Analiza Techniczna API Shoper: Prawidłowa Architektura Procesu Tworzenia Produktu, Zarządzanie Atrybutami i Obrazami

I. Wprowadzenie: Architektura Procesu Tworzenia Produktu w API Shoper

Analiza dokumentacji API Shoper, przeprowadzona w odpowiedzi na zapytanie dotyczące prawidłowej metodyki tworzenia produktów, ujawnia, że proces ten opiera się na architekturze rozproszonej, znanej jako "Model Rozdzielonych Zasobów" (Separated Resources Model). Zrozumienie tego modelu jest fundamentalne dla prawidłowej implementacji i uniknięcia typowych błędów.

Proces utworzenia kompletnego produktu (zawierającego dane podstawowe, atrybuty i obrazy) nie jest pojedynczą, atomową operacją wysłaną do jednego punktu końcowego. Zamiast tego, jest to *sekwencja* (workflow) logicznych, oddzielnych wywołań API do różnych, wyspecjalizowanych zasobów.

Kluczowe zasoby (endpoints) zaangażowane w ten proces to:

1. **Products**: Stanowi rdzeń (szkielet) produktu. Ten zasób przechowuje podstawowe dane handlowe (nazwa, cena, kod, stan magazynowy) oraz, co kluczowe, *powiązania* z innymi zasobami, takimi jak atrybuty.[1]
2. **Attributes**: Działa jako globalny "słownik" lub "schemat" dla definicji atrybutów. Zarządza się tu takimi pojęciami jak "Kolor" czy "Rozmiar", ale *nie* ich wartościami dla konkretnego produktu.[1]
3. **Product Images**: Reprezentuje kolekcję obiektów graficznych (obrazów) powiązanych bezpośrednio z danym produktem.[1]

Ta architektoniczna separacja bezpośrednio odpowiada na kluczowe pytania postawione w

zapytaniu:

- **W odniesieniu do Atrybutów (POST vs. PUT):** Wartości atrybutów (np. "Czerwony") nie są zarządzane poprzez wywołania POST lub PUT do zasobu /attributes. Zamiast tego, są one przypisywane *wewnątrz* pola attributes w ciele żądania wysyłanego do zasobu Products (tj. podczas POST /products lub PUT /products/<id>).
- **W odniesieniu do Obrazów (URL vs. "Inaczej"):** Obrazy są dodawane jako oddzielny, wtórny krok *po* utworzeniu produktu. Wykonuje się to poprzez żądanie POST do zasobu /product-images. Punkt końcowy jest elastyczny i akceptuje *zarówno* zdalny adres url, jak i dane obrazu zakodowane w content (Base64).[5]

Niniejszy raport szczegółowo dekonstruuje wymaganą sekwencję wywołań, analizuje struktury ciał żądań dla każdego kroku oraz wyjaśnia fundamentalne różnice między zasobami, aby zapewnić solidne podstawy dla udanej integracji.

II. Faza 1: Tworzenie Szkieletu Produktu (Endpoint Products: insert)

Pierwszym krokiem w procesie jest utworzenie podstawowego rekordu produktu, który będzie służył jako fundament (lub "kontener") dla atrybutów, obrazów i wariantów. Operację tę wykonuje się na głównym zasobie Products.

- **Zasób:** Products [1]
- **Metoda HTTP:** POST (logicznie wynikająca z akcji insert opisanej w dokumentacji [1])
- **Punkt Końcowy:** POST https://shop.url/webapi/rest/products

Dekonstrukcja Ciała Żądania (Request Body)

Chociaż dokumentacja dla products/insert [2] wymienia dziesiątki pól, ale nie oznacza ich jawnie jako "obowiązkowe", logika biznesowa platformy e-commerce dyktuje, że pewien minimalny zestaw danych jest niezbędny do pomyślnego utworzenia funkcjonalnego produktu. Próba utworzenia produktu bez tych danych prawdopodobnie zakończy się błędem invalid_request [7] lub utworzeniem "wadliwego" rekordu, który będzie bezużyteczny z punktu widzenia sklepu.

