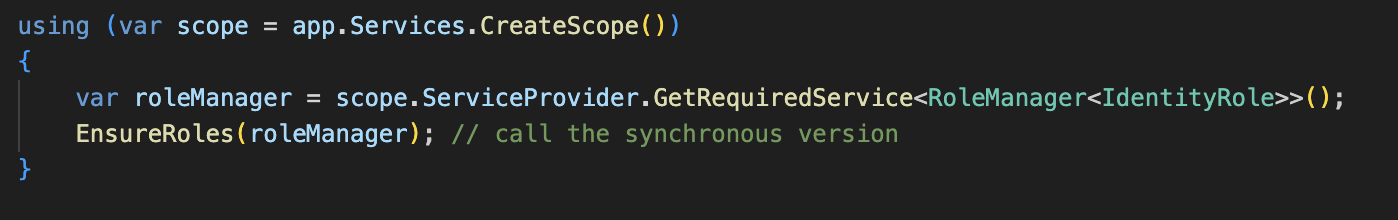


Rys. 3. 10 Uruchomienie harmonogramu

## 3.5. Tworzenie ról przy starcie aplikacji

Po uruchomieniu aplikacji, role **Admin** i **User** są tworzone automatycznie w bazie danych w ramach procedury inicjalizacyjnej. Proces ten jest realizowany w bloku inicjalizacji, który wykonuje się natychmiast po zbudowaniu aplikacji, zanim zostanie uruchomiony serwer HTTP.

Kod odpowiedzialny za to znajduje się bezpośrednio po wywołaniu builder.Build()



# 4. Przepływ danych

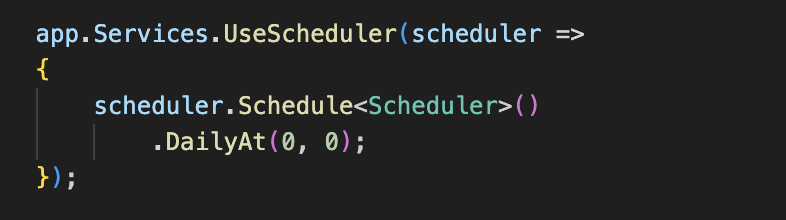
System PromoComparerAPI realizuje kompletny cykl przetwarzania danych promocyjnych — począwszy od pozyskiwania gazet promocyjnych z zewnętrznych źródeł, przez analizę treści z użyciem sztucznej inteligencji, aż po zapis danych w relacyjnej bazie danych i udostępnienie ich przez REST API.

## 4.1. Pozyskanie danych źródłowych

Proces pozyskiwania danych promocyjnych rozpoczyna się od **scrapowania** — czyli automatycznego pobierania gazet promocyjnych (plików PDF) z witryn internetowych.

Mechanizm ten może zostać uruchomiony na dwa sposoby:

* bezpośrednie wywołanie endpointu API -POST /api/scraping/parse
* automatyczny **harmonogram Coravel**, który wyzwala scrapowanie codziennie o północy



Rys. 4. 1 Harmonogramowanie zadania

Logikę biznesową obsługuje klasa *PdfHandlerService*

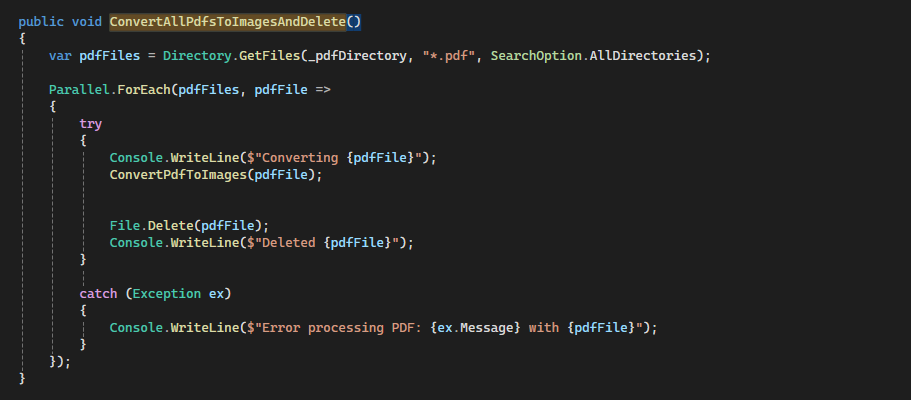
Wewnątrz tej metody system:

* iteruje po zdefiniowanych sklepach (Store),
* sprawdza, czy dany sklep ma przypisany adres źródłowy (Stem),
* wykonuje żądania HTTP do pobrania dostępnych plików PDF,
* zapisuje je do lokalnego katalogu roboczego.

Każdy plik trafia do podfolderu odpowiadającego nazwie sklepu, co pozwala łatwo kontrolować ich źródło i identyfikację podczas dalszego przetwarzania.

## 4.2. Konwersja plików PDF do obrazów

Aby umożliwić analizę treści zawartej w gazetkach, pliki PDF muszą zostać przekształcone. W tym celu aplikacja uruchamia metodę:



Rys. 4. 2 Metoda ConverAllPdfsToImageAndDelete

Proces działa w następujących krokach:

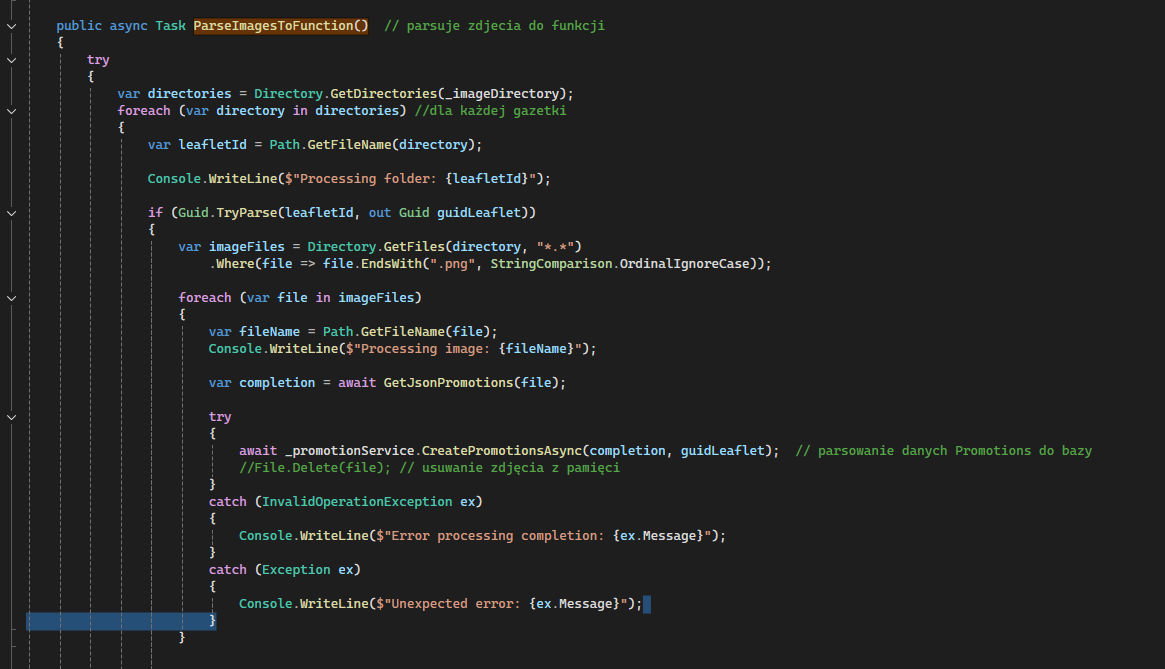
* dla każdego pliku PDF odczytywana jest liczba stron,
* każda strona jest konwertowana na plik JPEG/PNG,
* pliki są zapisywane do katalogu roboczego, a oryginalne PDF-y usuwane.

