Architektura i Strategia Implementacji Tożsamości Produktowej w Ekosystemie API Shoper: Kompleksowy Raport Techniczny

Streszczenie Wykonawcze

Niniejszy raport stanowi wyczerpującą analizę techniczną architektury interfejsu programistycznego (API) platformy Shoper, ze szczególnym uwzględnieniem mechanizmów nadawania i zarządzania identyfikatorami produktów: **Kodem Produktu** (code) oraz **Kodem Magazynowym** (stock.code). W dobie rosnącej złożoności systemów e-commerce, gdzie platformy sprzedażowe są ściśle zintegrowane z zewnętrznymi systemami zarządzania magazynem (WMS), planowania zasobów przedsiębiorstwa (ERP) oraz systemami Product Information Management (PIM), precyzyjne odwzorowanie tożsamości produktu staje się krytycznym wyzwaniem inżynieryjnym.

Analiza ta wykracza poza powierzchowne omówienie dokumentacji, zagłębiając się w ontologię danych platformy Shoper. Wykazuje ona fundamentalne rozróżnienie pomiędzy bytem katalogowym (Produktem) a bytem logistycznym (Zasobem/Stockiem). Raport ten systematyzuje wiedzę na temat struktur JSON, zależności między zasobami products i product-stocks, a także omawia zaawansowane scenariusze obsługi wariantów, polityki cenowej oraz zgodności z regulacjami prawnymi (kody GTU, BLOZ). Dokument ten jest przeznaczony dla architektów systemowych, starszych programistów backendowych oraz integratorów systemów, którzy poszukują definitywnego źródła wiedzy na temat poprawnej implementacji cyklu życia produktu w środowisku Shoper REST API.

---

1. Fundamenty Architektoniczne Tożsamości Produktowej w Środowisku Shoper

Aby w pełni zrozumieć mechanikę nadawania kodów w API Shoper, konieczne jest najpierw zgłębienie filozofii architektury danych, która leży u podstaw tej platformy. W przeciwieństwie do prostszych systemów e-commerce, które często traktują produkt jako płaską strukturę danych, Shoper adoptuje model relacyjny, w którym "Produkt" jest kontenerem semantycznym, a "Stock" (stan magazynowy) jest bytem operacyjnym. Ta dychotomia jest kluczowa dla zrozumienia, dlaczego i w jaki sposób system wymaga odrębnych, choć powiązanych, identyfikatorów.

1.1 Dualizm Identyfikacyjny: Koncepcja Produktu i Zasobu

W ekosystemie Shoper tożsamość towaru nie jest monolityczna. System rozróżnia tożsamość handlową (marketingową) od tożsamości logistycznej. Jest to podejście odzwierciedlające rzeczywiste procesy biznesowe w dużych przedsiębiorstwach handlowych, gdzie dział marketingu operuje na poziomie SKU (Stock Keeping Unit) handlowego, podczas gdy dział logistyki operuje na poziomie kodów EAN lub lokalizacji magazynowych.

1.1.1 Kod Produktu (code) – Tożsamość Katalogowa

Kod produktu, umiejscowiony w głównym korzeniu zasobu products, pełni rolę nadrzędnego identyfikatora w katalogu sklepu.[1] Jest to ciąg znaków (string), który:

- **Definiuje unikalność w przestrzeni nazw sklepu:** System nie zezwala na istnienie dwóch produktów o tym samym kodzie głównym, co wymusza na integratorze stosowanie unikalnych kluczy.
- **Jest widoczny dla klienta końcowego:** Często jest prezentowany na karcie produktu jako "Nr katalogowy" lub "Symbol".
- **Służy do indeksowania i wyszukiwania:** Jest kluczowym polem dla silników wyszukiwarek wewnętrznych oraz zewnętrznych integracji SEO.
- **Jest polem obligatoryjnym:** Nie jest możliwe utworzenie produktu poprzez API bez zdefiniowania tego pola.

