# **Architektura handlu konwersacyjnego: kompleksowy przewodnik po tworzeniu chatbotów AI Shopify z protokołem Model Context**

## **Część I: Fundacja MCP dla handlu konwersacyjnego**

Ta fundamentalna część ustanawia ramy koncepcyjne dla protokołu Model Context Protocol (MCP), pozycjonując go jako fundamentalną zmianę architektoniczną w sztucznej inteligencji w e-commerce. Zapewnia niezbędny kontekst do zrozumienia decyzji technicznych i strategicznych szczegółowo opisanych w kolejnych sekcjach.

### **1.1 Zmiana paradygmatu od interfejsów API niestandardowych do protokołu uniwersalnego**

Ewolucję sztucznej inteligencji w e-commerce historycznie ograniczała istotna bariera techniczna: wysokie tarcie integracji danych. Tradycyjnie, połączenie modelu AI, takiego jak chatbot konwersacyjny, z dynamicznymi danymi sklepu Shopify wymagało opracowania dedykowanych, niestandardowych integracji API.1Proces ten był zasobochłonny, kruchy i wymagał powtarzania dla każdego nowego modelu sztucznej inteligencji lub źródła danych. Aplikacja zbudowana do współpracy z modelem OpenAI wymagałaby gruntownej przebudowy, aby działać z modelem Anthropic lub Google. To bezpośrednie odwzorowanie sztucznej inteligencji na API stworzyło krajobraz rozproszonych, nieinteroperacyjnych rozwiązań, hamując innowacyjność i ograniczając adopcję zaawansowanych możliwości sztucznej inteligencji w przedsiębiorstwach dysponujących znacznymi zasobami programistycznymi.1

Protokół Model Context Protocol (MCP) stanowi odejście od tego modelu. Ustandaryzowany interfejs – uniwersalny język – umożliwia płynną komunikację w czasie rzeczywistym między modelami sztucznej inteligencji a zewnętrznymi źródłami danych, w szczególności sklepami Shopify.1Zamiast tworzyć unikalnego tłumacza dla każdej rozmowy, MCP zapewnia wspólną gramatykę i słownictwo, zrozumiałe dla każdego zgodnego systemu AI. Ta standaryzacja eliminuje podstawową złożoność uwierzytelniania, interpretacji schematów i formatowania danych, radykalnie skracając nakład pracy związany z rozwojem typowej integracji z tygodni do kilku godzin.1

Dla partnerów i deweloperów Shopify ta zmiana jest głęboka. Demokratyzuje ona dostęp do zaawansowanych funkcji sztucznej inteligencji (AI), umożliwiając szybki rozwój i wdrożenie inteligentnych agentów zdolnych do wykonywania złożonych operacji handlowych. Agenci ci mogą przeszukiwać katalogi produktów za pomocą języka naturalnego, zarządzać koszykami, odpowiadać na pytania dotyczące zasad i prowadzić użytkowników przez proces realizacji zamówienia – a wszystko to bez konieczności tworzenia dedykowanego zaplecza dla każdego sklepu.2To podejście oparte na protokole eliminuje złożoność integracji, tworząc podwaliny nowego ekosystemu, w którym asystenci AI staną się podstawowym, konwersacyjnym interfejsem dla handlu.2

### **1.2 Analiza architektury MCP: klient, host i serwer**

Siła i bezpieczeństwo protokołu Model Context Protocol wynikają z jego starannie zaprojektowanej, trzyczęściowej architektury: klienta, hosta i serwera. Model ten, oparty na specyfikacji JSON-RPC 2.0, zapewnia wyraźny podział zadań, co umożliwia solidną, skalowalną i bezpieczną integrację AI.4

* **Serwer:** U podstaw Serwera leży zestaw konkretnych, ściśle zdefiniowanych możliwości. W kontekście niniejszego raportu, Shopify udostępnia serwery MCP (np. Storefront, konta klientów). Rola serwera jest celowo wąska: oferuje on „narzędzia” (funkcje wykonywalne) i „zasoby” (dane strukturalne), ale nie posiada wiedzy o szerszym kontekście konwersacji.4Taka zasada projektowania gwarantuje łatwość budowy i konserwacji serwerów.
* **Klient:** Klient nawiązuje sesję jeden do jednego z serwerem, uwzględniając stan. W przypadku aplikacji partnerskiej Shopify, sama aplikacja zaplecza pełni rolę klienta MCP. Jest ona odpowiedzialna za kierowanie wiadomościami, zarządzanie cyklem życia połączenia i obsługę negocjacji protokołu z serwerem.4Każde połączenie jest izolowane, co oznacza, że ​​klient połączony z serwerem Shopify Storefront działa niezależnie od połączenia z innym serwerem.
* **Host:** Host jest koordynatorem i kontenerem dla jednej lub więcej instancji klienta. W typowej aplikacji chatbota Host stanowi rdzeń logiki, który integruje się z Modelem Dużego Języka (LLM). Jest to najbardziej uprzywilejowany komponent architektury, odpowiedzialny za zarządzanie całym przepływem konwersacji, agregację kontekstu od wielu klientów, koordynację wywołań narzędzi i – co najważniejsze – egzekwowanie zasad bezpieczeństwa i zgody użytkownika.4Gospodarz przechowuje pełną historię konwersacji i decyduje, jakie ograniczone, niezbędne informacje przekazać danemu serwerowi, zapewniając w ten sposób, że serwery działają w oparciu o zasadę ograniczonego dostępu.

Ten podział architektoniczny skutkuje kilkoma podstawowymi zasadami, które są kluczowe dla tworzenia systemów AI klasy korporacyjnej:

* **Kompozycyjność:** Rozróżnienie między klientem a serwerem ma charakter logiczny, a nie fizyczny. Aplikacja może pełnić obie funkcje jednocześnie.4Na przykład, główny agent AI (pełniący rolę hosta i klienta) może komunikować się ze specjalistycznym podagentem (pełniącym rolę serwera). Ten podagent może z kolei pełnić rolę klienta dla innych serwerów MCP (np. serwera logistyki wysyłkowej lub niestandardowego serwera RAG). Umożliwia to tworzenie złożonych, hierarchicznych i modułowych systemów agentowych, w których poszczególni agenci specjalizują się w określonych zadaniach i delegują podzadania innym.4
* **Bezpieczeństwo poprzez izolację:** Najważniejszą cechą architektoniczną MCP jest jego model bezpieczeństwa. Serwery są ściśle izolowane. Nie mogą odczytać pełnej historii konwersacji ani „zajrzeć” do innych serwerów.4Host pełni funkcję strażnika bezpieczeństwa, kontrolując wszystkie interakcje między serwerami i zapewniając, że serwer otrzymuje jedynie minimalny kontekst wymagany do wykonywania swojej funkcji. Zapobiega to ujawnieniu poufnych informacji przetwarzanych przez inny serwer (np. serwer kont klientów Shopify) przez hipotetyczną lukę w zabezpieczeniach jednego serwera narzędzi (np. zewnętrznego interfejsu API pogodowego).

Sama komunikacja odbywa się za pośrednictwem JSON-RPC 2.0, lekkiego protokołu zdalnego wywoływania procedur.4Definiuje trzy podstawowe typy wiadomości: Żądania (dwukierunkowe wiadomości oczekujące odpowiedzi), Odpowiedzi (wyniki lub błędy) i Powiadomienia (jednokierunkowe wiadomości niewymagające odpowiedzi).4Ten standaryzowany protokół uwzględniający stan gwarantuje niezawodne i przewidywalne interakcje w całym cyklu życia połączenia — od inicjalizacji i negocjacji możliwości po łagodne zamknięcie.4

### **1.3 Poruszanie się po ekosystemie Shopify MCP**

Implementacja protokołu Model Context Protocol w Shopify nie jest monolityczną całością, lecz starannie posegmentowanym ekosystemem odrębnych serwerów, z których każdy został zaprojektowany z myślą o konkretnym celu, użytkowniku i kontekście bezpieczeństwa. Zrozumienie roli tych serwerów to pierwszy krok w projektowaniu spójnej i efektywnej architektury aplikacji AI.

