ChatGPT

# Automatyzacja skanowania kart Pokémon i integracja z API sklepu Shoper

## Przegląd rozwiązania

Aby usprawnić dodawanie wielu kart Pokémon do sklepu internetowego, proponujemy zautomatyzowany proces składający się z kilku etapów. Proces rozpoczyna się od zeskanowania karty i odczytania jej parametrów, a kończy utworzeniem produktu w sklepie Shoper poprzez API. Główne kroki integracji to:
1. **OCR karty** – Wykrycie nazwy karty oraz jej numeru z obrazu (np. za pomocą OpenAI Vision).
2. **Wyszukanie danych karty** – Pobranie szczegółowych informacji o karcie z zewnętrznego API Pokémon TCG (TCGGO/RapidAPI) na podstawie nazwy i numeru karty.
3. **Przygotowanie danych produktu** – Automatyczne uzupełnienie struktury produktu (nazwa, opis, cena, itp.) danymi karty. Stałe pola jak stawka VAT czy czas dostawy można ustawić z góry.
4. **Wysłanie produktu do Shoper** – Utworzenie nowego produktu w sklepie Shoper przez wywołanie jego REST API i przekazanie zebranych danych. W razie potrzeby dodanie zdjęcia karty do produktu.

Taka architektura (frontend + API backend w kontenerze Docker) pozwoli na przetwarzanie skanów pojedynczo lub batchowo (cały folder obrazów) z minimalną interwencją użytkownika. Poniżej omawiamy szczegóły każdego etapu wraz z typowymi problemami i ich rozwiązaniem.

## Rozpoznawanie karty z obrazu (OCR)

Pierwszym wyzwaniem jest **dokładne odczytanie z karty jej nazwy oraz numeru**. Używając mechanizmu OCR (np. OpenAI Vision za pośrednictwem API), należy przetworzyć obraz karty Pokémon tak, aby wyizolować:
- **Nazwę karty** – zwykle dużym fontem na górze karty.
- **Numer karty** – zwykle w dolnym rogu karty, w formacie `X/Y` (np. `58/102`), gdzie `X` to numer karty w serii, a `Y` to liczba kart w serii.

Aby OCR działał niezawodnie, upewnij się, że skany są **wysokiej jakości i czytelne** (odpowiednia rozdzielczość, dobre oświetlenie, brak zniekształceń). W przypadku wczytywania wielu skanów z folderu, warto zautomatyzować ich kolejkowanie do OCR. Rezultatem tego etapu powinny być wyodrębnione ciągi tekstowe – nazwa Pokémon oraz numer. Na tym etapie można rozważyć także odczytanie symbolu lub nazwy zestawu (expansion set), ponieważ sama nazwa i numer mogą niekiedy nie być unikalne. Jeśli OCR jest w stanie rozpoznać mały symbol lub skrót zestawu wydrukowany na karcie, może to pomóc zawęzić wyszukiwanie (nie jest to jednak konieczne, o czym dalej).

> **Uwaga:** Jeśli ten sam Pokémon o danym numerze pojawia się w różnych seriach (np. karta Pikachu nr 25 istnieje w kilku różnych zestawach), samo „nazwa + numer" może zwrócić wiele wyników. W takich przypadkach dodatkowe informacje (rok wydania, symbol zestawu na karcie) pomogą zidentyfikować właściwą kartę. Na ogół jednak kombinacja nazwy i pełnego numeru `X/Y` będzie jednoznaczna dla większości kart.