W kontekście integracji z systemami ERP, pole to zazwyczaj mapowane jest na "Indeks Towarowy" lub "Główny SKU".

1.1.2 Kod Magazynowy (stock.code lub code w product-stocks) – Tożsamość Logistyczna

Kod magazynowy to identyfikator niższego rzędu, ściśle powiązany z fizyczną dostępnością towaru. Jego lokalizacja w strukturze JSON zależy od kontekstu (czy jest to produkt prosty, czy wariantowy), co jest częstym źródłem błędów implementacyjnych.[2]

- **Definiuje wariant logistyczny:** W przypadku produktu prostego (np. książka), kod magazynowy może być tożsamy z kodem produktu. Jednak w przypadku produktu z wariantami (np. obuwie), każdy rozmiar posiada odrębny kod magazynowy, mimo że wszystkie dzielą ten sam kod produktu.
- **Jest kluczem dla operacji magazynowych:** To ten kod jest skanowany przez kolektory danych w magazynie podczas kompletacji zamówienia.
- **Jest widoczny na dokumentach sprzedażowych:** Faktury, dokumenty WZ (Wydanie Zewnętrzne) generowane przez Shoper zazwyczaj referują do tego kodu, aby precyzyjnie określić, która wersja produktu została sprzedana.

1.2 Model Zasobów i Ich Interpendenja

Architektura REST API Shopera opiera się na separacji odpowiedzialności pomiędzy zasobami. Zrozumienie tej separacji jest niezbędne do uniknięcia zjawiska "Widmowych Zasobów" – sytuacji, w której produkt istnieje w katalogu, ale nie posiada fizycznej reprezentacji magazynowej, co uniemożliwia jego sprzedaż.

1. **Zasób products (/webapi/rest/products)**: Jest to punkt wejścia dla procesu kreacji. Metoda insert na tym zasobie tworzy szkielet produktu oraz jego podstawowy stan magazynowy.[4] To tutaj definiujemy nazwę, opis, kategorię oraz domyślny kod produktu.
2. **Zasób product-stocks (/webapi/rest/product-stocks)**: Jest to wyspecjalizowany zasób do zarządzania wariantami magazynowymi.[2] W przypadku produktów konfigurowalnych, główny zasób products jedynie inicjuje istnienie towaru, natomiast konkretne warianty (np. Czerwony-L, Czerwony-XL) są tworzone i zarządzane jako osobne rekordy w product-stocks.

1.3 Flaga Sterująca extended: Przełącznik Paradygmatu

Analiza dokumentacji ujawnia krytyczny parametr sterujący: flagę logiczną stock.extended.[6]

Jest to "zwrotnica" architektoniczna, która determinuje, w jaki sposób system interpretuje dane magazynowe.

- **Tryb Podstawowy (extended = 0 / false)**: Produkt jest monolitem. Posiada jedną cenę, jeden stan magazynowy i jeden kod magazynowy. W tym trybie obiekt stock zagnieżdżony w żądaniu utworzenia produktu jest wystarczający i kompletny.
- **Tryb Rozszerzony (extended = 1 / true)**: Produkt staje się kontenerem dla wariantów. Informacje w obiekcie stock w głównym żądaniu products stają się jedynie szablonem lub wartością domyślną. Rzeczywiste stany magazynowe są "delegowane" do pod-zasobów zarządzanych przez product-stocks. W tym scenariuszu kod magazynowy zdefiniowany w produkcie głównym traci na znaczeniu operacyjnym na rzecz kodów zdefiniowanych w poszczególnych wariantach.

---

2. Konfiguracja Środowiska i Kontekst Bezpieczeństwa

Zanim przystąpimy do analizy struktury JSON, należy omówić wymogi bezpieczeństwa niezbędne do przeprowadzenia operacji zapisu. API Shoper jest chronione protokołem OAuth 2.0, a każda operacja wymaga odpowiednich uprawnień (scopes).