Do konwersji najprawdopodobniej wykorzystywana jest biblioteka zewnętrzna. Pliki graficzne są przygotowywane do przetworzenia przez AI i zapisywane z identyfikowalnymi nazwami.

## 4.3. Analiza semantyczna obrazu (OpenAI)

Najbardziej zaawansowany etap przepływu danych to **ekstrakcja informacji semantycznej** z obrazów za pomocą modeli językowych OpenAI.

Klasa *ParseImagesToFunction*, która realizuje zaawansowany etap przepływu danych – wywołanie modelu GPT-4O w celu wykrycia i sformatowania informacji o promocjach. Metoda rozpoczyna się od pobrania listy katalogów w folderze Assets, z których każdy nazywa się identyfikatorem ulotki (GUID). Dla każdego folderu następuje weryfikacja poprawności GUID-a, a następnie przetworzenie wszystkich plików PNG.



Rys. 4.3 Funkcja ParseImagesToFunction

Kolejny kluczowy element to sposób, w jaki przygotowywany jest prompt i dane wejściowe do modelu. Metoda GetJsonPromotions wczytuje zawartość pliku jako strumień bajtów, załącza go wraz z opisem w formie tekstowej i prosi GPT-4O o zwrócenie danych wyłącznie w formacie JSON

Dla każdego folderu sklepu i każdego obrazu system:

* tworzy prompt tekstowy zawierający instrukcje dla modelu OpenAI (np. "Wypisz wszystkie promocje na stronie..."),
* wywołuje model OpenAI z obrazem jako inputem (model typu GPT-4 z obsługą obrazu lub inna kombinacja OCR + NLP),
* otrzymuje odpowiedź w formacie JSON z polami:
  + productName
  + originalPrice, discountPrice
  + promotionType
  + unit
  + validFrom, validTo
  + categoryName

Otrzymany wynik jest parsowany do obiektów .NET i przekazywany do kolejnego etapu.

## 4.4. Transformacja danych do modeli i zapis w bazie

Po uzyskaniu danych z modelu AI, system przystępuje do ich transformacji do struktur wewnętrznych i zapisuje je w bazie danych. Obsługuje to głównie PromotionService, ale również inne serwisy wspomagające:

* CategoryService – kategorie produktów,
* StoreService – identyfikacja sklepu,
* LeafletService – przypisanie do gazetki.

Promocje są zapisywane do tabeli Promotions z pełnym zestawem informacji, w tym:

* Id sklepu i kategorii (relacje obce),
* Opis, ceny, typ promocji, jednostka,
* Daty obowiązywania,
* Ewentualny identyfikator zewnętrzny (do deduplikacji).

System zapewnia spójność danych — kategorie i sklepy są tworzone dynamicznie, jeśli nie istnieją.

## 4.5. Udostępnianie danych przez API

Zgromadzone dane są udostępniane klientom (np. aplikacji frontendowej) przez *RESTful API*. Interfejsy kontrolerów takie jak *PromotionController* pozwalają na różnorodne operacje filtrowania i wyszukiwania:

Dane są serializowane do formatu JSON za pomocą DTO, co pozwala na:

* uproszczenie modelu danych dla frontendu,
* zmniejszenie rozmiaru odpowiedzi HTTP,
* ukrycie pól wewnętrznych, np. StoreId, EF navigation properties.

Wszystkie żądania są przetwarzane przez *IPromotionService* oraz *ApplicationDbContext*.

## 4.6. Personalizacja i interakcje użytkownika

Użytkownicy zarejestrowani w systemie mogą korzystać z funkcji personalizacji. Główna funkcja to możliwość dodania promocji do ulubionych. Dane te są przechowywane w relacji wiele-do-wielu między User i Promotion:

* Tabela pośrednia: FavouritePromotion
* Obsługa: UserPanelController i UserPanelService

Dostęp do tych zasobów wymaga autoryzacji (token *JWT* lub ciasteczko *auth* *cookie*). Tożsamość użytkownika odczytywana jest z *ClaimsPrincipal*.

# 5. Struktura bazy danych

Aplikacja PromoComparer wykorzystuje relacyjną bazę danych SQL Server, zarządzaną za pomocą Entity Framework Core. Schemat danych został zaprojektowany w sposób modularny, z uwzględnieniem zasad normalizacji i separacji odpowiedzialności.

## 5.1. Tabela Store – Sklep

Tabela Store reprezentuje sieci handlowe, których gazetki są pobierane, analizowane i prezentowane w systemie. Każdy sklep może mieć przypisane wiele gazetek (Leaflet), co odzwierciedla naturalną relację między siecią a jej kampaniami promocyjnymi.

**Atrybuty:**

* Id (Guid) – klucz główny, jednoznacznie identyfikuje sklep.
* Leaflets (ICollection<Leaflet>) – lista gazetek przypisanych do sklepu.

**Relacje:**

* **1 : N** z tabelą Leaflet – jeden sklep może mieć wiele gazet.

## 5.2. Tabela Leaflet – Gazetka promocyjna

Encja Leaflet odwzorowuje konkretną gazetkę promocyjną, która obowiązuje przez określony czas i zawiera zestaw promocji (Promotion). Powiązana jest jednoznacznie z danym sklepem (Store), a jej treść może być analizowana automatycznie lub ręcznie.

**Atrybuty:**

* Id (Guid) – identyfikator gazetki.
* StartDate (DateTime) – data rozpoczęcia promocji.
* EndDate (DateTime) – data zakończenia promocji.
* PdfLink (string) – link URL do pobrania gazetki w formacie PDF.
* StoreId (Guid) – klucz obcy do tabeli Store.
* Store (Store) – relacja nawigacyjna do sklepu.
* Promotions (ICollection<Promotion>) – zbiór promocji przypisanych do gazetki.

**Relacje:**

* **N : 1** z Store – gazetka należy do jednego sklepu.
* **1 : N** z Promotion – gazetka zawiera wiele promocji.

## 5.3. Tabela Promotion – Promocja

Tabela Promotion zawiera szczegółowe dane dotyczące pojedynczych ofert promocyjnych, które zostały zidentyfikowane w gazetkach. Zawiera informacje o zakresie obowiązywania, produktach, cenach oraz przynależności do kategorii. Jest centralną encją systemu.

**Atrybuty:**

* Id (Guid) – identyfikator promocji.
* UntilOutOfStock (bool) – flaga określająca, czy promocja trwa do wyczerpania zapasów.
* LeafletId (Guid) – odwołanie do gazetki, w której występuje promocja.
* Leaflet (Leaflet) – relacja nawigacyjna do gazetki.
* CategoryId (Guid) – odniesienie do kategorii produktu.
* Category (Category) – relacja nawigacyjna do encji Category.
* Favourites (ICollection<Favourite>) – relacja do użytkowników, którzy dodali promocję do ulubionych.