* **Storefront MCP:** To główny silnik dla wszystkich interakcji z klientami, bez uwierzytelniania. Jest to serwer publiczny, a każdy sklep Shopify ma aktywny punkt końcowy dostępny pod adresem https://{your-store-name}.myshopify.com/api/mcp.3Celem jest umożliwienie handlu konwersacyjnego poprzez udostępnienie podstawowych funkcjonalności, których potrzebuje klient: wyszukiwanie produktów w języku naturalnym, zarządzanie koszykiem (dodawanie, aktualizowanie, usuwanie produktów) oraz pobieranie regulaminu sklepu lub odpowiedzi na często zadawane pytania.2Serwer ten jest przeznaczony do obsługi dużego ruchu i użytku w czasie rzeczywistym przez asystentów zakupowych i chatboty wykorzystujące sztuczną inteligencję.
* **Konta klientów MCP:** Ten serwer obsługuje uwierzytelnione działania użytkownika po zalogowaniu. Zapewnia dostęp do chronionych danych klienta i dlatego jest zabezpieczony bardziej złożonym procesem uwierzytelniania OAuth 2.0.9Narzędzia udostępniane przez ten serwer są spersonalizowane i wrażliwe. Obejmują one m.in. takie możliwości, jak sprawdzanie statusu konkretnego zamówienia, przeglądanie historii zamówień i zarządzanie zwrotami.7Agent AI będzie wchodził w interakcję z serwerem dopiero po bezpiecznym zalogowaniu się klienta, co umożliwi nową klasę zautomatyzowanej, spersonalizowanej obsługi klienta.
* **Dev MCP:** Ten serwer to unikalne narzędzie przeznaczone dla programistów, a nie dla klientów końcowych. Działa lokalnie w środowisku programistycznym i łączy asystenta AI, takiego jak ten zintegrowany ze środowiskiem programistycznym (IDE), takim jak Cursor, lub aplikację desktopową, taką jak Claude Desktop, bezpośrednio z zasobami programistycznymi Shopify.3Serwer Dev MCP może przeszukiwać obszerną dokumentację deweloperską Shopify, analizować schematy GraphQL interfejsu API administratora i udostępniać aktualne, kontekstowe wskazówki dotyczące tworzenia aplikacji i motywów Shopify.3Działa jako ekspercki drugi pilot, przyspieszając rozwój poprzez dostarczanie dokładnych przykładów kodu i informacji o interfejsie API bezpośrednio w trakcie pracy programisty. Dzięki temu sugestie sztucznej inteligencji opierają się na bieżących najlepszych praktykach, a nie na przestarzałych danych szkoleniowych.10

Strategiczne rozdzielenie tych serwerów stanowi bezpośrednie zastosowanie zasad kompozycyjności i bezpieczeństwa MCP. Pozwala to na bezproblemowe współistnienie różnych modeli zabezpieczeń i strategii skalowania. Nieuwierzytelniony serwer Storefront o dużej przepustowości można zaprojektować pod kątem wydajności i publicznego dostępu, a serwer kont klientów chroniony protokołem OAuth można zabezpieczyć, aby rygorystycznie przetwarzać dane osobowe (PII). Natomiast lokalny serwer Dev MCP nie stanowi zagrożenia dla danych w magazynie. To modułowe podejście stanowi wzór zastosowania wzorca MCP w złożonej domenie, takiej jak handel, sugerując, że deweloperzy tworzący zaawansowane aplikacje powinni myśleć o koordynacji wielu wyspecjalizowanych klientów MCP w ramach własnej architektury, zamiast tworzyć jednego, monolitycznego klienta.

| **Tabela 1: Porównanie serwerów Shopify MCP** |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nazwa serwera** | **Główny cel** | **Użytkownik docelowy** | **Metoda uwierzytelniania** | **Kluczowe narzędzia ujawnione** | **Typowe środowisko hosta** |
| Sklep MCP | Handel konwersacyjny w czasie rzeczywistym | Kupujący / Klienci | Brak (punkt końcowy publiczny) | search\_shop\_catalog, update\_cart,search\_shop\_policies\_and\_faqs | Serwer zdalny (zaplecze aplikacji partnerskiej Shopify) |
| Konta klientów MCP | Spersonalizowane zarządzanie kontem po zalogowaniu | Zalogowani klienci | OAuth 2.0 z PKCE | get\_order\_status, get\_orders, zarządzaj zwrotami | Serwer zdalny (zaplecze aplikacji partnerskiej Shopify) |
| Dev MCP | Rozwój wspomagany sztuczną inteligencją i eksploracja API | Deweloperzy Shopify | Brak (proces lokalny) | search\_docs, introspect\_schema,validate\_liquid | Lokalne środowisko IDE (np. Cursor) lub aplikacja komputerowa (np. Claude) |

## **Część II: Podstawowa architektura aplikacji chatbota Shopify MCP**

W tej sekcji przełożono podstawowe koncepcje MCP na konkretny projekt architektoniczny aplikacji partnerskiej Shopify. Szczegółowo opisano główne komponenty, ich obowiązki oraz kluczowe wzorce interakcji, które tworzą solidne i skalowalne rozwiązanie handlu konwersacyjnego.

### **2.1 Anatomia aplikacji Shopify opartej na MCP**

Standardową architekturą chatbota opartego na MCP jest aplikacja partnerska Shopify składająca się z dwóch głównych komponentów: usługi zaplecza, która pełni funkcję klienta MCP i centrum koordynacji, oraz interfejsu użytkownika front-end dostarczanego jako rozszerzenie aplikacji motywu.7Tę strukturę ilustruje własna implementacja referencyjna Shopify,

shop-chat-agentrepozytorium, które zapewnia szablon do tworzenia takich aplikacji.12

Cykl rozwoju rozpoczyna się w ekosystemie partnerów Shopify. Konto partnera Shopify jest wymagane do tworzenia i zarządzania aplikacją, a sklep deweloperski z przykładowymi produktami jest niezbędny do testowania i iteracji.12Cała aplikacja, łącznie z rozszerzeniami zaplecza i frontendu, jest oparta na platformie i zarządzana za pomocą interfejsu wiersza poleceń Shopify, który usprawnia procesy tworzenia, konfiguracji i wdrażania.

Logiczny przebieg interakcji użytkownika jest następujący:

1. Użytkownik korzysta z widżetu czatu w witrynie sklepowej Shopify.
2. Komponent front-end (rozszerzenie aplikacji motywu) wysyła wiadomość użytkownika do usługi zaplecza aplikacji.
3. Usługa zaplecza, pełniąca rolę hosta i klienta MCP, przetwarza wiadomość. Organizuje analizę LLM, aby określić intencje użytkownika i zdecydować, czy wymagane jest podjęcie działania specyficznego dla Shopify.
4. Jeśli konieczne jest podjęcie działania, zaplecze wykonuje wywołanie JSON-RPC do odpowiedniego serwera Shopify MCP (np. Storefront MCP w celu wyszukania produktu).
5. Serwer Shopify MCP przetwarza żądanie, wchodzi w interakcję z danymi sklepu i zwraca ustrukturyzowaną odpowiedź.
6. Zaplecze odbiera odpowiedź, uwzględnia ją w kontekście konwersacji i generuje odpowiedź dla użytkownika, która może zawierać tekst lub interaktywne elementy interfejsu użytkownika.
7. Odpowiedź ta jest odsyłana do front-endu, który renderuje ją dla użytkownika, zamykając w ten sposób pętlę.