2.1 Uprawnienia (API Scopes)

Zgodnie z dokumentacją [3], operacja tworzenia produktu (insert) oraz manipulacji jego stanami magazynowymi nie jest dostępna dla każdego tokena dostępowego. Aplikacja integrująca musi podczas procesu autoryzacji (handshake) zażądać następujących zakresów uprawnień:

1. **products_create**: Uprawnienie absolutnie niezbędne. Pozwala na wysyłanie żądań POST do zasobu products. Bez tego uprawnienia serwer zwróci błąd HTTP 403 Forbidden.
2. **products_edit**: Konieczne w przypadku scenariusza dwuetapowego (utworzenie produktu -> aktualizacja wariantów). Pozwala na modyfikację istniejących rekordów.
3. **products_read**: Zalecane do weryfikacji poprawności utworzenia zasobu oraz sprawdzania unikalności kodów przed próbą zapisu (pre-validation).

2.2 Biblioteki Klienckie vs. Raw REST

Dokumentacja [1] wskazuje na istnienie oficjalnej biblioteki PHP DreamCommerce\ShopAppstoreLib. Użycie dedykowanego SDK znacząco upraszcza proces, abstrahując warstwę transportową HTTP i mechanizmy autoryzacji.

- **Korzyść z użycia SDK**: Automatyczna obsługa odświeżania tokenów (refresh token), co jest kluczowe przy długotrwałych procesach importu produktów. Obsługa wyjątków DreamCommerce\Exceptions\ClientException pozwala na bardziej granularne reagowanie na błędy walidacji API niż ręczne parsowanie kodów HTTP.

---

3. Szczegółowa Analiza Obiektu Produktu: Anatomia Żądania products: insert

Tworzenie produktu w Shoper API to proces składania precyzyjnego dokumentu JSON. Błąd w strukturze lub typie danych któregokolwiek z pól może skutkować odrzuceniem całego żądania. Poniższa sekcja dekonstruuje strukturę żądania, koncentrując się na polach odpowiedzialnych za identyfikację.

3.1 Struktura Korzenia (Root Object)

Na najwyższym poziomie struktury JSON definiujemy tożsamość katalogową produktu.

JSON

```
{
  "code": "SKU-MAIN-001",
  "category_id": 15,
  "pkwiu": "12.34.56.0",
  "unit_id": 1,
  "producer_id": 5,
...
}
```

Analiza Pola code [1]:

- **Typ Danych**: String.
- **Wymagalność**: Tak. Jest to pole obligatoryjne.
- **Walidacja**: System sprawdza unikalność w obrębie sklepu. Próba użycia istniejącego kodu spowoduje błąd.
- **Dobre Praktyki**: Zaleca się stosowanie znaków alfanumerycznych, myślników i podkreśleń. Należy unikać spacji i znaków specjalnych, które mogą powodować problemy przy generowaniu adresów URL (tzw. slug), jeśli konfiguracja sklepu wykorzystuje kod produktu w linkach.

3.2 Obiekt stock: Serce Tożsamości Logistycznej

To właśnie w obiekcie stock dochodzi do najczęstszych nieporozumień. Jest to obiekt zagnieżdżony wewnątrz głównego obiektu produktu. Nie należy mylić go z zasobem product-stocks, choć strukturalnie są one bliźniacze.

JSON

```
{
 ...
  "stock": {
     "code": "WH-LOC-A1",
     "price": 100.00,
     "active": true,
     "stock": 50,
     "weight": 1.5,
     "additional_codes": {... }
  }
 ...
}
```

Analiza Pola stock.code [3]:

- **Lokalizacja**: root -> stock -> code.
- **Typ Danych**: String.
- **Semantyka**: Jest to kod wariantu podstawowego. Jeśli produkt nie ma wariantów, jest to jedyny kod magazynowy.
- **Znaczenie Operacyjne**: W integracjach z kurierami (np. InPost, DPD) ten kod jest

przekazywany jako identyfikator towaru na liście przewozowej. Jego brak lub błędna wartość może zablokować automatyzację wysyłek.