**Relacje:**

* **N : 1** z Leaflet – promocja należy do jednej gazetki.
* **N : 1** z Category – przypisana do jednej kategorii.
* **1 : N** z Favourite – może być ulubioną wielu użytkowników.

## 5.4. Tabela Category – Kategoria

Encja Category reprezentuje klasyfikację produktów promocyjnych. Umożliwia filtrowanie i grupowanie ofert według typu produktów, co ułatwia użytkownikom przeglądanie interesujących ich promocji.

**Atrybuty:**

* Id (Guid) – identyfikator kategorii.
* Promotions (ICollection<Promotion>) – lista przypisanych promocji.

**Relacje:**

* **1 : N** z Promotion – każda kategoria może mieć wiele przypisanych promocji.

## 5.5. Tabela Favourite – Ulubiona promocja

Tabela Favourite to encja pośrednia, implementująca relację wiele-do-wielu między użytkownikami a promocjami. Umożliwia zapisywanie przez użytkowników swoich ulubionych promocji, co jest funkcją personalizacji.

**Atrybuty:**

* UserId (string) – identyfikator użytkownika (część klucza głównego).
* PromotionId (Guid) – identyfikator promocji (część klucza głównego).
* User (User) – relacja nawigacyjna do encji użytkownika.
* Promotion (Promotion) – relacja nawigacyjna do encji promocji.

**Relacje:**

* **N : 1** z User – wielu użytkowników może oznaczyć tę samą promocję.
* **N : 1** z Promotion – jedna promocja może być ulubiona przez wielu użytkowników.

## 5.6. Tabela User – Użytkownik systemu

Encja użytkownika (User) jest rozszerzeniem modelu ASP.NET Core Identity. Umożliwia autoryzację, zarządzanie rolami (Admin, User) oraz integrację z funkcją ulubionych promocji.

**Atrybuty :**

* Id (string) – unikalny identyfikator użytkownika.
* Favourites – kolekcja ulubionych promocji.

**Relacje:**

* **1 : N** z Favourite – użytkownik może mieć wiele ulubionych promocji.

# 6. Procedury przetwarzania danych

W systemie **PromoComparer** zdefiniowane są procedury aplikacyjne (metody kontrolerów i serwisów), które agregują dane z wielu tabel i wykonują obliczenia domenowe, takie jak analiza rabatów, filtrowanie po sklepie czy kategorii. Poniżej znajduje się zestawienie kluczowych procedur wykorzystywanych do obsługi zapytań użytkowników.

## 6.1. GetPromotionsByStore

Procedura służy do pobierania wszystkich promocji przypisanych do sklepu o podanym identyfikatorze. Wyniki zawierają szczegóły dotyczące produktu, rabatu i przynależności do sklepu i kategorii.

**Wykorzystywane encje:**

* Stores
* Leaflets
* Promotions
* Categories

**Logika działania:**

1. Na podstawie StoreId, pobierane są wszystkie gazetki (Leaflets) powiązane z danym sklepem.
2. Z tych gazetek wyciągane są wszystkie przypisane promocje (Promotions).
3. Dla każdej promocji:
   1. Obliczana jest wartość rabatu (różnica między ceną oryginalną a promocyjną).
   2. Obliczany jest procentowy rabat:
4. Formatowane są daty rozpoczęcia i zakończenia promocji.
5. Zwracana jest lista obiektów zawierających:
   1. Id promocji
   2. ProductName
   3. OriginalPrice i PriceAfterPromotion
   4. DiscountAmount i DiscountPercent
   5. CategoryName
   6. StoreName

## 6.2. GetTop10LargestPromotions

Procedura odpowiada za identyfikację 10 najbardziej atrakcyjnych promocji — pod względem wysokości rabatu w złotówkach, a w przypadku remisu również procentowego.

**Wykorzystywane encje:**

* Promotions
* Leaflets
* Categories
* Stores

**Logika działania:**

1. Pobierane są wszystkie **aktywne** promocje (te, których EndDate jest większy niż data bieżąca).
2. Dla każdej promocji obliczane są:
   1. DiscountAmount = OriginalPrice - PriceAfterPromotion
   2. DiscountPercent = analogicznie jak wyżej
3. Promocje są sortowane:
   1. malejąco według DiscountAmount
   2. następnie malejąco według DiscountPercent
4. Zwracane jest **top 10** promocji z polami:
   1. Id, ProductName, OriginalPrice, PriceAfterPromotion
   2. DiscountAmount, DiscountPercent
   3. CategoryName, StoreName, StoreId

## 6.3. GetPromotionsByCategory

Procedura umożliwia przegląd wszystkich promocji przypisanych do konkretnej kategorii (np. „nabiał”, „napoje gazowane”).

**Wykorzystywane encje:**

* Categories
* Promotions
* Leaflets
* Stores

**Logika działania:**

1. Na podstawie CategoryId system pobiera wszystkie promocje przypisane do tej kategorii.
2. Promocje są filtrowane – pozostają tylko te, których data zakończenia (EndDate) jest przyszła.
3. Dla każdej promocji obliczany jest:
   1. DiscountAmount i DiscountPercent (analogicznie jak wyżej).
4. Wyniki są formatowane i zawierają:
   1. Id promocji
   2. ProductName
   3. OriginalPrice, PriceAfterPromotion
   4. DiscountAmount, DiscountPercent
   5. CategoryName, StoreName

# 7. Kontrolery

Warstwa kontrolerów w aplikacji PromoComparerpełni rolę bramy wejściowej do systemu. Każdy kontroler odpowiada za określoną część logiki aplikacyjnej i pośredniczy między żądaniami HTTP a warstwą serwisów.

## 7.1. CategoryController

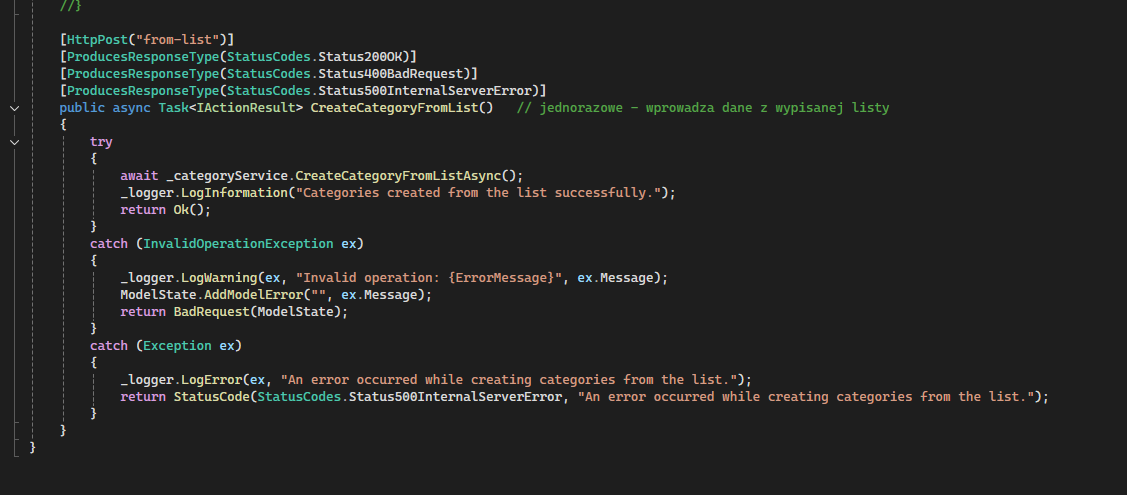
*CategoryController* pełni rolę warstwy pośredniczącej pomiędzy klientem API a serwisem zarządzającym kategoriami produktów. Został oznaczony atrybutem [ApiController] oraz zmapowany na ścieżkę api/Categories. W konstruktorze przyjmuje implementację interfejsu ICategoryService oraz logger, co umożliwia delegowanie logiki biznesowej do warstwy serwisów i rejestrowanie zdarzeń lub błędów.