### **2.2 Zaplecze: Klient MCP i centrum orkiestracji**

Usługa back-endowa to centralny układ nerwowy aplikacji. Odpowiada za całą logikę biznesową, zarządzanie stanem i komunikację zewnętrzną. Do budowy tej usługi powszechnie stosuje się Node.js, często w połączeniu z nowoczesnym frameworkiem internetowym, takim jak Remix, który jest domyślnym frameworkiem w oficjalnych szablonach aplikacji Shopify, lub Express.13

Podstawowe obowiązki zaplecza obejmują:

* **Implementacja klienta MCP:** Działa jako klient MCP, ustanawiając i utrzymując połączenia stanowe z serwerami MCP Shopify.7Polega ona na tworzeniu i wysyłaniu poprawnie sformatowanych żądań JSON-RPC 2.0 do punktów końcowych serwera (np.  
  tools/calldo wywołania narzędzia) i analizowania odpowiedzi.4Chociaż aplikacja funkcjonuje jako klient, analiza implementacji serwera MCP typu open source może dostarczyć cennych wzorców do obsługi cyklu żądanie/odpowiedź protokołu.14
* **Orkiestracja LLM:** To warstwa „inteligencji”, w której zaplecze komunikuje się z Modelem LLM (np. poprzez API Claude lub OpenAI). Jest ona odpowiedzialna za zbieranie danych wprowadzanych przez użytkownika, umieszczanie ich w starannie opracowanym komunikacie i instruowanie LLM, aby użył dostępnych narzędzi MCP do spełnienia żądania użytkownika. Proces ten jest szczegółowo opisany w Części III.
* **Punkt końcowy API dla frontendu:** Backend udostępnia API (np. punkt końcowy REST lub GraphQL), z którym komunikuje się widżet czatu frontendu. Ten punkt końcowy obsługuje przychodzące wiadomości użytkowników i przesyła strumieniowo odpowiedzi sztucznej inteligencji.
* **Zarządzanie uwierzytelnianiem:** Zaplecze jest odpowiedzialne za bezpieczne przechowywanie i używanie niezbędnych danych uwierzytelniających API w celu komunikacji zarówno z interfejsami API Shopify Admin/Storefront, jak i serwerami Shopify MCP.

### **2.3 Frontend: rozszerzenie aplikacji motywu i interfejs użytkownika MCP**

Komponent chatbota skierowany do użytkownika najlepiej wdrożyć w formie **rozszerzenia aplikacji motywu** Shopify .7Dzięki takiemu podejściu widżet czatu można bezproblemowo osadzić w motywie dowolnego sprzedawcy za pomocą prostego przełącznika w narzędziu dostosowywania motywu, co zapewnia natywny wygląd i styl bez konieczności ręcznej edycji kodu.12Rozszerzenie składa się ze standardowych zasobów internetowych — HTML, CSS i JavaScript — które odpowiadają za renderowanie interfejsu czatu i zarządzanie komunikacją z zapleczem.

Najbardziej przełomowym aspektem architektury front-end jest integracja **interfejsu użytkownika MCP** . Interfejs czatu oparty wyłącznie na tekście ogranicza możliwości użytkownika, szczególnie w tak bogatej wizualnie dziedzinie jak handel.17Interfejs użytkownika MCP przełamuje tę „ścianę tekstu”, umożliwiając agentowi AI zwracanie w pełni interaktywnych, wizualnych komponentów bezpośrednio do okna czatu.3

Wdrożenie obsługi interfejsu użytkownika MCP nakłada określone wymagania na front-end:

1. **Renderowanie komponentów interaktywnych:** Frontend musi być w stanie odbierać i renderować zasoby interfejsu użytkownika przesyłane z backendu. Zasoby te mogą być dostarczane w kilku formatach, w tym w formacie HTML inline lub zdalnych identyfikatorach URI zasobów, które są następnie ładowane do piaskownicy <iframe>w celu zapewnienia bezpieczeństwa i izolacji stylów.17Dzięki temu złożone komponenty, takie jak karuzele produktów z selektorami wariantów, galerie obrazów i przyciski dodawania do koszyka, mogą być natywnie wyświetlane w konwersacji.3
2. **Obsługa systemu intencji:** Komponenty interaktywne stwarzają problem z synchronizacją stanu. Jeśli użytkownik kliknie przycisk „Dodaj do koszyka” w oknie dialogowym <iframe>, agent AI, który kontroluje stan konwersacji, nie będzie świadomy tej czynności. Interfejs użytkownika MCP rozwiązuje ten problem dzięki eleganckiemu **systemowi komunikatów opartemu na intencjach** .17Zamiast bezpośredniej modyfikacji stanu, komponent interfejsu użytkownika emituje zdarzenie „intencji” (np.  
   view\_details, checkout, lub notify). JavaScript front-endu nasłuchuje tych intencji, przechwytuje ich dane (np. identyfikator wariantu produktu do dodania) i przekazuje te ustrukturyzowane informacje do back-endu. Taka architektura zapewnia, że ​​agent AI pozostaje jedynym źródłem prawdy dla stanu konwersacji, umożliwiając mu przetworzenie intencji, wywołanie odpowiedniego narzędzia MCP (takiego jak update\_cart), a następnie podjęcie decyzji o aktualizacji konwersacji.17

Ten projekt pokazuje, że głównym wyzwaniem związanym z tworzeniem zaawansowanego chatbota MCP nie jest wyłącznie sztuczna inteligencja, ale zarządzanie stanem w trzech odrębnych domenach: kontekstu konwersacyjnego LLM, stanu sklepu Shopify w czasie rzeczywistym (np. zawartość koszyka, stan magazynowy) oraz stanu interfejsu użytkownika. Protokoły MCP i MCP UI zapewniają skuteczny wzorzec centralizacji zarządzania stanem w agencie AI, tworząc bardziej spójny i solidny system. Zadanie programisty przenosi się z tradycyjnego programowania interfejsu użytkownika na projektowanie agenta sterowanego zdarzeniami, zdolnego do płynnej obsługi asynchronicznego strumienia intencji generowanych przez użytkownika i komponenty.

### **2.4 Wzorce uwierzytelniania i autoryzacji**

Bezpieczna i niezawodna strategia uwierzytelniania jest nieodzownym warunkiem wstępnym dla każdej produkcyjnej aplikacji Shopify. Chatbot oparty na platformie MCP wymaga wielowarstwowego podejścia do obsługi różnych poziomów dostępu do danych.

* **Dostęp do interfejsu API administratora i witryny sklepowej:** podstawowa funkcjonalność aplikacji opiera się na danych uwierzytelniających API uzyskanych poprzez utworzenie **aplikacji niestandardowej** w Panelu partnera Shopify lub w panelu administracyjnym sklepu sprzedawcy.16Podczas tej konfiguracji programista musi zażądać określonych zakresów API, które przyznają niezbędne uprawnienia. W przypadku typowego chatbota zakresy te obejmują:
  + read\_products, read\_product\_listings: Aby wyszukać i pobrać informacje o produkcie.
  + unauthenticated\_write\_checkouts, unauthenticated\_read\_checkouts: Zarządzanie koszykiem zakupowym za pomocą interfejsu API Storefront.19
  + read\_customers, write\_customers, read\_orders, write\_orders: Aby uzyskać dostęp do bardziej zaawansowanych funkcji wymagających dostępu do interfejsu API administratora.16

Wygenerowany token dostępu do interfejsu API administratora i token dostępu do interfejsu API witryny sklepowej należy bezpiecznie przechowywać jako zmienne środowiskowe w zapleczu. Nigdy nie należy go ujawniać po stronie klienta ani przekazywać do systemu kontroli wersji.14