3.3 Kody Dodatkowe (additional_codes): Kontekst Regulacyjny

Shoper oferuje bogaty zestaw pól dla specjalistycznych identyfikatorów w obiekcie additional_codes.[1] Wymagają one szczegółowego omówienia, gdyż często są mylnie utożsamiane z kodem magazynowym.

Pole	Typ	Opis i Kontekst Biznesowy	Źródło
ean	String	Europejski Kod Towarowy (EAN-13) lub UPC. Kluczowy dla integracji z Google Merchant Center (GTIN) oraz marketplace'ami (Allegro, Amazon). Znajduje się bezpośrednio w obiekcie stock, nie w additional_codes.	[3]
isbn	String	International Standard Book Number. Używany wyłącznie w branży księgarskiej. Jego obecność może aktywować specyficzne szablony na karcie produktu.	[1]
bloz7 / bloz12	Integer	Baza Leków i	[1]

		Środków Ochrony Zdrowia. Krytyczne pola dla aptek internetowych w Polsce. Są to identyfikatory wymagane przez prawo farmaceutyczne do sprzedaży leków i suplementów.	
gtu	String	Grupa Towarowo-Usługowa. Pole wprowadzone w odpowiedzi na wymogi polskiego pliku JPK_V7 (Jednolity Plik Kontrolny). Przyjmuje wartości od GTU_01 do GTU_13. Jest to pole *podatkowe*, a nie *logistyczne*, ale jego obecność w API jest niezbędna dla legalności obrotu.	[1]
kgo	String	Koszt Gospodarowania Odpadami. Ważne dla branży AGD/RTV.	[1]
warehouse	String	Kod lokalizacji magazynowej (np. "Regał 4, Półka B"). To pole jest często	[6]

		mylone z stock.code. Jeśli integrator chce przekazać informację o *miejscu* składowania, należy użyć pola additional_codes.warehouse.	

---

4. Strategia Implementacji dla Produktów Prostych

Dla produktów niewymagających konfiguracji (tzw. Simple Products), proces tworzenia jest atomowy. Wykonujemy jedno żądanie HTTP POST, które ustanawia kompletną tożsamość produktu.

4.1 Konstrukcja Payloadu JSON

Poniżej przedstawiono wzorcowy przykład żądania dla produktu prostego, z pełnym ometkowaniem kodami.

JSON

```
{
  "category_id": 101,
  "code": "ELE-LAP-001",
  "pkwiu": "26.20.11.0",
  "stock": {
    "price": 3500.00,
    "active": true,
    "stock": 10,
```

```
    "weight": 2.5,
    "code": "MAG-ELE-001-A",
    "ean": "5901234567890",
    "delivery_id": 1,
    "additional_codes": {
      "gtu": "GTU_06",
      "warehouse": "SEC-A-ROW-1",
      "producer": "DELL-XPS-13"
    }
  },
  "translations": {
    "pl_PL": {
      "name": "Laptop Dell XPS 13",
      "active": true,
      "description": "<p>Opis produktu...</p>"
    }
  }
}
```

Komentarz Implementacyjny:
W powyższym przykładzie widzimy wyraźne rozdzielenie ról:

- code ("ELE-LAP-001") to indeks handlowy używany w sklepie.
- stock.code ("MAG-ELE-001-A") to indeks magazynowy używany do kompletacji.
- additional_codes.producer ("DELL-XPS-13") to kod producenta (MPN).
- additional_codes.gtu ("GTU_06") to klasyfikacja podatkowa dla elektroniki.

Taka struktura zapewnia pełną zgodność z wymogami logistycznymi i prawnymi już w momencie utworzenia rekordu.