Rys.7.1 Kod kontrolera CategoryController

Metoda *GetAllCategories* odpowiada na żądania HTTP GET bez dodatkowych parametrów pod adresem /api/Categories.

Wewnątrz tej akcji wywoływana jest asynchronicznie metoda serwisu, zwracająca kolekcję DTO reprezentujących kategorie. W przypadku powodzenia controller zwraca kod 200 ze zserializowaną listą, natomiast wszelkie wyjątki są przechwytywane, logowane jako błąd i przekierowywane do klienta z kodem 500. Dodatkowo, *CreateCategoryFromList*, dostępna pod ścieżką POST /api/Categories/from-list, inicjuje jednokrotne utworzenie kategorii na podstawie wewnętrznie zdefiniowanej listy.

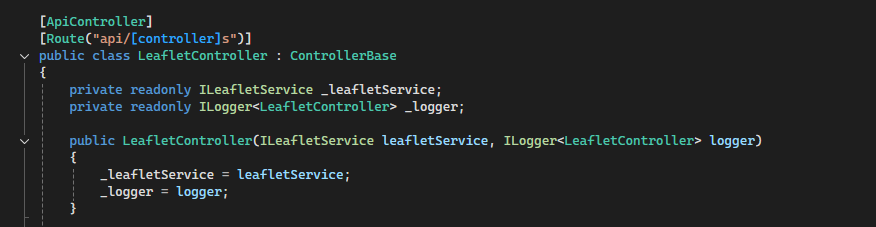


Rys.7.2 Metoda GetAllCategories

W przypadku nieprawidłowej operacji serwer reaguje kodem 400 z informacją o błędzie, a przy nieprzewidzianych wyjątkach – kodem 500 .

## 7.2. LeafletController

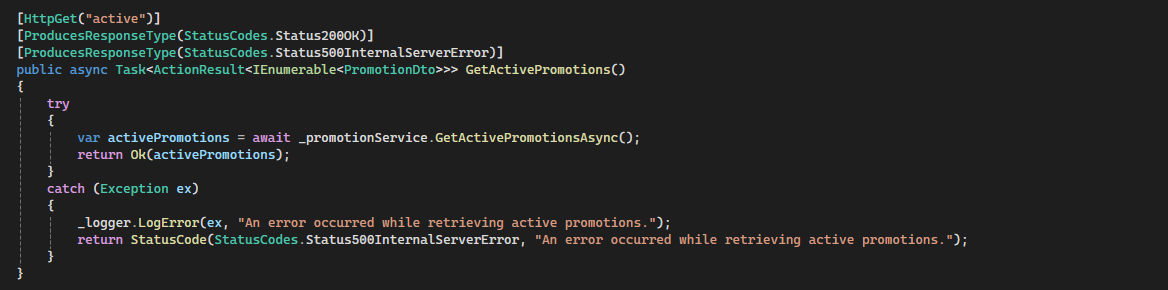
Kontroler LeafletController odpowiada za obsługę „ulotkowych” danych w systemie. Jego ścieżka bazowa to api/Leaflets, co realizowane jest przez [Route("api/[controller]s")]. Pomimo że większość metod została zakomentowana, widać intencję wsparcia pełnego CRUD—pobierania wszystkich rekordów, pojedynczego wpisu czy tworzenia nowych – każda z nich wyposażona w odpowiednie atrybuty HTTP, typy odpowiedzi i logikę przechwytywania wyjątków. Komentarze kodu sugerują, że w przyszłości kontroler ten będzie w pełni operacyjny, z użyciem ILeafletService do wykonywania operacji na obiektach DTO modelu ulotki.



Rys.7.3 Kod kontrolera LeafletController

## 7.3. PromotionController

W *PromotionController* skoncentrowano się na możliwościach filtrowania promocji według różnych kryteriów. Bazowa ścieżka to api/Promotions, a konstruktor przyjmuje serwis *IPromotionService* i *logger*. Metoda *GetActivePromotions*, wywoływana pod adresem GET /api/Promotions/active, zwraca aktualnie obowiązujące promocje, natomiast GetTopPromotions pod /api/Promotions/top przekazuje listę wyróżnionych ofert, wykorzystując typ DTO z dodatkowymi informacjami o topowych promocjach. Dla każdej z tych akcji przewidziano obsługę wyjątków z logowaniem błędów i zwracaniem statusu 500.



Rys. 7.4 Metoda GetActivePromotions

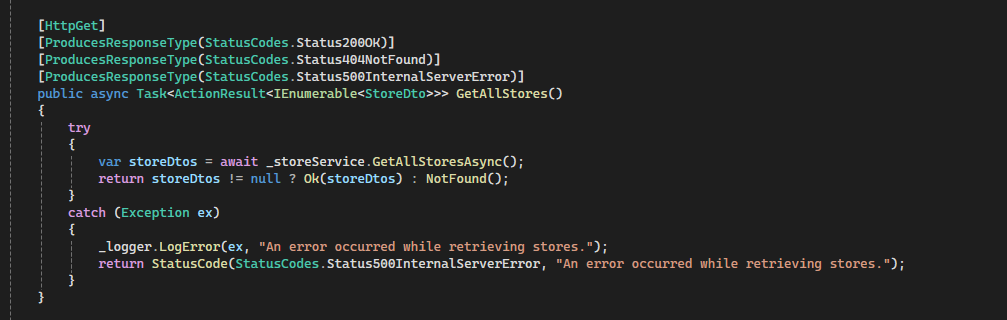
Dodatkowo kontroler udostępnia metodę GetAllActivePromotionsByStore, dostępną pod GET /api/Promotions/store/{storeId}, która najpierw pobiera listę promocji dla wskazanego sklepu, a w przypadku braku wyników odpowiada kodem 404. Analogicznie działa *GetAllActivePromotionsByCategory* pod /api/Promotions/category/{categoryId}, również zwracając status 404, gdy nie odnajdzie żadnej promocji dla danej kategorii. Dzięki tej strukturze klient może elastycznie pozyskiwać zarówno wszystkie aktywne promocje, jak i zawężać wyniki do konkretnego sklepu lub kategorii.