Analiza przykładu implementacji PHP w dokumentacji [6] potwierdza, że następujące pola stanowią *praktycznie obowiązkowy* rdzeń żądania POST /products:

1. **translations (Tablica asocjacyjna):** Kluczowe dla określenia nazwy produktu w co najmniej jednym języku. Bez nazwy produkt nie może być wyświetlony.
    - *Struktura:* translations[pl_PL][name], translations[pl_PL][description].[6]
2. **stock (Tablica asocjacyjna):** Niezbędne do zdefiniowania podstawowej ceny, stanu magazynowego i statusu aktywności produktu.
    - *Struktura:* stock[price], stock[stock], stock[active].[6]
3. **code (String):** Unikalny identyfikator produktu (SKU). Krytyczny dla zarządzania magazynem i integracji.
    - *Struktura:* "code": "TWÓJ-SKU-123".[6]
4. **tax_id (Integer):** Identyfikator definicji podatku (VAT), do której przypisany jest produkt.
    - *Struktura:* "tax_id": 1.[6]
5. **category_id (Integer):** Identyfikator głównej kategorii, w której produkt będzie widoczny.[6]
6. **unit_id (Integer):** Identyfikator jednostki miary (np. "szt.", "kg").[2]

Pomyślne wykonanie tego żądania zwróci odpowiedź JSON zawierającą id nowo utworzonego produktu. Ten identyfikator jest absolutnie kluczowy dla wszystkich kolejnych operacji, takich jak dodawanie obrazów.

III. Analiza Głębinowa: Prawidłowe Zarządzanie Atrybutami Produktu

Ta sekcja stanowi sedno odpowiedzi na zapytanie użytkownika dotyczące atrybutów. Zamieszanie wokół POST vs. PUT wynika z fundamentalnego niezrozumienia różnicy między *definiowaniem* atrybutu (jako schematu) a *przypisywaniem* mu wartości (jako danych produktu).

Rozdział 1: Błędne Założenie – Czym *Nie* Jest Przypisywanie Atrybutów (Zasób Attributes)

Zasób Attributes [3] służy *wyłącznie* do zarządzania globalnym słownikiem (definicjami) atrybutów dostępnych w całym sklepie.

Analiza Attributes: insert (POST /attributes)

- **Punkt Końcowy:** POST https://shop.url/webapi/rest/attributes.[8]
- **Cel:** "Creates a new object" [3] – tworzy nową, globalną *definicję* atrybutu.
- **Ciało Żądania:** Wymaga pól definiujących *schemat* atrybutu, takich jak name (np. "Kolor"), attribute_group_id (do której grupy należy) oraz type (np. 0 - text field, 2 - drop down).[8]
- **Wniosek:** To wywołanie *tylko* tworzy definicję "Kolor". *Nie* przypisuje ono koloru "Czerwony" do żadnego produktu.[8]

Analiza Attributes: update (PUT /attributes/<id>)

- **Punkt Końcowy:** PUT https://shop.url/webapi/rest/attributes/<id>.[9]
- **Cel:** "Modifies an object" [3] – modyfikuje globalną *definicję* atrybutu.
- **Ciało Żądania:** Akceptuje name, description, options (dla listy rozwijanej) itp..[9]
- **Wniosek:** Służy do zmiany globalnej definicji, np. "Zmień nazwę atrybutu o ID 5 na 'Kolor Podstawowy'".

Używanie zasobu /attributes do przypisywania wartości atrybutu do konkretnego produktu jest błędem architektonicznym. Te punkty końcowe zarządzają *metadanymi* (schematem), a nie *danymi* produktu.

Rozdział 2: Prawidłowa Metoda – Przypisywanie Wartości Atrybutów (Zasób Products)

Prawidłowy mechanizm przypisywania wartości atrybutów (np. "Kolor: Czerwony") do produktu znajduje się *wewnątrz* zasobu Products.

Dokumentacja dla products/insert [6] oraz products/update [10] jawnie wymienia pole o nazwie attributes jako część ciała żądania wysyłanego do tych punktów końcowych.

To wyjaśnia dylemat "POST vs. PUT":

1. Użyj **POST /products** z polem attributes, aby *stworzyć nowy produkt* i *jednocześnie* przypisać mu wartości atrybutów.
2. Użyj **PUT /products/<id>** z polem attributes, aby *zaktualizować* lub *dodać* wartości

atrybutów do *istniejącego* produktu.

Analiza Struktury Pola attributes (Rozwiązanie Niespójności)

Podczas analizy dokumentacji zidentyfikowano krytyczną niespójność dotyczącą struktury danych pola attributes:

- Źródło 1 [6]: Opisuje typ pola attributes jako string, ale jednocześnie opis słowny mówi "an associative array (keys: attribute identifiers, values: attribute value)". Jest to wewnętrznie sprzeczne.
- Źródło 2 [2]: Opisuje pole jako array i dostarcza znacznie bardziej precyzyjny opis: "a nested associative array - keys of main array = **attribute group identifiers**, values: an associative array (keys: **attribute identifiers**, values: **attribute value**)".
- Źródło 3 [12]: Dokumentacja webhooka product-create, opisując *wynikową* strukturę danych, ujawnia logikę systemu: attributes[n].id (group identifier), attributes[n].attributes[o].id (attribute identifier), attributes[n].attributes[o].value.