---

5. Zaawansowana Strategia Konfiguracji: Obsługa Wariantów (Stock Rozszerzony)

Największym wyzwaniem w integracji z API Shoper jest obsługa produktów wariantowych (Configurable Products). Wymaga ona sekwencyjnego podejścia i zrozumienia relacji między zasobami. Nie jest możliwe zdefiniowanie wariantów i ich kodów w jednym strzale API (single shot).

5.1 Algorytm Tworzenia Produktu Wariantowego

Proces ten musi przebiegać w ściśle określonych krokach:

1. Utworzenie Produktu Matki (Parent Product):
   Wysyłamy żądanie POST do products. Kluczowym elementem jest ustawienie pola stock.extended na 1 (lub true).
   JSON
   ```
   {
    "code": "TSHIRT-SUMMER-2024",
    "stock": {
      "extended": 1,
      "price": 50.00,
      "active": true
    },
   ...
   }
   ```
   Uwaga: Cena i stan magazynowy tutaj są jedynie wartościami bazowymi, często wirtualnymi.
2. Pobranie ID Produktu:
   Odpowiedź z API zwróci identyfikator nowego produktu (np. product_id: 500).
3. Identyfikacja lub Utworzenie Opcji:
   Należy znać ID grupy wariantów oraz ID poszczególnych wartości (np. Rozmiar: S, M, L). Wymaga to wcześniejszej interakcji z zasobem attributes lub option_groups.
4. Generowanie Wariantów Magazynowych (product-stocks):
   Jest to moment, w którym nadajemy unikalne kody magazynowe dla każdego wariantu. Dla każdej kombinacji (np. Koszulka-Czerwona-S) musimy wysłać żądanie POST do product-stocks.5

5.2 Payload dla Zasobu product-stocks

Dokumentacja [5] precyzuje strukturę dla tworzenia wariantu. To tutaj decydujemy o tożsamości konkretnej sztuki towaru.

JSON

```
{
 "product_id": 500,
 "code": "TSHIRT-SUM-24-RED-S",
 "price": 50.00,
 "stock": 100,
 "active": true,
 "options": [
  {
   "option_id": 10,
   "value_id": 25
  },
  {
   "option_id": 11,
   "value_id": 30
  }
 ]
}
```

Kluczowe Aspekty:

- **Pole code**: Tutaj wprowadzamy "Kod Magazynowy Wariantu". Jest on unikalny dla tej konkretnej kombinacji opcji.
- **Tablica options**: Definiuje, czym fizycznie jest ten wariant. Mapuje ID opcji (np. Kolor) na ID wartości (np. Czerwony).

5.3 Zarządzanie Cenami i Wagami w Kontekście Wariantów

API Shoper oferuje zaawansowane mechanizmy dziedziczenia i modyfikacji cen oraz wag dla wariantów, sterowane polami price_type i weight_type.[5] Zrozumienie tych mechanizmów jest kluczowe dla poprawnego zmapowania logiki biznesowej ERP.

Tabela przedstawiająca typy kalkulacji cenowej (price_type):

Wartość price_type	Typ Kalkulacji	Opis Mechanizmu	Zastosowanie Biznesowe
0	Cena z produktu	Wariant dziedziczy cenę z produktu	Proste warianty (np. kolor), które

		głównego. Pole price w wariancie jest ignorowane.	nie wpływają na koszt produkcji.
1	Nowa cena	Wariant posiada własną, sztywną cenę, niezależną od produktu głównego.	Warianty premium lub wyprzedaże konkretnych rozmiarów.
2	Dodaj do ceny bazowej	Cena wariantu to cena_bazowa + price.	Opcje dodatkowe (np. +5 zł za opakowanie ozdobne).
3	Odejmij od ceny bazowej	Cena wariantu to cena_bazowa - price.	Warianty ekonomiczne lub uszkodzone.

Podobna logika stosowana jest dla weight_type (waga), co ma bezpośrednie przełożenie na obliczanie kosztów wysyłki w koszyku. Integrator musi zdecydować, czy kod wariantu w systemie źródłowym niesie ze sobą informację o innej wadze (np. materac 90cm vs 160cm) i odpowiednio ustawić flagę weight_type.