## Pobieranie informacji o karcie z API Pokémon TCG

Mając nazwę i numer karty, kolejnym krokiem jest wywołanie **Pokémon TCG API** (udostępnianego przez TCGGO na RapidAPI) w celu uzyskania szczegółowych danych. API to pozwala wyszukiwać karty po nazwie, numerze oraz zwraca m.in. identyfikator zestawu, nazwę zestawu, cenę rynkową oraz link do grafiki karty [1] [2]. Przed użyciem upewnij się, że:
- **Posiadasz klucz API (RapidAPI)** – Zarejestruj się w serwisie RapidAPI i zasubskrybuj API *"Pokémon TCG API"* od TCGGO. Otrzymasz `X-RapidAPI-Key` oraz nazwę hosta (`X-RapidAPI-Host`: `pokemon-tcg-api.p.rapidapi.com`), które musisz podawać w nagłówkach każdego zapytania do API.
- **Korzystasz z właściwego endpointu wyszukiwania** – API oferuje endpoint `GET /cards/search`, do którego można przekazać zapytanie tekstowe. Najprostszy sposób to użycie parametru `search` z wartością będącą połączeniem nazwy i numeru karty. Przykładowo:

```
GET https://pokemon-tcg-api.p.rapidapi.com/cards/search?search=Pikachu 58/102
```

(należy odpowiednio zakodować znaki specjalne, np. znak `"/"` w numerze). Możesz też przekazać samą nazwę, jednak dodanie numeru zawęzi wyniki. Przykład z dokumentacji API wyszukuje karty z frazą „charizard ex" w nazwie [3].

Po wywołaniu API otrzymasz odpowiedź w formacie JSON zawierającą dane pasujących kart. Jeśli zapytanie jest jednoznaczne, w wyniku będzie jedna karta (lub pierwsza na liście). Z otrzymanych danych najważniejsze będą:
- **Nazwa karty** (potwierdzenie, że znaleźliśmy właściwą kartę).
- **Zestaw** – unikalny kod lub nazwa zestawu, z którego pochodzi karta (np. *Base Set*, *Jungle*, itp.). Może się przydać do opisu lub jako element nazwy produktu.
- **Numer karty** – warto zweryfikować, czy numer `X/Y` się zgadza z odczytem OCR.
- **Cena rynkowa** – API zwraca aktualne dane cenowe (np. średnią cenę z TCGPlayer) [1]. W strukturze JSON może to być pole w obiekcie `cardmarket` lub `tcgplayer` (np. *market price*). Wybierz cenę odpowiednią dla Twojego zastosowania – np. cenę rynkową jako bazową cenę produktu w sklepie (pamiętając, że możesz ją później ręcznie dostosować).
- **Obraz karty** – link URL do obrazka karty w wysokiej rozdzielczości. Typowo API zwraca link do obrazka (np. `images.large` lub podobne pole). Ten link wykorzystamy, aby dodać zdjęcie produktu.

Jeśli API nie zwróci danych, sprawdź poprawność zapytania: czy nazwa została poprawnie odczytana (np. brak literówek) oraz czy numer karty jest przekazany w odpowiednim formacie. Czasem trzeba spróbować różnych wariantów zapytania – np. samą nazwę bez numeru lub odwrotnie. Zadbaj też o prawidłowe uwierzytelnienie przy każdym żądaniu (nagłówki RapidAPI). Przy poprawnej implementacji API Pokémon powinno zwrócić komplet informacji o karcie (baza danych obejmuje ponad 23 tysiące kart i zawiera ceny oraz zdjęcia [2]).

## Przygotowanie struktury produktu w aplikacji

Kiedy posiadamy już szczegóły karty, możemy automatycznie zbudować **strukturę produktu** zgodną z wymaganiami API Shoper. W tym etapie mapujemy dane z API Pokémon na pola produktu w sklepie. Należy uwzględnić:

- **Nazwa produktu (`name`)** – Możesz użyć nazwy Pokémon jako nazwy produktu. Aby produkty były unikalne i bardziej opisowe, warto rozważyć dodanie do nazwy np. zestawu lub numeru

karty. Przykładowo: `"Pikachu 58/102 Base Set"` lub `"Pikachu – Base Set (58/102)"`. Dzięki temu w katalogu sklepu produkty będą łatwiejsze do rozróżnienia.