Synteza tych trzech źródeł pozwala na sformułowanie ostatecznego, prawidłowego formatu. Należy zignorować mylący typ string z [6] i.[10] Prawidłowa struktura opiera się na zagnieżdżonej tablicy asocjacyjnej, gdzie kluczami są identyfikatory liczbowe, co jest zgodne z [2] i.[12]

Prawidłowa struktura JSON dla pola attributes w ciele żądania POST /products lub PUT /products/<id> jest następująca:

JSON

```
{
  "attributes": {
    "GROUP_ID_1": {
      "ATTRIBUTE_ID_1": "Wartość tekstowa atrybutu 1",
      "ATTRIBUTE_ID_2": "Wartość tekstowa atrybutu 2"
    },
    "GROUP_ID_2": {
      "ATTRIBUTE_ID_3": "Wartość dla atrybutu 3 w innej grupie"
    }
  }
}
```

Gdzie GROUP_ID_* to numeryczne ID grupy atrybutów, a ATTRIBUTE_ID_* to numeryczne ID definicji atrybutu (uzyskane wcześniej z zasobu /attributes).

Tabela 1: Rozróżnienie Operacji na Atrybutach

Poniższa tabela wizualizuje różnice w zarządzaniu atrybutami, stanowiąc bezpośrednią odpowiedź na zapytanie.

Cel Operacji	Metoda HTTP	Punkt Końcowy (Endpoint)	Przykład Ciała Żądania (Fragment)	Źródło
Definiowanie: Stworzenie nowej definicji atrybutu (np. "Materiał")	POST	/webapi/rest/attributes	{ "name": "Materiał", "attribute_group_id": 1, "type": 0 }	[8]
Definiowanie: Modyfikacja globalnej definicji atrybutu	PUT	/webapi/rest/attributes/<id>	{ "name": "Materiał (Premium)" }	[9]
Przypisywanie: Przypisanie wartości atrybutów *podczas tworzenia* produktu	POST	/webapi/rest/products	{ "code": "SKU123",..., "attributes": { "1": { "5": "Bawełna" } } }	[2]
Przypisywanie: Aktualizacja wartości atrybutów	PUT	/webapi/rest/products/<id>	{ "attributes": { "1": { "5": "Poliester" } } }	[2]

istniejącego produktu				

IV. Analiza Głębinowa: Prawidłowe Dodawanie Obrazów Produktu

Druga część zapytania dotyczyła sposobu dodawania obrazów: poprzez URL czy "inaczej" (np. Base64). Analiza wykazuje, że podobnie jak atrybuty, obrazy są zarządzane przez oddzielny zasób i wymagają określonej sekwencji operacji.

Rozdział 1: Identyfikacja Prawidłowego Zasobu i Metody

Analiza ciała żądania Products: insert [6] wykazuje całkowity *brak* jakichkolwiek pól służących do przesyłania danych obrazu (w przeciwieństwie do pola attributes, które tam występuje).

To prowadzi do kluczowego wniosku: dodawanie obrazów *musi* być operacją wtórną, wykonaną *po* pomyślnym utworzeniu produktu (w Fazie 1) i uzyskaniu jego unikalnego product_id.

Prawidłowym zasobem do tej operacji jest Product Images:

- **Zasób:** Product Images [1]
- **Metoda HTTP:** POST (wynikająca z akcji insert [1])
- **Punkt Końcowy:** POST https://shop.url/webapi/rest/product-images [5]

Ciało żądania dla tego punktu końcowego zawiera pole product_id [5], co ostatecznie potwierdza, że jest to operacja wtórna, służąca do powiązania obrazu z istniejącym rekordem produktu.