---

6. Integracja Wielomagazynowa (Multi-Warehouse)

Jednym z najbardziej zaawansowanych aspektów API Shoper jest obsługa wielu magazynów. Gdy w sklepie aktywny jest moduł "Magazyny", pole stock (ilość) w obiekcie products oraz product-stocks staje się polem tylko do odczytu (read-only).[3]

6.1 Zmiana Paradygmatu: Obiekt warehouses

W trybie wielomagazynowym, zamiast prostego pola stock: float, należy operować na mapie

warehouses.

JSON

```
"stock": {
 ...
  "warehouses": {
    "1": { "stock": 10.0 },
    "2": { "stock": 25.0 }
  }
 ...
}
```

Gdzie klucze "1" i "2" to identyfikatory zdefiniowanych w systemie magazynów.

Implikacje dla Kodów:
Mimo że stany magazynowe są rozbite na różne lokalizacje fizyczne, Shoper w swojej architekturze zachowuje jeden kod magazynowy (stock.code) dla danego wariantu, niezależnie od tego, w ilu magazynach się on znajduje. Jest to istotne ograniczenie architektoniczne. Jeśli system ERP używa różnych kodów SKU dla tego samego produktu w zależności od magazynu (np. WAW-PROD-01 i KRK-PROD-01), integracja musi zmapować je na jeden wspólny identyfikator w Shoperze, a rozróżnienie realizować wyłącznie na poziomie ilości.

---

7. Obsługa Błędów i Wzorce Projektowe

Solidna integracja musi przewidywać sytuacje awaryjne. API Shoper zwraca kody błędów HTTP, które należy odpowiednio obsłużyć.

7.1 Typowe Błędy przy Nadawaniu Kodów

1. **409 Conflict (Duplicate Entry)**: Próba utworzenia produktu z kodem, który już istnieje.
    - *Strategia:* Przed wysłaniem POST, wykonaj GET z filtrem ?filters={"code":"..."} aby sprawdzić istnienie.

2. **400 Bad Request (Validation Error)**: Błędny typ danych (np. string zamiast float w cenie) lub przekroczenie limitu znaków.
    - *Strategia:* Walidacja danych po stronie klienta (integratora) przed wysłaniem żądania, zgodnie z typami opisanymi w sekcji 3 i 5.

7.2 Wykorzystanie Webhooków do Weryfikacji

Po wysłaniu żądania API, warto nasłuchiwać na webhook product.create.[4] Payload webhooka zawiera pełną strukturę utworzonego produktu. Porównanie danych wysłanych z danymi otrzymanymi w webhooku (w szczególności pola stock.code) jest najlepszą metodą weryfikacji spójności danych (Data Integrity Check).

7.3 Przykład Obsługi Wyjątków w PHP

Poniższy fragment kodu demonstruje użycie bloku try-catch z biblioteką DreamCommerce do bezpiecznego tworzenia produktu.[10]

PHP

```
try {
    // Inicjalizacja klienta
    $client = DreamCommerce\ShopAppstoreLib\Client::factory('OAuth', $options);
    $resource = new DreamCommerce\ShopAppstoreLib\Resource\Product($client);

    // Przygotowanie danych
    $data =,
        //... inne pola
    ];

    // Próba utworzenia
    $productId = $resource->post($data);

    // Logowanie sukcesu
```

```
    Logger::info("Utworzono produkt ID: $productId z kodem UNIQ-CODE-123");

} catch (DreamCommerce\ShopAppstoreLib\Exception\ValidationException $ex) {
    // Specyficzna obsługa błędów walidacji (np. zduplikowany kod)
    $errors = $ex->getContext();
    Logger::error("Błąd walidacji pola 'code': ". print_r($errors, true));
} catch (DreamCommerce\ShopAppstoreLib\Exception\Exception $ex) {
    // Ogólny błąd API
    Logger::critical("Krytyczny błąd API: ". $ex->getMessage());
}
```

---

8. Studia Przypadku (Case Studies)

8.1 Scenariusz A: Apteka Internetowa

Integrator systemów medycznych (KS-Apteka) łączy się z Shoperem.