Rys.7.5 Metoda GetAllActivePromotionsByCategory

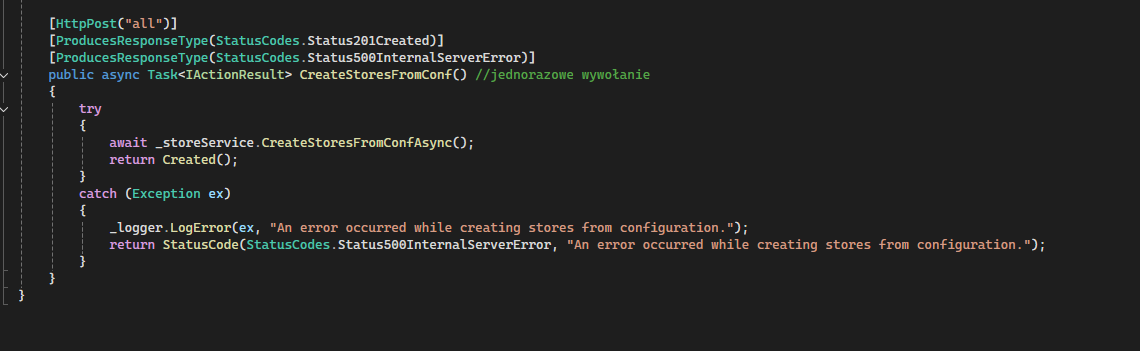
## 7.4. StoreController

StoreController odpowiada za zarządzanie zasobami sklepów. Dzięki oznaczeniu [Route("api/[controller]s")] wszystkie akcje trafiają pod ścieżkę api/Stores. Metoda *GetAllStores* realizuje zapytanie GET, które przy pomocy serwisu IStoreService pobiera całą listę sklepów. Jeżeli lista jest pusta, zwracany jest kod 404, natomiast przy błędach wewnętrznych – kod 500.



Rys.7.6 Metoda GetAllStores

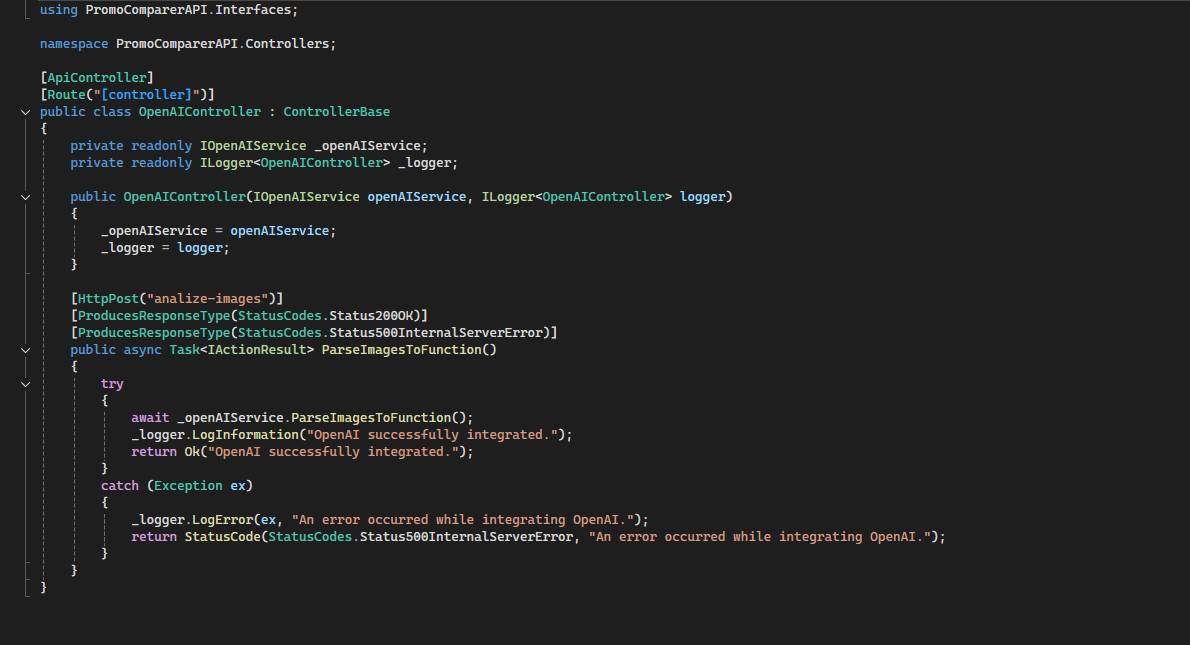
Specjalna metoda *CreateStoresFromConf*, dostępna pod POST /api/Stores/all, służy do jednorazowego załadowania konfiguracji sklepów, co może być wykorzystywane przy początkowym inicjalizowaniu bazy danych. Po pomyślnym wywołaniu klient otrzymuje status 201, a wszelkie błędy są logowane i przekazywane z kodem 500 .



Rys.7.7 Metoda CreateStoresFromConf

## 7.5. OpenAIController

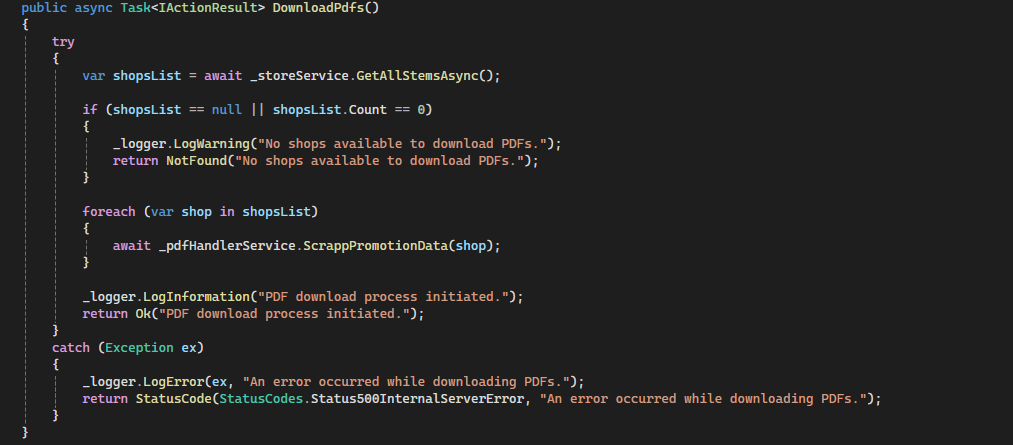
Kontroler OpenAIController skupia się na integracji z zewnętrzną usługą OpenAI. Mapowany pod ścieżkę [controller], czyli domyślnie /OpenAI, zdefiniował jedną akcję *ParseImagesToFunction*, dostępną pod POST /OpenAI/analize-images. Jej zadaniem jest wywołanie metody serwisu IOpenAIService, która odpowiada za przetwarzanie obrazów za pomocą API OpenAI. Dzięki loggerowi użytkownik otrzymuje informację zwrotną o sukcesie lub szczegóły błędu w przypadku niepowodzenia.



Rys. 7.8 Kod kontrolera OpenAIController

## 7.6. ScrapingController

*ScrapingController* pełni funkcję automatycznego pobierania i konwersji plików promocyjnych. Ścieżka bazowa to /Scraping. Metoda *DownloadPdfs*, dostępna pod *GET /Scraping/download-pdfs*, wykorzystuje serwis *IStoreService* do pozyskania listy sklepów, a następnie dla każdego wywołuje *ScrappPromotionData* serwisu *IPdfHandlerService*.