* **Przepływ OAuth 2.0 w Kontach Klientów MCP:** Dostęp do chronionych danych klienta, takich jak historia zamówień użytkownika, wymaga znacznie wyższego poziomu bezpieczeństwa. Za obsługę tego procesu odpowiada **serwer MCP Kont Klientów** , który jest chroniony przez **przepływ kodu autoryzacyjnego OAuth 2.0 z kluczem kontrolnym do wymiany kodu (PKCE)** .9Jest to branżowy standard zabezpieczający aplikacje, w których nie można bezpiecznie przechowywać informacji poufnych, np. w aplikacjach front-end.  
  Wdrożenie tego przepływu jest procesem wieloetapowym:
  1. **Konfiguracja:** Plik aplikacji shopify.app.tomlmusi zostać zaktualizowany w celu zadeklarowania wymaganych zakresów danych klienta i zarejestrowania jednego lub większej liczby plików redirect\_uris.9
  2. **Żądanie dostępu:** Deweloper aplikacji musi formalnie wystąpić o dostęp do chronionych danych klienta (PII) za pośrednictwem Panelu partnera Shopify.9
  3. **Inicjowanie przepływu:** Gdy agent AI musi wykonać uwierzytelnioną czynność (np. „sprawdź moje ostatnie zamówienie”), zaplecze najpierw próbuje wywołać narzędzie MCP. Otrzyma 401 Unauthorizedodpowiedź.9Ta odpowiedź jest wyzwalaczem inicjującym przepływ OAuth.
  4. **Autoryzacja:** Zaplecze steruje działaniem front-endu, aby przekierował użytkownika do adresu URL autoryzacji Shopify, w tym parametrów, takich jak client\_id, scope, redirect\_urii code\_challengewygenerowanych dla PKCE.
  5. **Zgoda użytkownika:** Użytkownikowi wyświetla się ekran zgody, na którym Shopify prosi go o udzielenie aplikacji uprawnień dostępu do jego danych.
  6. **Wywołanie zwrotne i wymiana tokenów:** Po wyrażeniu zgody Shopify przekierowuje użytkownika z powrotem do zarejestrowanego konta redirect\_uriz kodem autoryzacyjnym. System zaplecza otrzymuje ten kod i wraz z oryginalnym kodem code\_verifierwymienia go na token dostępu i token odświeżania, wykonując bezpieczne połączenie między serwerami do punktu końcowego tokena Shopify.9
  7. **Secure Storage and Retry:** The access token is securely stored (e.g., in an encrypted session or database) and used to retry the original MCP tool call, which will now succeed.

This dual-authentication model ensures that general, unauthenticated shopping queries are fast and frictionless, while access to sensitive personal data is gated behind a robust, user-consent-driven security protocol.

## Part III: AI Agent Orchestration and Conversational Flow

This part delves into the "intelligence" layer of the application, focusing on the techniques and frameworks required to transform a Large Language Model from a simple text generator into a sophisticated e-commerce agent. It details how to program the AI to reason, use MCP tools effectively, manage conversational state, and deliver a seamless user experience.

### 3.1 Designing the AI Agent's "Brain": Tool-Use and Reasoning

The core capability that elevates a chatbot to an AI agent is **tool-use**, also known as function calling.21 Instead of merely generating text based on its training data, the LLM is given access to a suite of external tools—in this case, the functions exposed by Shopify's MCP servers. When presented with a user's query, the model's task is not just to answer, but to first reason about the user's intent and then decide which tool, if any, should be called to fulfill the request.

This process is governed by a reasoning loop, a pattern best exemplified by the **ReAct (Reason-Act-Observe)** framework.23 An agent following this pattern cycles through a sequence of steps:

1. **Reason:** Based on the user's prompt and the conversational history, the LLM "thinks" about the problem. It formulates a plan, identifies the necessary information, and determines which tool can provide it. For example, for the query "Do you have any red running shoes under $100?", the model would reason that it needs to search the product catalog with filters for category, color, and price.
2. **Act:** The LLM generates a structured request to call a specific tool with the required arguments. For the example above, it would generate a call to the search\_shop\_catalog tool with parameters like {"query": "running shoes", "filters": {"color": "red", "price\_max": 100}}.
3. **Observe:** The application's backend (the Host) executes this tool call via the MCP client. The result from the Shopify MCP server—a list of products or an error message—is then passed back to the LLM as an "observation."
4. **Repeat:** The LLM processes this new information. If the observation answers the user's question, it synthesizes the final response. If the initial tool call was insufficient or led to a new question, it re-enters the reasoning phase to determine the next step, potentially calling another tool.

Effective tool-use is highly dependent on the quality of the tool descriptions and the system prompt provided to the LLM. The descriptions for each MCP tool must be clear and comprehensive, explaining what the tool does and what its parameters are, so the model can accurately determine when to use it.21 The system prompt sets the overall context for the agent, defining its persona, its objectives, and the rules of engagement for using its tools.24

### 3.2 Orchestration with LangChain & LlamaIndex

Implementing a robust reasoning loop, managing tool calls, and parsing structured outputs from an LLM can be a complex engineering task. To abstract this complexity, developers can leverage high-level AI orchestration frameworks like LangChain and LlamaIndex. These frameworks provide the building blocks for creating sophisticated, tool-using agents.

* **LangChain:** LangChain offers a mature ecosystem for building agentic applications. A key component for this use case is the **LangChain MCP Adapters** package, which simplifies integration significantly.23 The  
  MultiServerMCPClient adapter can connect to one or more MCP servers and automatically introspects the available tools via the tools/list command. It then translates these MCP tools into a format that is natively understood by LangChain agents.23 This allows a developer to bypass the manual implementation of JSON-RPC calls and focus on the agent's logic. For more complex, multi-step workflows, LangChain's  
  **LangGraph** library can be used to define the agent's control flow as a state machine, enabling advanced features like conditional logic, error recovery, and parallel tool execution.23
* **LlamaIndex:** LlamaIndex also provides robust support for MCP integration through its llama-index-tools-mcp package.27 The primary components are the  
  BasicMCPClient, which handles the low-level communication with the MCP server (supporting HTTP, SSE, and stdio transports), and the McpToolSpec, which serves as a wrapper to fetch tools from the client and convert them into LlamaIndex FunctionTool objects.27 This allows developers to seamlessly incorporate MCP-provided functionalities into their LlamaIndex agents. For developers who may need to interact directly with the Shopify GraphQL API as a fallback or for custom functionalities not exposed via MCP, the separate  
  llama-index-tools-shopify package provides a ShopifyToolSpec for executing raw GraphQL queries.30

The choice between these frameworks often comes down to developer preference and the specific architectural needs of the project, but both provide a powerful abstraction layer that accelerates the development of MCP-powered agents.

### 3.3 Mastering Conversational Flow and State Management

A successful AI shopping assistant must provide a coherent and context-aware conversational experience. This requires meticulous state management to track the conversation's history and the user's evolving intent.

* **Utrzymywanie kontekstu w pamięci:** Agent musi „pamiętać” poprzednie etapy rozmowy, aby móc odpowiadać na pytania uzupełniające i utrzymywać kontekst. Osiąga się to poprzez implementację komponentu **pamięci konwersacyjnej** . Frameworki takie jak LangChain oferują różne typy pamięci, takie jak ConversationBufferMemory, który przechowuje historię czatu i uwzględnia ją w kolejnych wywołaniach LLM.23Dzięki temu agent może płynnie obsługiwać rozmowy wieloetapowe. Na przykład:
  1. Użytkownik: „Pokaż mi kilka desek snowboardowych.”
  2. Agent: (Dzwoni search\_shop\_catalog, pokazuje wyniki)
  3. Użytkownik: „Czy masz jakieś dobre dla początkujących?”
  4. Agent: (Bez pamięci to zapytanie jest bezsensowne. Dzięki pamięci agent rozumie, że „any” odnosi się do wcześniej omówionych desek snowboardowych i może zawęzić wyszukiwanie).32
* **Projektowanie przepływów konwersacji:** Logika agenta powinna być zaprojektowana tak, aby prowadzić użytkownika przez typowy proces zakupowy — od początkowego zapoznania się z produktem do ostatecznego zakupu.3Polega to na orkiestracji sekwencji stanów i wywołań narzędzi. Na przykład przepływ może zaczynać się od wyszukiwania produktu, prowadzić do wyjaśnienia wariantów (rozmiar/kolor), prezentować interaktywną kartę produktu (za pośrednictwem interfejsu użytkownika MCP), obsługiwać  
  add\_to\_cartintencję i na koniec zachęca użytkownika do finalizacji zakupu.
* **Radzenie sobie ze zmianami kontekstu:** Użytkownicy nie zawsze podążają liniową ścieżką. Mogą zmienić zdanie lub zadać niezwiązane z tematem pytania w trakcie procesu (np. pytając o zasady wysyłki po dodaniu produktu do koszyka).32Agent musi być wystarczająco odporny, aby poradzić sobie z tymi przerwami i odpowiedzieć na nowe zapytanie, wywołując odpowiednie narzędzie (np.  
  search\_shop\_policies\_and\_faqs), a następnie łagodnie powrócić do poprzedniego kontekstu lub zapytać użytkownika, jaki jest jego preferowany sposób postępowania.