- **Opis (`description`)** – Opis produktu można wygenerować na podstawie danych karty. Na przykład: *„Karta kolekcjonerska Pokémon Pikachu z zestawu Base Set (1999), numer 58/102. Stan: mint/near mint."* Możesz uwzględnić rzadkość karty (Common/Uncommon/Rare etc., jeśli zwraca ją API) i inne ciekawostki. Część informacji (np. rok wydania, edycja pierwsza czy nie) może wymagać własnej logiki lub uzupełnienia ręcznego, ale podstawowe dane wyciągnięte z API pozwolą stworzyć solidny opis.
- **Kategoria produktu** – Jeśli Twój sklep Shoper używa kategorii, warto automatycznie przypisać nowy produkt do odpowiedniej kategorii. Możesz mieć np. kategorię „Pokémon TCG" lub osobne podkategorie dla każdej serii. W API Shoper przypisanie kategorii odbywa się zwykle przez podanie `category_id` (lub listy kategorii). Ustal z góry ID kategorii domyślnej dla kart Pokémon (można je znaleźć w panelu Shoper lub przez endpoint API kategorii) i dodaj do produktu.
- **Cena (`price`)** – Ceny pobrane z API wstaw jako cenę produktu. Shoper może wymagać ceny brutto lub netto i powiązania z VAT, w zależności od konfiguracji sklepu. Najprościej podać cenę brutto i wskazać właściwą stawkę VAT (patrz niżej). Upewnij się, że format liczbowy jest zgodny z oczekiwaniami API (użyj kropki jako separatora dziesiętnego, np. 12.34).
- **Stawka VAT (`tax_id`)** – To pole prawdopodobnie jest wymagane. Określa, jaką stawkę podatku przypisać do produktu (np. 23% VAT). Shoper identyfikuje stawki poprzez `tax_id` – należy podać identyfikator istniejącej stawki zdefiniowanej w sklepie. Jeśli wszystkie karty mają tę samą stawkę (np. 23%), możesz ją traktować jako stałą i zawsze ustawiać ten sam `tax_id`. Sprawdź w dokumentacji lub poprzez API listę stawek, aby znaleźć właściwe ID.
- **Stan magazynowy (`stock`)** – Tutaj ważne są dwie rzeczy: ilość oraz cena (tak, cena w Shoper jest często przekazywana właśnie w kontekście stanu magazynowego). Shoper przechowuje informacje o wariantach/stanach magazynowych produktu w strukturze *stock*. Dla prostego produktu (bez wariantów) będziesz mieć jeden wpis stock określający cenę i dostępność. W JSON warto to zorganizować jako obiekt, np.:

```
"stock": {
    "price": 45.00,
    "stock": 1
}
```

  gdzie `price` to cena brutto produktu, a `stock` to liczba sztuk w magazynie (zakładamy, że masz po 1 szt. z każdej karty). Powyższy przykład wskazuje, że produkt kosztuje 45.00 (waluta domyślna sklepu, np. PLN) i jest dostępna 1 sztuka. **Uwaga:** Nazewnictwo pola `stock.stock` jest nieintuicyjne – w praktyce pierwsze `stock` to sekcja danych, a drugie oznacza ilość sztuk. Taka struktura wynika z modelu Shoper.
- **Dostępność i czas dostawy** – Shoper rozróżnia status dostępności produktu oraz termin wysyłki.
- **Status dostępności (`availability_id`)** określa, czy produkt jest dostępny od ręki, na zamówienie, wyprzedany itp. Dla kart, które posiadasz, ustaw status jako dostępny (najpewniej jest to domyślna dostępność o ID odpowiadającym np. „w magazynie").
- **Czas wysyłki (`delivery_id`)** to pole reprezentujące termin realizacji (np. „Wysyłka w 24h", „Wysyłka w 3-5 dni" itp.). Skoro chcesz ustawić stały czas dostawy, wybierz z listy w panelu Shoper ten, który pasuje (np. 24 godziny) i zanotuj jego ID. Następnie każde nowe API call do utworzenia produktu powinno zawierać `delivery_id` ze stałą wartością. Te pola są dostępne w API Shoper – w dokumentacji widać m.in. `availability_id` oraz `delivery_id` jako wymagane atrybuty produktu [4].