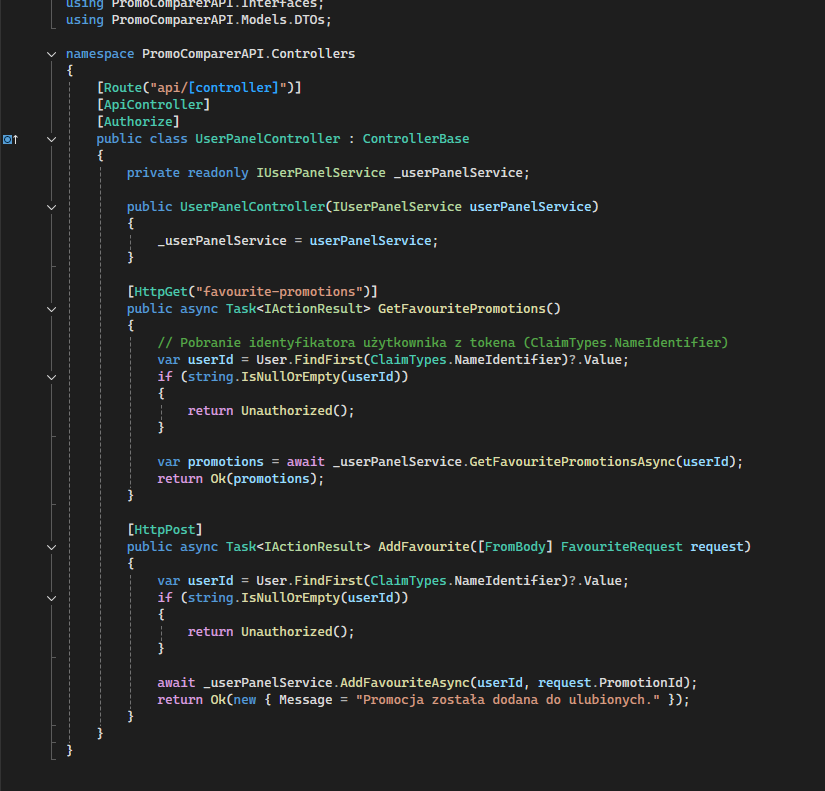


Rys. 7.9 Metoda DownloadPdfs

Dzięki logowaniu możliwe jest śledzenie braku dostępnych sklepów (404) czy błędów (500). W kolejnej akcji DownloadImages pod GET /Scraping/convert-to-images uruchamiana jest konwersja wszystkich pobranych PDF-ów na obrazy z użyciem metody ConvertAllPdfsToImagesAndDelete, a wynik operacji komunikowany jako tekstowy komunikat w odpowiedzi 200 lub – w razie problemów – 500 .

## UserPanelController

Ostatni z omawianych kontrolerów, *UserPanelController*, służy do obsługi działań użytkownika po uwierzytelnieniu. Oznaczony atrybutem [Authorize] i zmapowany pod *api/UserPanel*, udostępnia dwie akcje: *GetFavouritePromotions* oraz *AddFavourite*.



Rys. 7.10 Metody GetFavouritePromotions i AddFavourite

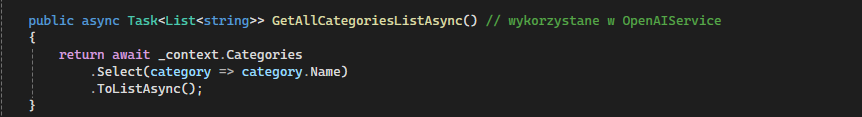
Obie metody pobierają identyfikator użytkownika z tokena JWT (klaima *ClaimTypes.NameIdentifier*). Pierwsza z nich zwraca listę ulubionych promocji w formie DTO, druga zaś pozwala na dodanie nowej promocji do ulubionych, zwracając komunikat potwierdzający powodzenie operacji. Brak klama lub jego niepoprawność skutkuje odpowiedzią 401 *Unauthorized* .

# Usługi

## 8.1. CategoryService

W serwisie *CategoryService* wstrzykiwany jest kontekst bazy danych, co pozwala na bezpośrednie operacje CRUD na encjach kategorii. W konstruktorze przypisywana jest referencja do *ApplicationDbContext*, dzięki czemu wszystkie metody pracują na wspólnym obiekcie kontekstu.

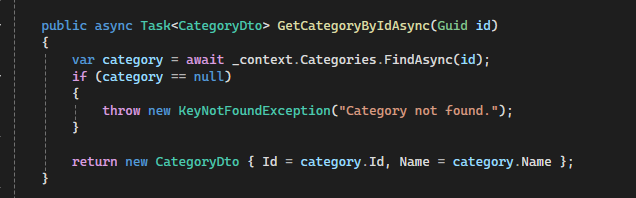
Kluczową rolę pełni metoda *GetAllCategoriesListAsync*, wykorzystywana później przez OpenAIService do dynamicznego budowania promptów. Używa ona wyłącznie projekcji nazw kategorii, co minimalizuje transfer danych.



Rys. 8.1 Metoda GetAllCategoriesListAsync

Przedstawiony sposób pobierania danych zapewnia efektywne wykorzystanie EF Core i SQL Server, ograniczając do minimum ilość przesyłanych kolumn .

Metoda *GetCategoryIdFromCategoryNameAsync* została zoptymalizowana przez dodanie *AsNoTracking*(), co wyłącza śledzenie encji przez kontekst i zmniejsza narzut pamięciowy przy odczytach, jednocześnie rzucając *KeyNotFoundException* w przypadku braku kategorii:



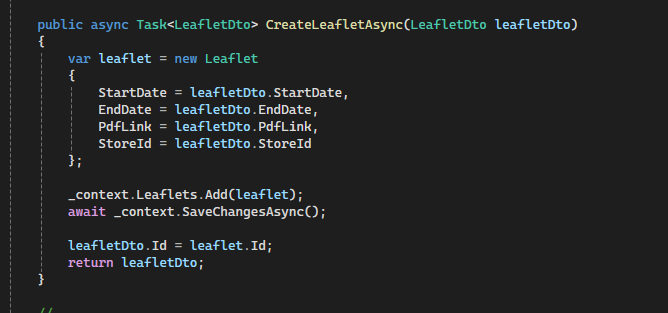
Rys. 8.2 Metoda GetCategoryIdFromCategoryNameAsync

Dzięki temu, w przypadku nieznalezienia rekordu, wyżej położona warstwa może precyzyjnie obsłużyć wyjątek, np. zwracając kod 404 do klienta .

## 8.2. LeafletService

*LeafletService* odpowiada za parsowanie dat i zarządzanie encjami ulotek w bazie. Konstruktor wstrzykuje *IStoreService*, by na podstawie aliasu sklepu uzyskać jego identyfikator.

Metoda *CreateLeafletAsync* rozdziela łańcuch zawierający zakres dat, waliduje format, a następnie konwertuje początku i końce dni na precyzyjne znaczniki czasowe („00:00:00” i „23:59:59.9999999”). Całość zapisywana jest jako nowy rekord, a wygenerowany Guid zwracany kontrolerowi



Rys. 8.3 Metoda CreateLeafletAsync

Dzięki temu kontroler nie musi zajmować się logiką parsowania dat ani walidacji formatu wejściowego .