### **3.4 Ulepszanie wrażeń użytkownika dzięki przesyłaniu strumieniowemu odpowiedzi**

Kilkusekundowe oczekiwanie na wygenerowanie kompletnej odpowiedzi przez LLM skutkuje niskim komfortem użytkowania. Aby zapewnić bardziej interaktywne i responsywne wrażenie, odpowiedzi agenta powinny być **przesyłane strumieniowo** do front-endu w miarę ich generowania, token po tokenie, tworząc efekt „maszyny do pisania”.7Dzięki temu odczuwalne opóźnienie jest znacznie mniejsze, a użytkownik pozostaje zaangażowany.34

Istnieją dwa podstawowe podejścia techniczne do implementacji przesyłania strumieniowego z zaplecza Node.js do front-endu internetowego:

1. **Server-Sent Events (SSE):** SSE to prosty i wydajny protokół komunikacji jednokierunkowej zbudowany na bazie protokołu HTTP, co czyni go doskonałym rozwiązaniem w tym przypadku użycia.34
   * **Zaplecze:** Serwer Node.js ustawia nagłówki odpowiedzi na Content-Type: text/event-stream, Cache-Control: no-cachei Connection: keep-alive. Gdy fragmenty tekstu są odbierane z LLM, serwer zapisuje je w strumieniu odpowiedzi w formacie data: {chunk\_text}\n\n. Można wysłać komunikat końcowy event: done, aby zasygnalizować koniec strumienia.34
   * **Frontend:** JavaScript po stronie klienta wykorzystuje natywne EventSourceAPI do połączenia z punktem końcowym strumieniowania. onmessageNasłuchiwacz zdarzeń odbiera każdy fragment danych, który jest następnie dołączany do ekranu czatu.34
2. **Pobierz API z ReadableStreams:** To nowocześniejsze i bardziej elastyczne podejście, które daje deweloperowi kontrolę niższego poziomu nad strumieniem.34
   * **Zaplecze:** Serwer ustawia Transfer-Encoding: chunkednagłówek. Następnie zapisuje surowe fragmenty tekstu z LLM bezpośrednio do strumienia odpowiedzi bez żadnego specjalnego formatowania.34
   * **Frontend:** Klient wysyła fetchżądanie . Kod pobiera czytnik ze strumienia ( ) i wchodzi w pętlę, wielokrotnie wywołując fragmenty przetwarzania w miarę ich pojawiania się response.body, aż do zakończenia strumienia.ReadableStreamresponse.body.getReader()await reader.read()34

Naprawdę zaawansowana implementacja wykracza poza strumieniowe przesyłanie tylko ostatecznej odpowiedzi tekstowej. Przesyła strumieniowo cały proces wnioskowania agenta. Zapewnia to użytkownikowi cenną przejrzystość działań agenta, wyświetlając komunikaty o stanie, takie jak „Myślę…”, „Szukam produktów…” lub „Aktualizuję koszyk”, zanim ostateczna odpowiedź zostanie zsyntetyzowana. Wymaga to bardziej złożonej, sterowanej zdarzeniami architektury, zdolnej do obsługi mieszanego strumienia tokenów tekstowych, wskaźników wywołań narzędzi i żądań renderowania interfejsu użytkownika, ale zapewnia znacznie lepsze wrażenia użytkownika.

## **Część IV: Głębokie zanurzenie w integrację podstawowych narzędzi**

Ta część zawiera praktyczną, zorientowaną na implementację analizę najważniejszych narzędzi platformy Shopify MCP. Łączy ona koncepcje architektoniczne i orkiestracyjne z poprzednich sekcji z konkretnymi działaniami e-commerce, które napędzają konwersacyjne doświadczenie zakupowe. Poniższa tabela zawiera ogólny przegląd kluczowych narzędzi dostępnych za pośrednictwem nieuwierzytelnionego serwera Storefront MCP.

| **Tabela 2: Podstawowe narzędzia witryny sklepowej MCP** |  |  |  |
| --- | --- | --- | --- |
| **Nazwa narzędzia** | **Opis** | **Kluczowe parametry wejściowe** | **Oczekiwana struktura wyjściowa** |
| search\_shop\_catalog | Przeszukuje katalog produktów sklepu, wykorzystując zapytania w języku naturalnym i filtry strukturalne. | query: (string) Termin wyszukiwania w języku naturalnym. filters: (object, opcjonalnie) Ustrukturyzowane filtry dotyczące ceny, dostępności, koloru itp. | Lista obiektów produktu, z których każdy zawiera identyfikator, tytuł, opis, warianty, cenę i adresy URL obrazów. Może również zwracać zasoby interfejsu użytkownika MCP. |
| update\_cart | Modyfikuje zawartość bieżącego koszyka. Umożliwia dodawanie, aktualizowanie i usuwanie pozycji. | lines: (tablica obiektów) Tablica elementów zamówienia, z których każdy określa merchandiseId(identyfikator wariantu) i quantity. Ilość równa 0 usuwa element. | Zaktualizowany obiekt koszyka, zawierający wszystkie pozycje zamówienia, sumę częściową i adres URL realizacji zamówienia. Może również zwracać zasoby interfejsu użytkownika MCP. |
| search\_shop\_policies\_and\_faqs | Pobiera informacje z regulaminu sklepu i aplikacji Shopify Knowledge Base. | query: (string) Pytanie użytkownika dotyczące zasad obowiązujących w sklepie (np. „Jaka jest polityka zwrotów?”). | Odpowiedź tekstowa oparta na odpowiednich dokumentach polityki lub wpisach w sekcji FAQ. |

### **4.1 Odkrywanie produktów z**search\_shop\_catalog

Narzędzie to search\_shop\_catalogstanowi podstawę wyszukiwania produktów w interfejsie konwersacyjnym. Umożliwia ono agentowi AI tłumaczenie zapytania użytkownika w języku naturalnym, takiego jak „pokaż mi czerwone buty do biegania poniżej 100 dolarów”, na ustrukturyzowane zapytanie, które można wykonać w katalogu produktów Shopify.3Rolą LLM jest analiza intencji użytkownika i wyodrębnienie odpowiednich elementów — typu produktu („buty do biegania”), koloru („czerwony”) i ceny (

< 100) — i sformatuj je jako argumenty wywołania narzędzia.32

Skuteczność tego narzędzia zależy zasadniczo od jakości i struktury danych produktowych w sklepie Shopify. Absolwent LLM nie jest w stanie znaleźć informacji, których brakuje lub które są źle zorganizowane.36Aby narzędzie działało optymalnie, sprzedawcy muszą zadbać o to, aby dane dotyczące ich produktów były kompleksowe i dobrze ustrukturyzowane:

* **Szczegółowe opisy produktów:** Opisy powinny zawierać informacje i być napisane naturalnym językiem, który przewiduje pytania klientów.36
* **Ustrukturyzowane dane i taksonomia:** Prawidłowe użycie typów produktów, tagów i wariantów jest kluczowe. Implementacja znaczników schematu pomaga modelom sztucznej inteligencji (i tradycyjnym wyszukiwarkom) lepiej zrozumieć atrybuty produktu, takie jak materiał, rozmiar i kolor.38
* **Wysokiej jakości obrazy:** obrazy z opisowym tekstem alternatywnym zapewniają dodatkowy kontekst, który można wykorzystać do wyszukiwania i rekomendacji.38

Gdy search\_shop\_catalognarzędzie zwraca listę pasujących produktów, najlepszą praktyką jest przedstawienie tych wyników w formie wizualnej, a nie w postaci prostej listy tekstowej. Jest to główny przypadek użycia interfejsu użytkownika MCP. Agent zaplecza może zażądać od serwera MCP sformatowania wyników jako interaktywnej karuzeli produktów, która jest następnie renderowana bezpośrednio w oknie czatu, umożliwiając użytkownikowi przewijanie obrazów, przeglądanie cen i wybieranie opcji bez opuszczania konwersacji.3

### **4.2 Zarządzanie koszykiem z**update\_cart

Narzędzie to update\_cartstanowi transakcyjne serce chatbota, umożliwiające wprowadzanie wszelkich modyfikacji w koszyku zakupowym użytkownika.3To wszechstronne narzędzie, które obsługuje dodawanie pozycji, zmianę ilości i usuwanie pozycji na podstawie listy pozycji podanej w argumentach. Na przykład, aby dodać konkretny wariant produktu do koszyka, agent powinien zadzwonić pod numer

update\_cartz linestablicą zawierającą obiekt z merchandiseIdwariantem i pożądanym quantity.40Aby usunąć element, należy ustawić ilość na 0.