Rozdział 2: Metody Dostarczania Danych Obrazu (Odpowiedź na "URL czy inaczej")

Analiza dokumentacji dla Product Images: insert [5] dostarcza bezpośredniej odpowiedzi na pytanie użytkownika. API Shoper jest elastyczne i akceptuje **obie** popularne metody dostarczania danych obrazu w jednym żądaniu POST:

1. **Metoda URL (Zdalne Pobranie):**
   - **Pole:** url (Typ: string, Obowiązkowe: no)
   - **Opis:** "if present, file contents is being downloaded from specified URL".[5]
   - **Implementacja:** Wystarczy podać publicznie dostępny adres URL obrazu. Serwer Shoper sam pobierze plik.
2. **Metoda Base64 (Przesłanie Treści):**
   - **Pole:** content (Typ: string, Obowiązkowe: no)
   - **Opis:** "base64-encoded file contents".[5]
   - **Implementacja:** Należy lokalnie zakodować plik obrazu do formatu Base64 (wraz z prefiksem) i przesłać cały ciąg znaków w tym polu.

Implementator powinien wybrać jedną z tych metod – nie należy używać obu jednocześnie.

Pułapka Implementacyjna: Product Images vs. Product Files

Należy zachować szczególną ostrożność, aby nie pomylić zasobu Product Images (służącego do galerii produktu) z bardzo podobnym zasobem Product Files (służącym do załączników, np. instrukcji PDF, plików.zip).

- Product Images: insert [5]: Używa product_id do powiązania obrazu z produktem.
- Product Files: insert [11]: Również używa pola content (Base64), ale do powiązania pliku wymaga translation_id (identyfikatora tłumaczenia produktu), a nie product_id.

Jest to krytyczna różnica w implementacji; użycie niewłaściwego punktu końcowego spowoduje niepowodzenie operacji powiązania pliku z produktem.

Tabela 2: Dekonstrukcja Ciała Żądania POST /product-images

Poniższa tabela przedstawia kluczowe pola dla żądania dodania obrazu.

Nazwa Pola	Typ Danych	Obowiązkowe	Opis Działania	Źródło

product_id	integer	Nie*	Identyfikator produktu (uzyskany z Fazy 1), z którym obraz ma być powiązany. *Praktycznie obowiązkowy dla powiązania*.	[5]
url	string	Nie	API pobierze obraz ze wskazanego URL. Należy użyć *albo* tego pola, *albo* content.	[5]
content	string	Nie	Zawartość pliku zakodowana w Base64. Należy użyć *albo* tego pola, *albo* url.	[5]
hidden	boolean	Nie	Ustawienie, czy obraz ma być domyślnie ukryty.	[5]
translations	array	Nie	Tablica asocjacyjna dla opisów SEO (alt) i tytułów obrazu. Np. translations[pl	[5]

			_PL][name].	

*Dokumentacja [5] oznacza wszystkie pola jako "Mandatory: no", co jest prawdopodobnie uproszczeniem. Logika systemu wymaga, aby obraz był powiązany z produktem (product_id) i aby miał źródło (albo url, albo content).

V. Zsyntetyzowany, Kompletny Przepływ Pracy (Workflow)

Poniżej przedstawiono kompletny, wieloetapowy proces tworzenia produktu wraz z atrybutami i obrazami, syntetyzujący wszystkie powyższe analizy.

Krok 0: Przygotowanie – Definiowanie Atrybutów (Operacja Jednorazowa)

Ten krok jest wykonywany tylko wtedy, gdy definicje atrybutów (np. "Kolor", "Materiał") jeszcze nie istnieją w sklepie.

- **Akcja:** POST /webapi/rest/attributes
- **Cel:** Stworzenie globalnej definicji atrybutu "Materiał" w grupie o ID 1, jako pole tekstowe.
- **Ciało Żądania:**
  JSON
  ```
  {
   "name": "Materiał",
   "attribute_group_id": 1,
   "type": 0,
   "active": true
  }
  ```

- **Rezultat:** Zanotuj zwrócone id atrybutu (np. 15) oraz id grupy (tutaj 1).

Krok 1: Tworzenie Produktu i Przypisywanie Atrybutów

Jest to główna operacja tworzenia rdzenia produktu, która jednocześnie przypisuje wartości do *wcześniej zdefiniowanych* atrybutów (z Kroku 0).

- **Akcja:** POST /webapi/rest/products
- **Cel:** Stworzenie nowego produktu z ceną, stanem, nazwą oraz przypisaną wartością "Bawełna" do atrybutu "Materiał" (ID: 15) w grupie (ID: 1).
- **Kompletne Przykładowe Ciało Żądania:**
  JSON

```
{
  "producer_id": 2,
  "category_id": 10,
  "unit_id": 1,
  "tax_id": 1,
  "code": "SKU-PREMIUM-TSHIRT",
  "translations": {
    "pl_PL": {
      "name": "Wysokiej jakości T-shirt",
      "short_description": "Krótki opis produktu.",
      "description": "Długi, szczegółowy opis zalet tego produktu."
    }
  },
  "stock": {
    "price": 149.99,
    "stock": 50,
    "active": true
  },
  "attributes": {
    "1": {
      "15": "Bawełna"
    }
  }
}
```

Krok 2: Przechwycenie product_id

- **Akcja:** Przetworzenie odpowiedzi JSON z Kroku 1.
- **Odpowiedź Serwera (Przykład):**
  JSON

```
{
  "id": 451,
  "status": true
}
```

- **Rezultat:** Należy wyodrębnić i zapisać wartość id (w tym przypadku 451). Ten identyfikator jest niezbędny do Kroku 3.