- **Jednostka miary (`unit_id`)** – Jeśli w Twoim sklepie wymagane jest podanie jednostki (np. sztuka, opakowanie itp.), upewnij się, że przekazujesz odpowiedni `unit_id`. Dla sztuk może to być np. 1 (standardowa jednostka „szt.”). Jeśli nie podasz jednostki, API może zwrócić błąd, dlatego lepiej ją uwzględnić, nawet jeśli zawsze taka sama.
- **Inne pola stałe** – W zależności od ustawień sklepu możesz ustawić dodatkowo np. producenta (`producer_id` – jeśli chcesz, możesz utworzyć producenta „Nintendo” lub „Pokémon” i przypisać do kart), ewentualnie wagę produktu (`weight`, jeśli istotna dla wysyłki – pojedyncza karta waży bardzo niewiele, np. 0.01 kg, ale można ustawić minimalną wartość lub zero).

Po zmapowaniu danych utwórz obiekt (np. słownik w Pythonie lub JSON w formie stringu) zgodny z wymaganiami Shoper. Przykładowy JSON nowego produktu może wyglądać tak (z wypełnionymi polami przykładowymi danymi):

```
{
  "name": "Pikachu – Base Set (58/102)",
  "description": "Karta kolekcjonerska Pokémon Pikachu z zestawu Base Set
(1999), numer 58/102. Stan: near mint.",
  "category_id": 12,
  "tax_id": 1,
  "unit_id": 1,
  "availability_id": 1,
  "delivery_id": 3,
  "producer_id": null,
  "stock": {
    "price": 45.00,
    "stock": 1
  }
}
```

Oczywiście wartości `category_id`, `tax_id`, `availability_id`, `delivery_id` dostosuj do własnej konfiguracji sklepu. Obecność powyższych pól powinna zapobiec błędom typu „brak wymaganego pola” podczas tworzenia produktu.

## Wysłanie produktu do sklepu Shoper przez API

Ostatni krok to skorzystanie z **API REST Shoper** do utworzenia produktu. Zakładamy, że masz już zarejestrowaną aplikację/klienta w Shoper (w panelu administracyjnym Shoper generuje się `client_id` i `client_secret`). Podstawowe kroki komunikacji z API Shoper to:

1. **Autoryzacja (OAuth)** – zanim wyślesz jakiekolwiek dane, musisz uzyskać token dostępu. Wywołaj endpoint tokenu (np. `POST /webapi/rest/auth` lub `/webapi/rest/oauth/token` – szczegóły w dokumentacji Shoper) z danymi klienta. Shoper stosuje OAuth2, więc żądanie powinno zawierać `grant_type`, `client_id`, `client_secret` itp. Po poprawnym uwierzytelnieniu otrzymasz `access_token` (i opcjonalnie `refresh_token`). Pamiętaj, że token dostępu jest ważny przez pewien czas (np. 90 dni [5]). Możesz go ponownie używać do wielu operacji bez konieczności logowania za każdym razem, aż wygaśnie.
2. **Przygotowanie żądania POST** – endpoint do tworzenia produktu to:

```
POST https://<adres Twojego sklepu>/webapi/rest/products
```

(np. `https://mojsklep.shoparena.pl/webapi/rest/products`). Nie przekazujemy parametrów w URL – wszystkie dane produktu wyślemy w body jako JSON. Ustaw odpowiednie nagłówki:

3. `Authorization: Bearer <access_token>` – token uzyskany w kroku 1. **To jest kluczowe**: każdy request do API (poza uzyskaniem tokenu) musi nieść nagłówek autoryzacji z tokenem typu Bearer [6]. Brak lub niepoprawny token spowoduje błąd `unauthorized_client`. Upewnij się, że nie wysyłasz już przy zapytaniu z tokenem klient_id ani secret – wystarczy sam token w nagłówku [7].
4. `Content-Type: application/json` – dane wysyłamy w formacie JSON.
5. **Wysłanie danych produktu** – do body żądania dołącz skonstruowany wcześniej JSON z informacjami o produkcie. Jeżeli używasz np. biblioteki *requests* w Pythonie, przekaż `json=product_data` (co automatycznie serializuje słownik do JSON i ustawia content-type). W przypadku innych języków analogicznie.
6. **Sprawdzenie odpowiedzi** – jeśli wszystko pójdzie dobrze, API Shoper zwróci obiekt nowo utworzonego produktu (lub przynajmniej status 201 Created z informacją o zasobie). Możesz w odpowiedzi znaleźć `product_id` utworzonego produktu – zapisz go, przyda się do dodania obrazka.

**Typowe błędy przy tworzeniu produktu i ich rozwiązanie:**
- Błąd autoryzacji (`401 unauthorized` lub komunikat *unauthorized_client*): oznacza to, że token jest nieprawidłowy lub nie został podany. Rozwiązanie: upewnij się, że do **każdego** wywołania (po zalogowaniu) dodajesz nagłówek `Authorization: Bearer <token>` [6]. Sprawdź też, czy token nie wygasł – jeśli tak, użyj refresh_token lub zaloguj się ponownie.
- Błąd walidacji danych (`400 Bad Request`): Shoper zwróci w polu `error_description` informacje, co poszło nie tak. Częste przyczyny: brak wymaganych pól lub zła struktura JSON. Przykładowo komunikat *„Decoding failed: Syntax error”* sugeruje, że JSON jest niepoprawny składniowo lub pola nie zgadzają się z oczekiwaniami [8] [9]. Rozwiązanie: upewnij się, że JSON jest prawidłowo sformatowany (np. użyj funkcji serializujących zamiast składać ręcznie string), że nazwy pól są dokładnie takie jak w dokumentacji, a wartości mają odpowiedni typ.
- Brak wymaganej kategorii/parametru: Jeśli API zwróci błąd, że np. *category_id* czy inny parametr jest wymagany, musisz go dodać do danych. W razie wątpliwości, które pola są obowiązkowe, przejrzyj dokumentację Shoper dla tworzenia produktów lub spróbuj utworzyć produkt z minimalnym zestawem danych przez panel administracyjny i sprawdź, co jest konieczne (np. kategoria może być obowiązkowa w niektórych konfiguracjach sklepu).

Po wyeliminowaniu błędów powinieneś osiągnąć automatyczne tworzenie produktów. Nowo dodane produkty pojawią się w katalogu Twojego sklepu Shoper z uzupełnionymi polami (nazwą, opisem, ceną, itd.). Na tym etapie większość danych jest już na swoim miejscu – pozostaje **dodanie obrazka karty**.

## Dodawanie grafiki karty jako zdjęcia produktu

Grafika karty uzyskana z API Pokémon (URL do obrazka) powinna zostać dodana jako zdjęcie produktu w sklepie. Shoper API udostępnia do tego osobny zasób **Product Images**. Możliwe podejścia:
- **Dodanie obrazu przez URL**: jeśli Shoper API akceptuje bezpośrednio URL (należałoby to potwierdzić w dokumentacji), mógłbyś przekazać w strukturze produktu listę obrazów, gdzie każdy ma pole URL. W dokumentacji jest wzmianka, że *product.create* może przyjmować pole *images* (tablica) [10], jednak format nie jest tam opisany. Być może jest to zarezerwowane dla przesyłania danych binarnych lub ID istniejących grafik.