## 8.3. PromotionService

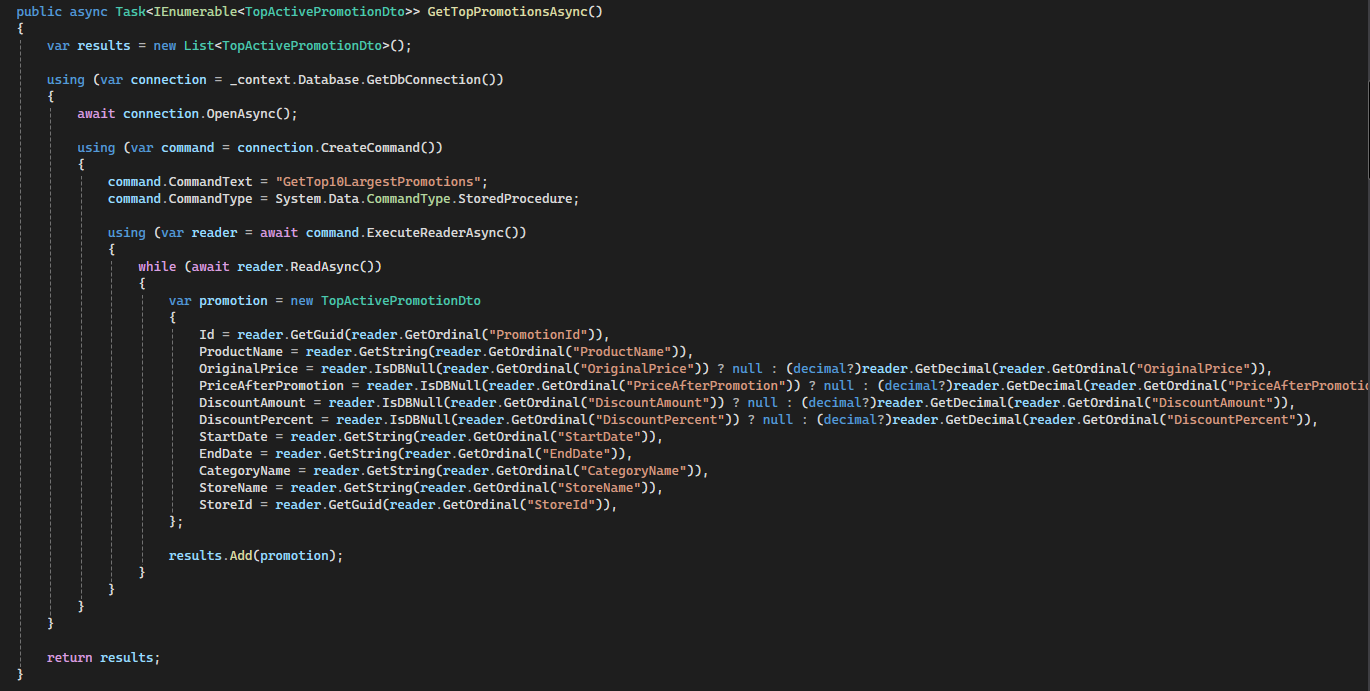
*PromotionService* agreguje promocje z kilku źródeł. Metody oparte na LINQ (*GetAllPromotionsAsync*, *GetActivePromotionsAsync*) projektują wyniki bezpośrednio do DTO, co eliminuje potrzeby mapowania w kontrolerze i zapobiega ładowaniu niepotrzebnych właściwości:



Rys. 8.4 Metody GetAllPromotionsAsync, GetActivePromotionsAsync

Takie rozwiązanie pozwala na jednoprzebiegowe zapytanie, łączące trzy tabele w SQL i przekazujące gotowe obiekty do klienta .

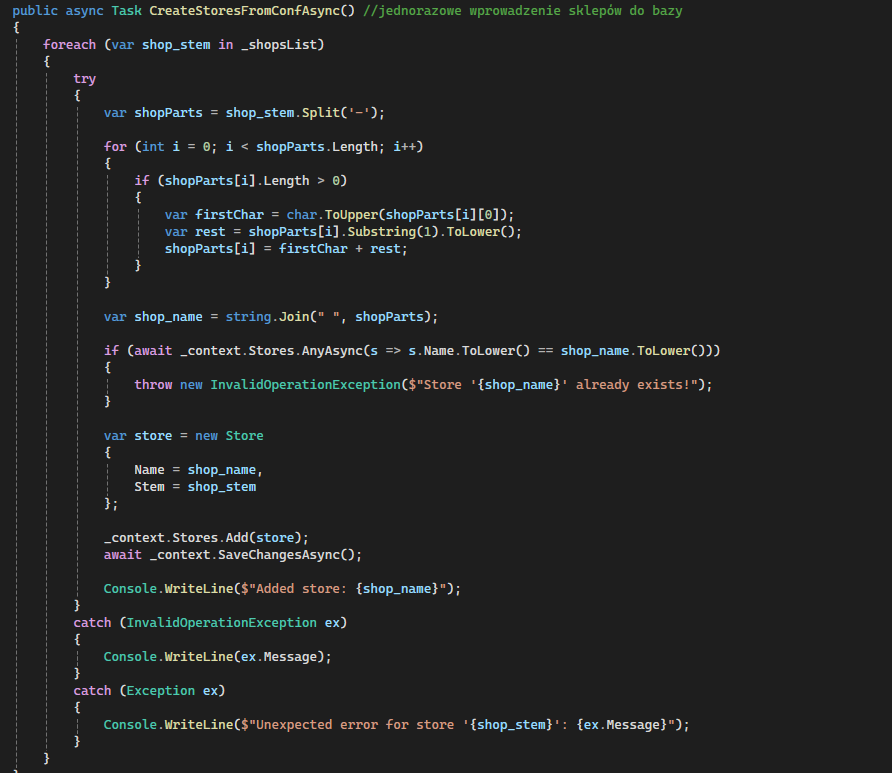
Dla bardziej zaawansowanych filtrów, jak pobranie top 10 promocji lub promocji według sklepu/kategorii, używane są procedury składowane wraz z DbConnection i SqlCommand.



Rys. 8.5 Wyznaczenie Top 10 aktywnych promocji

## 8.4. StoreService

W *StoreService* wstrzykiwany jest *IConfiguration*, z którego pobierana jest lista aliasów sklepów. Dzięki temu początkowa konfiguracja nie jest na stałe zakodowana w aplikacji, a w pliku *appsettings.json*. Metoda *CreateStoresFromConfAsync* formatuje aliasy na czytelne nazwy, sprawdza ich unikalność i następnie zapisuje.

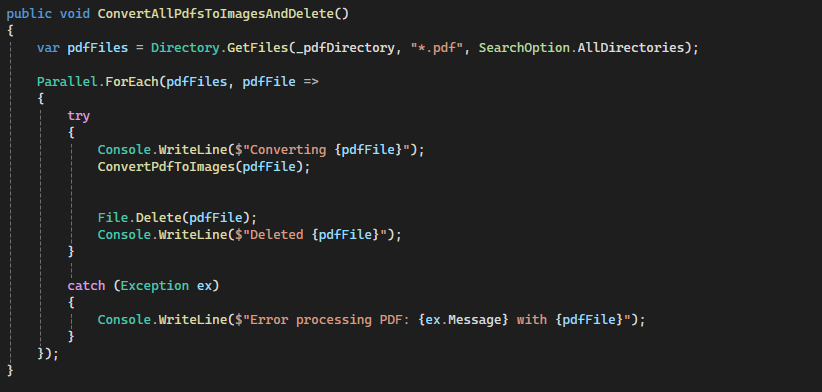


Rys. 8.6 Metoda CreateStoresFromConfAsyn

## 8.5. PdfHandlerService

*PdfHandlerService* łączy w sobie pobieranie, przetwarzanie i konwersję plików PDF. W statycznym konstruktorze ustawiana jest ścieżka do *Ghostscript*, co jest wymagane przez Magick.NET. Obiekt *HttpClient* tworzony jest raz na cały serwis, by unikać wyczerpania gniazd TCP.