- **Wyzwanie**: Konieczność przesyłania kodów BLOZ oraz monitorowania dat ważności (choć API dat ważności natywnie nie wspiera wprost, używa się atrybutów).
- **Rozwiązanie**: Pole code otrzymuje EAN leku. Pole stock.code otrzymuje wewnętrzny ID apteczny. Pole stock.additional_codes.bloz7 jest wypełniane bezwzględnie. Użycie wariantów jest rzadkie, dominuje model extended = 0.

8.2 Scenariusz B: Marka Odzieżowa (Fashion)

Producent odzieży z systemem ERP Comarch Optima.

- **Wyzwanie**: Jeden model kurtki występuje w 5 kolorach i 6 rozmiarach (30 wariantów). Ceny są identyczne, ale stany magazynowe drastycznie różne.
- **Rozwiązanie**:
  1. products: insert tworzy "Kurtkę Zimową" z extended = 1. Kod produktu: KURTKA-ZIM-2024.
  2. Skrypt pętli iteruje po macierzy wariantów z ERP.

3. 30 zapytań do product-stocks: insert. Każdy wariant otrzymuje code w formacie KURTKA-ZIM-2024-- (np. KURTKA-ZIM-2024-RED-XL).
4. Wykorzystanie price_type = 0 (cena z produktu), aby zmiana ceny głównej automatycznie propagowała się na wszystkie 30 wariantów.

---

9. Wnioski Końcowe

Poprawne nadanie kodu produktu i kodu magazynowego w API Shoper nie jest trywialną operacją przypisania wartości do pola. Jest to proces decyzyjny wymagający zrozumienia architektury systemu.

Kluczowe wnioski dla deweloperów:

1. **Rozróżniaj kontekst**: Zawsze wiedz, czy tworzysz produkt prosty, czy konfigurowalny.
2. **Pilnuj zagnieżdżenia**: stock.code zawsze musi znajdować się wewnątrz obiektu stock (przy tworzeniu produktu) lub w korzeniu (przy tworzeniu wariantu w product-stocks).
3. **Planuj warianty**: Warianty wymagają osobnych żądań API. Nie da się ich "wstrzyknąć" w głównym payloadzie produktu.
4. **Wykorzystaj kody dodatkowe**: Nie zaśmiecaj pola code danymi o lokalizacji magazynowej czy podatkach – użyj do tego dedykowanych pól w additional_codes.

Przestrzeganie przedstawionych w tym raporcie wzorców i struktur danych gwarantuje stabilność integracji, poprawność procesów logistycznych oraz zgodność z wymogami prawnymi, stanowiąc solidny fundament dla skalowalnego biznesu e-commerce.

Cytowane prace

1. Products: insert - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/products/insert
2. Product Stock - integer - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/product-stocks
3. Products: list - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/products/list
4. REST API - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/getting-started
5. Product Stock: insert - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/product-stocks/insert
6. Product Stock: list - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/product-stocks/list
7. Products - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/products
8. Product Files: list - Shoper Developers, otwierano: listopada 19, 2025,

https://developers.shoper.pl/developers/api/resources/product-files/list
9. Product Stock: update - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/product-stocks/update
10. Order Products: insert - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/api/resources/order-products/insert
11. product.create - Shoper Developers, otwierano: listopada 19, 2025, https://developers.shoper.pl/developers/webhooks/methods/product-create
12. Debugging — ShopAppstoreLib - PHP 0.2 documentation, otwierano: listopada 19, 2025, https://shopappstorephplib.readthedocs.io/en/latest/library/debugging.html