- **Dodanie obrazu przez osobny endpoint**: Bezpieczniejsza metoda to najpierw utworzyć produkt (co zwraca *product_id*), a następnie wykonać upload obrazka pod endpointem `POST /webapi/rest/product-images`. W body należałoby przesłać binarną zawartość pliku lub dane Base64. Możesz pobrać obraz z URL (np. za pomocą żądania HTTP w swojej aplikacji) i następnie wysłać go do Shoper. Zwróć uwagę na parametry: najprawdopodobniej trzeba wskazać `product_id`, czy obraz ma być głównym (`main`: true/false), oraz oczywiście sam plik (być może w formacie multipart/form-data). Sprawdź dokumentację: zarys wskazuje, że produkt może mieć do 255 obrazów, a obiekt obrazu ma pola m.in. `gfx_id`, `product_id`, `main` itp. [11].

Jeśli zdecydujesz się na drugie podejście, scenariusz jest następujący: po stworzeniu produktu wywołujesz np.:

```
POST /webapi/rest/product-images
```

z body zawierającym plik (obraz karty) oraz atrybuty `product_id` (ID produktu z Shoper) i np. `main: true` (aby ustawić to zdjęcie jako główne). Nie zapomnij o autoryzacji Bearer w nagłówku. W odpowiedzi powinieneś otrzymać potwierdzenie dodania obrazka (lub nowy `gfx_id`). Po odświeżeniu strony produktu w panelu/adminie sklepu, zobaczysz dodane zdjęcie.

**Uwaga:** Dodanie obrazka zwiększa złożoność – możesz zaimplementować je od razu lub potraktować jako etap drugi (najpierw weryfikując, że tworzenie produktów bez obrazków działa). Jeśli API product-images sprawiałoby kłopoty, alternatywnie można wykorzystać import przez panel lub za pomocą integracji z URL, ale najlepiej trzymać się REST API dla pełnej automatyzacji.

## Przetwarzanie wielu skanów i wydajność

Skoro aplikacja ma obsługiwać dodawanie wielu kart (skany pojedynczo lub całymi seriami), warto zadbać o wydajność i niezawodność procesu:
- **Kolejkowanie zadań** – Przy wczytaniu folderu z wieloma obrazami, rozważ kolejkę zadań: np. dla każdego obrazu wywołaj kolejno OCR -> API Pokémon -> API Shoper. Unikniesz w ten sposób jednoczesnego wysyłania wielu żądań, co mogłoby przeciążyć API zewnętrzne lub spowodować ograniczenie (rate limit).
- **Limity API** – Zapoznaj się z limitami: darmowy plan Pokémon TCG API przez RapidAPI ma ograniczenia (np. liczba zapytań na minutę/dzień). Jeśli masz bardzo dużo kart, być może konieczne będzie wykupienie wyższego planu lub wprowadzenie opóźnień między żądaniami, by nie przekroczyć limitów. Podobnie API Shoper może mieć ograniczenia co do liczby operacji na sekundę – sprawdź dokumentację lub po prostu implementuj drobne opóźnienie (np. 1 sekundę) między tworzeniem kolejnych produktów, aby serwer sklepu nadążał je przetwarzać.
- **Logowanie i obsługa błędów** – W procesie masowego dodawania dobrze jest logować wyniki poszczególnych kroków (np. „Karta Pikachu 58/102 – dodano OK, product_id=12345" lub „Błąd: karta Bulbasaur nieznaleziona w API" itp.). Ułatwi to identyfikację ewentualnych problemów z konkretnymi kartami bez przerywania całego procesu. Możesz np. pominąć kartę, której nie udało się przetworzyć, i przejść do następnej, a na koniec wygenerować raport.
- **Weryfikacja danych** – Automatyzacja oszczędzi mnóstwo czasu, ale dobrze jest wprowadzić minimalną weryfikację: np. czy cena nie jest zerowa, czy nazwa produktu nie jest pustym stringiem (gdyby OCR lub API zawiodło). W razie wykrycia braków można oflagować taki produkt do ręcznej interwencji zamiast dodawać błędne dane do sklepu.