Synchronizacja stanu jest kluczowa podczas korzystania z tego narzędzia. Agent konwersacyjny musi pozostać jedynym źródłem informacji o stanie koszyka. Idealny przepływ interakcji wykorzystuje system intencji interfejsu użytkownika MCP:

1. Na podstawie wyników użytkownikowi prezentowana jest interaktywna karta produktu search\_shop\_catalog.
2. Użytkownik wybiera wariant „Duży” „Niebieski” i klika przycisk „Dodaj do koszyka” w komponencie interfejsu użytkownika MCP.
3. Komponent interfejsu użytkownika nie modyfikuje bezpośrednio koszyka. Zamiast tego uruchamia add\_to\_cartintencję zawierającą wybrane merchandiseId( gid://shopify/ProductVariant/...) i ilość.3
4. Aplikacja front-end przechwytuje tę intencję i przesyła ją do koordynatora back-end.
5. Agent AI odbiera intencję, rozumie działanie użytkownika i formalnie wywołuje update\_cartnarzędzie MCP z prawidłowymi parametrami.
6. Po otrzymaniu pomyślnej odpowiedzi od update\_cartnarzędzia agent może potwierdzić działanie użytkownikowi, wysyłając mu wiadomość tekstową („Dodałem niebieską koszulkę do koszyka”) i potencjalnie poprosić interfejs użytkownika o odświeżenie komponentu interfejsu użytkownika koszyka w celu odzwierciedlenia nowego stanu.17

Ten proces działający w zamkniętej pętli gwarantuje, że wewnętrzny stan agenta, rzeczywisty koszyk Shopify i wizualna reprezentacja koszyka przez użytkownika pozostają idealnie zsynchronizowane.

### **4.3 Pobieranie zasad za pomocą**search\_shop\_policies\_and\_faqs

Dzięki temu narzędziu agent AI może pełnić rolę bezpośredniego przedstawiciela obsługi klienta, odpowiadając na często zadawane pytania dotyczące zasad obowiązujących w sklepie, takich jak wysyłka, zwroty, prywatność i inne.3Dzięki temu personel pomocniczy zostaje odciążony od konieczności odpowiadania na znaczną liczbę powtarzających się zapytań.

Kluczową informacją dla programistów jest to, że wydajność tego narzędzia jest bezpośrednio i wyłącznie powiązana z treścią zarządzaną przez sprzedawcę w **aplikacji Shopify Knowledge Base** .41Ta aplikacja stanowi jedyne źródło informacji dla sztucznej inteligencji. Automatycznie pobiera informacje z ustawień sklepu (takich jak strefy wysyłki i zasady zwrotów) i umożliwia sprzedawcom ręczne tworzenie, edytowanie i zastępowanie niestandardowych często zadawanych pytań (FAQ).41

Aby search\_shop\_policies\_and\_faqsnarzędzie było skuteczne, aplikacja Shopify Partner musi uświadamiać swoim użytkownikom, jak ważne jest dbanie o bazę wiedzy. Najlepsze praktyki obejmują:

* **Tworzenie niestandardowych często zadawanych pytań:** Sprzedawcy powinni proaktywnie dodawać jasne, zwięzłe pary pytań i odpowiedzi dotyczące tematów nieobjętych domyślnymi ustawieniami sklepu.41
* **Przeglądanie pytań bez odpowiedzi:** Aplikacja Baza Wiedzy dostarcza metryk dotyczących pytań klientów, na które sztuczna inteligencja nie potrafiła odpowiedzieć. Sprzedawcy powinni regularnie przeglądać ten dziennik i tworzyć nowe FAQ, aby uzupełnić te luki w wiedzy.41
* **Testowanie bazy wiedzy:** Aplikacja zawiera funkcję testową, która umożliwia sprzedawcom przeszukiwanie własnej bazy wiedzy w celu sprawdzenia, jakie informacje sztuczna inteligencja pobierze w odpowiedzi na dane pytanie.41

Architektura chatbota powinna również uwzględniać solidną strategię awaryjną. Gdy narzędzie nie zwróci odpowiedzi lub odpowiedź będzie niezadowalająca, agent powinien być zaprogramowany tak, aby płynnie eskalował rozmowę, na przykład proponując połączenie użytkownika z konsultantem lub udostępniając link do formularza kontaktowego.

### **4.4 Uwierzytelnione działania na kontach klientów MCP**

For personalized, post-login interactions, the application must switch from using the public Storefront MCP to the authenticated **Customer Accounts MCP server**. This transition is triggered when a user makes a request that requires access to their private data, such as "Where is my last order?" or "I need to start a return."

As detailed in Part II, this requires the user to complete a full OAuth 2.0 flow to grant the application consent.9 Once the application has obtained a valid access token for the customer, the AI agent gains access to a new set of powerful, user-specific tools.

Key tools available through this server include:

* **Order Status and History:** The agent can retrieve a list of a customer's past orders and check the fulfillment status of a specific order by its number.
* **Return Management:** The agent can guide a user through initiating a return request for eligible items from a past order.
* **Account Management:** The agent can assist with retrieving or updating profile information, such as saved shipping addresses.

When implementing these tools, security and privacy are paramount. The application must handle the customer's OAuth token securely, storing it in an encrypted session and using it only for the duration required. Furthermore, the application should adhere to the principle of least privilege, only requesting the minimum API scopes necessary to perform its functions, thereby respecting user data and building trust. All tool calls must consistently handle potential errors, such as a resource not being found (e.g., "Order not found with number: #12345") or processing failures, and communicate these issues clearly to the user.9

## Part V: Advanced Concepts and Production Readiness

Transitioning a Shopify MCP chatbot from a functional prototype to a production-grade, scalable, and resilient service requires addressing a series of advanced technical challenges and strategic decisions. This final part covers the critical non-functional requirements that ensure the application is robust, efficient, and economically viable.

### 5.1 Strategic Search: Native search\_shop\_catalog vs. Custom RAG

One of the most significant architectural decisions is how to implement product search. While the native search\_shop\_catalog tool is powerful and easy to integrate, it may not be sufficient for all e-commerce scenarios.8 For stores with highly complex or nuanced catalogs, a custom

**Retrieval-Augmented Generation (RAG)** pipeline may be necessary to achieve the desired level of search accuracy and relevance.44 The decision to "build" a custom RAG versus "buy" the native functionality involves a series of trade-offs.