Krok 3: Dodawanie Obrazów do Utworzonego Produktu (Operacja Wtórna)

Używając product_id (np. 451) z Kroku 2, można teraz dodać obrazy do galerii produktu. Ten krok można powtarzać wielokrotnie dla każdego obrazu.

- **Akcja:** POST /webapi/rest/product-images

Przykład 3a (Metoda URL)

- **Cel:** Dodanie obrazu ze zdalnego serwera.
- **Ciało Żądania:**

```
JSON
{
  "product_id": 451,
  "url": "https://cdn.example.com/images/tshirt-bawelna-front.jpg",
  "translations": {
    "pl_PL": {
      "name": "T-shirt bawełniany - widok z przodu"
    }
  }
}
```

Przykład 3b (Metoda Base64 / content)

- **Cel:** Dodanie obrazu przesłanego bezpośrednio z lokalnego systemu.

- **Ciało Żądania:**
```
JSON
{
 "product_id": 451,
 "content": ".....//Z",
 "translations": {
  "pl_PL": {
   "name": "T-shirt bawełniany - zbliżenie na materiał"
  }
 }
}
```

Po wykonaniu tych trzech kroków, produkt jest w pełni utworzony w systemie Shoper, posiada dane podstawowe, przypisane wartości atrybutów oraz kompletną galerię obrazów.

VI. Podsumowanie i Kluczowe Wnioski Implementacyjne

Analiza dokumentacji API Shoper pozwoliła na sformułowanie jednoznacznej i prawidłowej metodyki tworzenia produktów, ze szczególnym uwzględnieniem atrybutów i obrazów.

1. **Architektura Sekwencyjna:** Tworzenie produktu w API Shoper jest procesem *sekwencyjnym*, opartym na *rozdzielonych zasobach*. Wymaga co najmniej dwóch różnych wywołań (do /products i /product-images), a nie pojedynczej, atomowej operacji.
2. **Rozwiązanie dylematu Atrybutów (POST vs. PUT):** Atrybuty są przypisywane do produktu poprzez pole attributes w ciele żądania POST /products (podczas tworzenia) lub PUT /products/<id> (podczas aktualizacji). Zasoby POST /attributes i PUT /attributes/<id> służą *wyłącznie* do zarządzania globalnymi definicjami (schematami) atrybutów, a nie ich wartościami na produkcie.
3. **Rozwiązanie dylematu Obrazów (URL vs. "Inaczej"):** Obrazy dodaje się poprzez *oddzielne* wywołanie POST /product-images *po* utworzeniu produktu, używając product_id do powiązania. API jest elastyczne i akceptuje *obie* metody: pole url [5] dla zdalnych zasobów oraz pole content [5] dla danych zakodowanych w Base64.
4. **Krytyczne Ostrzeżenia Implementacyjne:**
   - **Struktura Atrybutów:** Należy zignorować mylące definicje typu string dla pola attributes [6] i zastosować zagnieżdżoną strukturę tablicy asocjacyjnej opartą na GROUP_ID i ATTRIBUTE_ID, zgodnie z syntezą źródeł [2] i.[12]
   - **Rozróżnienie Zasobów:** Należy zwrócić szczególną uwagę na różnicę między

Product Images (galeria produktu, wymaga product_id [5]) a Product Files (załączniki, wymaga translation_id [11]), ponieważ oba akceptują dane Base64, co może prowadzić do pomyłek.

Cytowane prace

1. Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources
2. Products - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/products
3. Attributes - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/attributes
4. Product Images - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/product-images
5. Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/product-images/insert
6. Products: insert - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/products/insert
7. REST API - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/getting-started
8. Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/attributes/insert
9. Attributes: update - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/attributes/update
10. Products: update - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/products/update
11. Product Files: insert - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/api/resources/product-files/insert
12. product.create - Shoper Developers, otwierano: listopada 13, 2025, https://developers.shoper.pl/developers/webhooks/methods/product-create