## Podsumowanie

Łącząc możliwości OCR i API, można niemal bezobsługowo przenieść informacje ze **skanu karty Pokémon do produktu w sklepie Shoper**. Kluczem jest poprawna integracja z dwoma API zewnętrznymi i odpowiednie mapowanie danych. Twoja aplikacja webowa (działająca w Dockerze) powinna:
- **Rozpoznawać tekst z obrazów** (OpenAI Vision) – uzyskując nazwę i numer karty.
- **Wykorzystywać API Pokémon TCG** do pobrania szczegółów (zestaw, cena, obraz) – upewnij się, że używasz poprawnych endpointów i uwierzytelnienia RapidAPI, aby otrzymać komplet danych [3].
- **Tworzyć obiekty produktów** zgodnie z wymaganiami Shoper – uzupełniając wszystkie niezbędne pola (nazwa, opis, kategoria, cena, VAT, dostępność, czas dostawy, stan magazynowy itp.) [4].
- **Wysyłać dane do Shoper** przez REST API z tokenem Bearer – połączenie musi być autoryzowane, a dane poprawnie sformatowane w JSON [7] [6].
- **Dodawać zdjęcia** do produktów – poprzez odpowiedni endpoint, korzystając z linku do obrazka uzyskanego z API karty.

Dopracowanie każdego z tych kroków zapewni płynne działanie całego procesu. W efekcie po załadowaniu czytelnego skanu karta niemal od razu pojawi się w Twoim sklepie jako gotowy produkt, bez żmudnego ręcznego przepisywania danych. Automatyzacja zaoszczędzi mnóstwo czasu przy dużej liczbie kart, a dzięki aktualnym danym z API (np. cenom rynkowym) Twoje opisy i ceny będą na bieżąco. Powodzenia w implementacji! Jeśli napotkasz problemy, pomocne może być sięgnięcie do dokumentacji Shoper (rozdziały o produktach i ich polach) oraz społeczności developerów (fora, GitHub) w poszukiwaniu przykładów integracji. Dzięki temu w krótkim czasie usprawnisz dodawanie produktów do swojego sklepu i zyskasz przewagę czasową nad ręcznym wprowadzaniem setek kart.

**Źródła:** Korzystałem z oficjalnych informacji o API Pokémon TCG (RapidAPI) oraz dokumentacji Shoper i doświadczeń developerów. M.in. wątki na forum 4programmers dotyczące uwierzytelniania i tworzenia produktów przez API Shoper [7] [6], jak również fragmenty dokumentacji Shoper potwierdzające istnienie kluczowych pól (availability_id, delivery_id itp.) wymaganych przy tworzeniu produktu [4]. Informacje o zakresie danych dostępnych w Pokémon TCG API (ceny, obrazy) potwierdzają wpisy w dokumentacji i ogłoszeniach o wersji 2 API [1] [2]. Te materiały pomogły w przygotowaniu powyższego planu integracji. Powodzenia we wdrożeniu!

---

1 Version 2 of the Pokémon TCG API released! : r/pkmntcg - Reddit
https://www.reddit.com/r/pkmntcg/comments/lfofci/version_2_of_the_pok%C3%A9mon_tcg_api_released/

2 Pokémon TCG API
https://pokemontcg.io/

3 PokemonTCG API - The Ultimate TCG Pricing API
https://pokemon-api.com/

4 Products: update - Shoper Developers
https://developers.shoper.pl/developers/api/resources/products/update

5 6 7 Shoper i REST API :: 4programmers.net
https://4programmers.net/Forum/C_i_.NET/322855-shoper_i_rest_api

8 9 Jak pobrać konkretny produkt z bazy Shopera :: 4programmers.net
https://4programmers.net/Forum/Python/365839-jak_pobrac_konkretny_produkt_z_bazy_shopera

10 product.create - Shoper Developers
https://developers.shoper.pl/developers/webhooks/methods/product-create

11 Product Images - Shoper Developers
https://developers.shoper.pl/developers/api/resources/product-images