Metoda *ScrappPromotionData* korzysta z *HtmlAgilityPack* do znalezienia przycisków pobierania PDF oraz odpowiadających im tekstów z datami, po czym deleguje zapis do *DownloadAndSavePdf*:

Następnie *ConvertAllPdfsToImagesAndDelete* wykonuje równoległą konwersję wszystkich PDF-ów na obrazy z wysoką rozdzielczością (300 DPI) oraz natychmiastowym usunięciem oryginałów:  Rys. 8.7 Metoda ConvertAllPdfsToImagesAndDelete

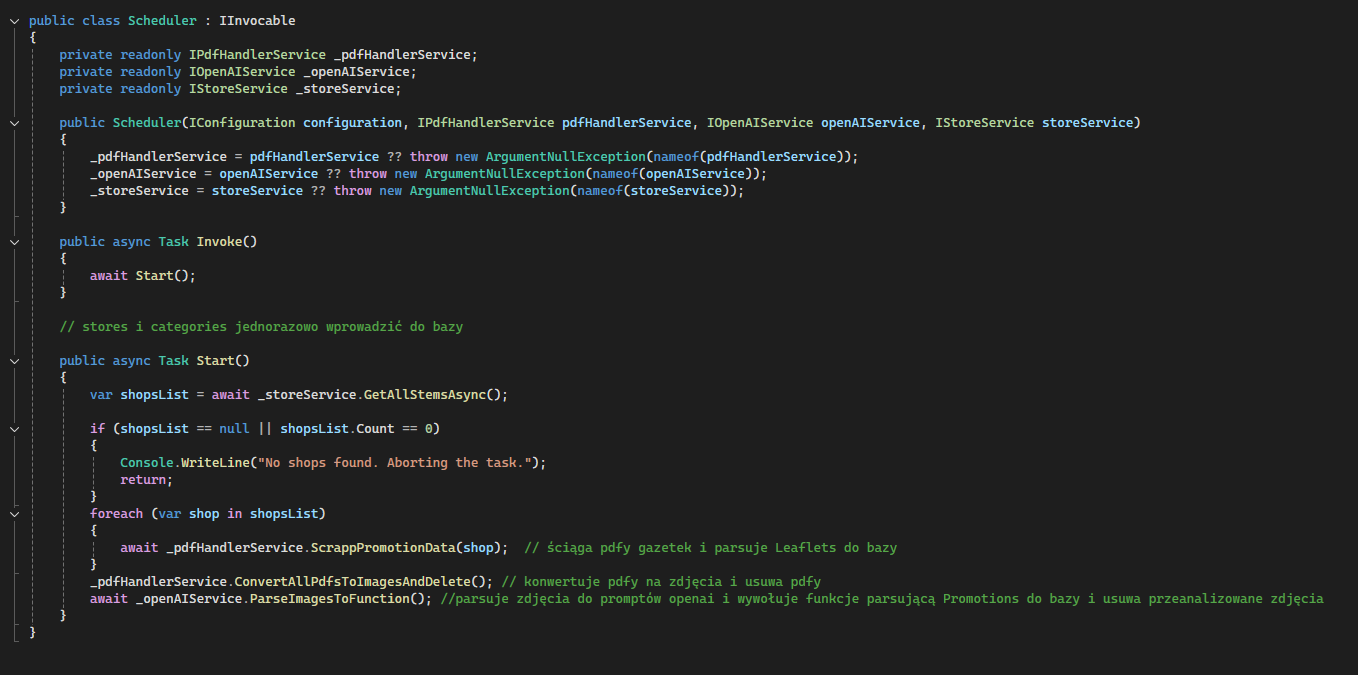
## 8.6. OpenAIService

W konstruktorze *OpenAIService* inicjalizowany jest klient ChatClient z kluczem API i modelem gpt-4o. Bezpośrednio sprawdzana jest obecność katalogu ze zdjęciami, co zapobiega późniejszym błędom.

Metoda *ParseImagesToFunction* iteruje po folderach nazwanych GUID, wczytuje każdą grafikę PNG i wywołuje prywatną metodę *GetJsonPromotions*, w której obraz jest przesyłany jako *BinaryData*. Prompt zawiera dynamicznie wstawioną listę kategorii pobraną z *CategoryService*,.

## Scheduler

Klasa *Scheduler*, implementująca *IInvocable* z pakietu *Coravel*, orkiestruje cały cykl aktualizacji danych. W konstruktorze wstrzykiwane są niezbędne serwisy, a metodzie *Start* sprawdzana jest lista sklepów, po czym kolejno wywoływane są: pobranie PDF-ów, konwersja do obrazów i analiza przez OpenAI



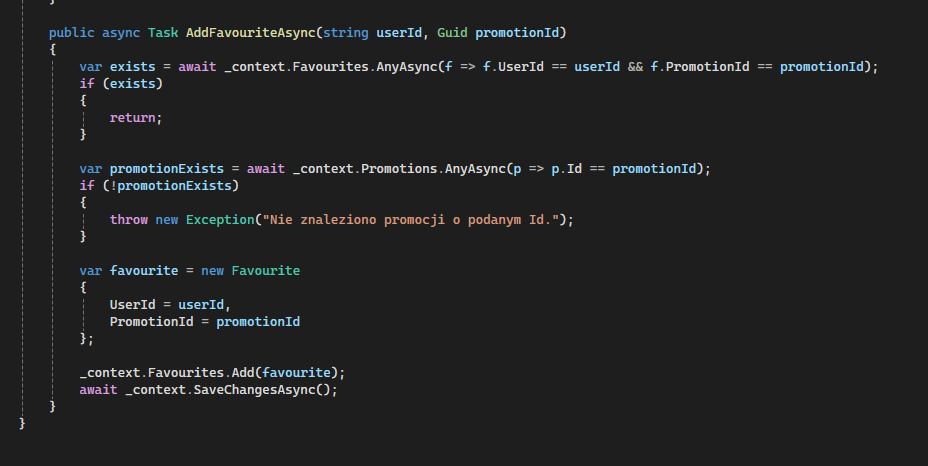
Rys. 8.8 Kod Scheduler

Dzięki temu wystarczy zaplanować wywołanie *Scheduler.Invoke* w harmonogramie, by regularnie odświeżać bazę promocji bez ręcznej ingerencji.

## UserPanelService

W *UserPanelService* wykorzystywany jest wzorzec „*Include–ThenInclude” EF Core*, by uniknąć problemu N+1 zapytań i zwrócić pełne obiekty promocji wraz z kategorią i ulotką w jednym odczycie.

Metoda *AddFavouriteAsync* najpierw weryfikuje istnienie rekordu, a w przypadku próby dodania duplikatu po prostu zwraca, zachowując idempotentność działania.



Rys. 8.9 Metoda AddFavouriteAsync