* **When to Use the Native search\_shop\_catalog Tool:** This approach is ideal for the majority of Shopify stores, particularly those with well-structured product data, standard product categories, and where conventional semantic search capabilities are adequate. The primary advantages are speed of implementation, lower cost, and zero infrastructure overhead, as the entire search mechanism is managed by Shopify.8
* **When to Build a Custom RAG Pipeline:** A custom RAG solution becomes compelling in specific, high-value scenarios:
  + **Complex or Niche Catalogs:** For stores selling products with intricate specifications, compatibility requirements, or subjective attributes (e.g., fashion, industrial parts, artisanal goods), a custom RAG can be trained on domain-specific knowledge to understand user queries with far greater nuance.44
  + **Unstructured Data Sources:** If valuable product information is locked within unstructured content like blog posts, buying guides, or PDF manuals, a RAG pipeline can index this content and make it searchable, something the native tool cannot do.47
  + **Full Control and Customization:** A custom RAG provides complete control over the entire retrieval process, including the choice of embedding models, chunking strategies, and re-ranking algorithms, allowing for fine-tuning that can significantly boost retrieval accuracy and relevance.44

Building a custom RAG pipeline introduces significant architectural complexity and cost. It requires additional infrastructure, including a vector database (like Pinecone, Milvus, or Astra DB), an embedding model (from providers like OpenAI or Hugging Face), and a robust data ingestion pipeline that uses Shopify webhooks to keep the vector index synchronized with any changes to the product catalog.48

| **Table 4: Decision Matrix: Native Search vs. Custom RAG** |  |  |
| --- | --- | --- |
| **Factor** | **Native search\_shop\_catalog Tool** | **Custom RAG Pipeline** |
| **Implementation Complexity** | Low | High |
| **Infrastructure Cost** | None (Included with Shopify) | High (Vector DB, Embedding API, Compute) |
| **Data Control & Privacy** | Data remains within Shopify ecosystem | Full control; can be self-hosted for maximum privacy |
| **Retrieval Accuracy** | Good for structured data | Potentially superior, especially for niche domains |
| **Suitability for Niche Catalogs** | Moderate | High |
| **Time to Market** | Fast | Slow |

### 5.2 Deployment and Scalability Strategies

Choosing the right hosting platform is a critical decision that impacts cost, scalability, performance, and developer experience. The Node.js backend of the Shopify app can be deployed to several modern cloud platforms, each with distinct advantages.

* **Fly.io:** This platform is an excellent choice for applications that benefit from long-running processes or require persistent state. It deploys applications as lightweight virtual machines ("Fly Machines") that can be scaled globally.51 Fly.io offers built-in support for deploying Shopify Remix applications, automatically parsing environment variables and simplifying the configuration process.13 Its ability to easily attach persistent volumes makes it a strong candidate for applications that use a SQLite database or need to persist other data on disk.13
* **Vercel:** As the creator of Next.js, Vercel provides a highly optimized, serverless platform for JavaScript and TypeScript applications. Its primary compute model is serverless functions, which are cost-effective for handling the bursty, unpredictable traffic patterns typical of a chatbot.54 Vercel offers templates for deploying both headless Shopify storefronts and generic MCP servers, and its tight integration with Git makes for a seamless CI/CD workflow.54
* **Cloudflare Workers:** This platform runs code on a global edge network, offering extremely low latency by executing functions close to the end-user. Cloudflare has invested heavily in the MCP ecosystem, providing advanced tooling for deploying remote MCP servers, including a workers-oauth-provider library that simplifies the implementation of the complex OAuth 2.0 flows required for secure servers.57 This makes it a compelling choice for performance-critical applications.

The choice of platform is influenced by the application's architecture. A simple, stateless orchestration layer might be most cost-effective on Vercel or Cloudflare Workers. In contrast, an application with a complex custom RAG pipeline, requiring a persistent vector database connection and potentially long-running indexing jobs, would be a much better fit for the VM-based model of Fly.io.

| **Table 3: Deployment Platform Comparison** |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Compute Model** | **State Management** | **Ease of Shopify Remix Deployment** |
| **Fly.io** | Virtual Machines (VMs) | Excellent (Persistent volumes, LiteStream for SQLite) | High (Built-in support via flyctl) |
| **Vercel** | Serverless Functions | Limited (Requires external database for persistence) | Moderate (Requires headless configuration) |
| **Cloudflare Workers** | Serverless Functions (Edge) | Limited (Requires external DB, KV store for state) | Moderate (Requires manual setup) |

### 5.3 Production Resilience: Rate Limiting and Error Handling

A production application must be architected to be resilient in the face of API failures and limitations. Shopify's APIs are strictly rate-limited to ensure platform stability, and failure to respect these limits will result in a degraded or non-functional application.58

* **Understanding Shopify's Rate Limits:** Shopify primarily uses two rate-limiting mechanisms:
  1. **Algorytm Leaky Bucket:** Dotyczy interfejsu API REST Admin i niektórych innych punktów końcowych. Każda aplikacja ma „zasobnik” o stałej pojemności. Wywołania API dodają „koszt” do zasobnika, a pojemność jest przywracana ze stałą szybkością. Jeśli seria żądań zapełni zasobnik, kolejne żądania będą ograniczane z 429 Too Many Requestsbłędem do momentu przywrócenia pojemności.58
  2. **Obliczony koszt zapytania:** Dotyczy interfejsu API administratora GraphQL. Każde pole w zapytaniu ma przypisany koszt punktowy. Całkowity koszt zapytania jest obliczany przed wykonaniem, a aplikacja ma budżet punktowy na minutę, oparty na planie Shopify sklepu. Jeśli żądany koszt zapytania przekroczy dostępną liczbę punktów, zostanie ono odrzucone.59
* **Strategie zarządzania limitami przepustowości:** Solidna aplikacja musi wdrażać wielotorową strategię, aby uniknąć ograniczeń:
  + **Optymalizuj zapytania:** Żądaj tylko tych danych, które są absolutnie niezbędne. Używaj konkretnych pól w zapytaniach GraphQL zamiast pobierać całe obiekty.58
  + **Inteligentne buforowanie:** buforuj często używane, nieulotne dane (np. zasady sklepu, szczegóły produktów), aby ograniczyć liczbę zbędnych wywołań API.
  + **Kolejkowanie i ograniczanie żądań:** Wdróż kolejkę żądań po stronie klienta, aby rozłożyć w czasie wywołania API i zapobiec nagłym wzrostom obciążenia, które mogłyby zapełnić dziurawy obszar.58
  + **Wykładniczy backoff:** Po 429wystąpieniu błędu aplikacja musi przerwać wysyłanie żądań i spróbować ponownie dopiero po pewnym czasie. Zalecanym podejściem jest strategia wykładniczego backoffu, w której opóźnienie zwiększa się po każdym kolejnym błędzie.58
* **Kompleksowa obsługa błędów:** Oprócz limitów przepustowości aplikacja musi płynnie obsługiwać szereg innych potencjalnych błędów, w tym 401 Unauthorized(co powinno uruchomić przepływ OAuth dla MCP kont klientów) 404 Not Found(gdy żądany zasób nie istnieje) oraz 5xxwewnętrzne błędy serwera z Shopify.9Agent powinien być w stanie zinterpretować te błędy i przekazać użytkownikowi jasną i pomocną informację zwrotną, zamiast po prostu zawieść.

Decyzja o wdrożeniu zaawansowanych funkcji, takich jak niestandardowy potok RAG, bezpośrednio wpływa na te kwestie operacyjne. System RAG z natury wymaga większej liczby wywołań API – do osadzania modeli, baz danych wektorowych i LLM do syntezy – co zwiększa potrzebę skrupulatnego ograniczania przepustowości, monitorowania kosztów i odpornej obsługi błędów w wielu usługach zewnętrznych.

### **5.4 Modele monetyzacji dla aplikacji partnerskich**

Chociaż jest to raport techniczny, wybór architektury jest nierozerwalnie związany z modelem biznesowym. Zmienne koszty związane z wywołaniami API LLM muszą zostać uwzględnione w strategii cenowej aplikacji. Kilka modeli jest realnych dla aplikacji partnerskiej Shopify tego typu:

* **Poziomy subskrypcji:** To najpopularniejszy model w sklepie z aplikacjami Shopify. Różne poziomy subskrypcji mogą być oferowane w zależności od limitów użytkowania (np. 500 rozmów opartych na sztucznej inteligencji miesięcznie w planie podstawowym, 5000 w planie profesjonalnym) lub dostępu do zaawansowanych funkcji (np. niestandardowe funkcje RAG lub szczegółowe analizy dostępne tylko w wyższych poziomach).62
* **Rozliczanie oparte na zużyciu:** Model „płać za rzeczywiste wykorzystanie”, w którym sprzedawcy są rozliczani na podstawie liczby interakcji ze sztuczną inteligencją lub wykorzystanych wywołań API. Model ten dostosowuje koszty bezpośrednio do wartości, ale może być mniej przewidywalny dla sprzedawców, co może stanowić barierę dla wdrożenia.64Dobrym kompromisem jest często podejście hybrydowe, w którym podstawowa subskrypcja obejmuje określony limit wykorzystania i opłaty za jego przekroczenie.
* **Cennik oparty na wartości:** Ten model stara się przechwycić część wartości generowanej przez aplikację, na przykład poprzez pobieranie niewielkiej prowizji od sprzedaży bezpośrednio przypisanej do asystenta AI. Choć teoretycznie jest to atrakcyjne, często jest trudne do wdrożenia ze względu na wyzwania związane z precyzyjną atrybucją i wrażliwość sprzedawców na spadek marży.64
* **Freemium lub bezpłatne wersje próbne:** Oferowanie ograniczonego bezpłatnego planu lub 14-30-dniowego bezpłatnego okresu próbnego to kluczowa strategia zmniejszania trudności związanych z adopcją. Pozwala to sprzedawcom przekonać się o wartości asystenta AI we własnym sklepie, zanim zdecydują się na płatny plan.63

Ostatecznie, skuteczny chatbot oparty na platformie MCP wymaga architektury, która jest nie tylko funkcjonalnie poprawna i odporna technicznie, ale także ekonomicznie opłacalna. Decyzje techniczne dotyczące implementacji wyszukiwania, platform wdrożeniowych i odporności operacyjnej muszą być podejmowane z jasnym zrozumieniem ich wpływu na całkowity koszt posiadania (TCO) oraz zdolność aplikacji do generowania trwałej wartości zarówno dla dewelopera, jak i sprzedawcy.

## **Wniosek**

Protokół Model Context Protocol stanowi przełomowy moment dla rozwoju e-commerce na platformie Shopify. Zasadniczo redefiniuje on relację między systemami sztucznej inteligencji (AI) a danymi handlowymi, zastępując kruchy i kosztowny paradygmat dedykowanych integracji API ujednoliconym, bezpiecznym i wysoce kompozycyjnym protokołem. Dla partnerów i deweloperów Shopify, MCP to nie tylko nowe narzędzie, ale zupełnie nowy wzorzec architektoniczny do tworzenia inteligentnych, konwersacyjnych doświadczeń.

W raporcie szczegółowo opisano kompleksowy cykl życia tworzenia chatbota Shopify AI dla klienta MCP. Kluczowe wnioski dla liderów technicznych i architektów to:

1. **Zaakceptuj wzorzec architektoniczny:** Skuteczna implementacja wymaga zrozumienia MCP nie tylko jako protokołu, ale także architektury klient-host-serwer z podstawowymi zasadami kompozycyjności i bezpieczeństwa przez izolację. Ekosystem Shopify, składający się z odrębnych serwerów Storefront, kont klientów i serwerów Dev MCP, stanowi bezpośrednie zastosowanie tego wzorca, umożliwiając specjalistyczne funkcjonalności z odpowiednimi modelami bezpieczeństwa.
2. **Priorytetowe traktowanie zarządzania stanem:** Główne wyzwanie inżynieryjne polega na zarządzaniu stanem w trzech domenach: kontekście konwersacyjnym LLM, danych w czasie rzeczywistym ze sklepu Shopify oraz interaktywnym interfejsie użytkownika. Połączenie platformy MCP do obsługi działań na danych i opartego na intencjach systemu interakcji użytkownika w interfejsie MCP zapewnia potężne ramy do centralizacji zarządzania stanem w ramach agenta AI, co powinno być głównym celem projektu zaplecza.
3. **Wykorzystaj frameworki orkiestracji:** Budowanie logiki wnioskowania, obsługi narzędzi i zarządzania stanem od podstaw jest nieefektywne. Frameworki takie jak LangChain i LlamaIndex, z dedykowanymi adapterami MCP, zapewniają niezbędne abstrakcje, które przyspieszają rozwój i pozwalają zespołom skupić się na tworzeniu zaawansowanych, wieloetapowych przepływów pracy agentów, zamiast na implementacji protokołów niskiego poziomu.
4. **Podejmuj strategiczne decyzje „Buduj czy kupuj”:** Platforma oferuje potężne, natywne narzędzia, takie jak search\_shop\_catalog. Jednak dla firm o złożonych potrzebach mogą one okazać się niewystarczające. Decyzja o wdrożeniu niestandardowego systemu RAG jest decyzją o znaczącym wpływie na infrastrukturę, koszty i złożoność. Wybór ten musi być podjęty świadomie, w oparciu o jasną analizę kompromisów między kontrolą, dokładnością i czasem wprowadzenia produktu na rynek.
5. **Inżynier ds. odporności produkcyjnej:** Aplikacja klasy produkcyjnej nie może ignorować wymagań niefunkcjonalnych. Proaktywne i solidne strategie radzenia sobie z limitami wydajności API Shopify, wdrażanie kompleksowej obsługi błędów oraz wybór skalowalnej architektury wdrożeniowej to nie tylko przemyślenia, ale kluczowe elementy początkowego projektu. Te czynniki operacyjne ostatecznie zadecydują o niezawodności, wydajności i długoterminowej żywotności aplikacji.

Przyszłość e-commerce w coraz większym stopniu opiera się na konwersacji. Dzięki opanowaniu protokołu Model Context Protocol i wzorców architektonicznych, które on umożliwia, programiści są teraz gotowi wyjść poza proste chatboty i stworzyć nową generację prawdziwie inteligentnych, interaktywnych i transakcyjnych asystentów zakupowych.

#### Cytowane prace

1. Shopify Model Context Protocol (MCP) for AI Ecommerce - Passionfruit SEO, otwierano: września 16, 2025, <https://www.getpassionfruit.com/blog/shopify-model-context-protocol-(mcp)-explained-the-ai-ecommerce-revolution>
2. Shopify Storefront MCP: How AI Shopping Assistants Convert More Customers, otwierano: września 16, 2025, <https://ecommerce.folio3.com/blog/shopify-storefront-mcp-how-ai-shopping-assistants-convert-more-customers/>
3. Shopify AI Storefront & MCP Explained for Developers - Arsturn, otwierano: września 16, 2025, <https://www.arsturn.com/blog/shopify-ai-storefront-mcp-explained-for-developers>
4. The Model Context Protocol (MCP) — A Complete Tutorial | by Dr. Nimrita Koul | Medium, otwierano: września 16, 2025, <https://medium.com/@nimritakoul01/the-model-context-protocol-mcp-a-complete-tutorial-a3abe8a7f4ef>
5. Model Context Protocol (MCP), and notes on implementing MCP server in Go, otwierano: września 16, 2025, <https://www.glukhov.org/post/2025/07/mcp-server-in-go/>
6. MCP JSON-RPC Server - Glama, otwierano: września 16, 2025, <https://glama.ai/mcp/servers/@melvincarvalho/mcpjs>
7. About Storefront MCP - Shopify developer documentation, otwierano: września 16, 2025, <https://shopify.dev/docs/apps/build/storefront-mcp>
8. What Are Shopify MCP Servers and Why They Matter for AI Commerce - Avenue Z, otwierano: września 16, 2025, <https://avenuez.com/blog/what-are-shopify-mcp-servers-ai-commerce/>
9. Customer Accounts MCP server - Shopify.dev, otwierano: września 16, 2025, <https://shopify.dev/docs/apps/build/storefront-mcp/servers/customer-account>
10. Shopify Dev MCP server, otwierano: września 16, 2025, <https://shopify.dev/docs/apps/build/devmcp>
11. How to Use Shopify Dev MCP Server - Apidog, otwierano: września 16, 2025, <https://apidog.com/blog/shopify-dev-mcp-server/>
12. Build a Storefront AI agent - Shopify developer documentation, otwierano: września 16, 2025, <https://shopify.dev/docs/apps/build/storefront-mcp/build-storefront-ai-agent>
13. Shopify · Fly Docs - Fly.io, otwierano: września 16, 2025, <https://fly.io/docs/js/shopify/>
14. siddhantbajaj/shopify-mcp-server - GitHub, otwierano: września 16, 2025, <https://github.com/siddhantbajaj/shopify-mcp-server>
15. ramakay/ShopifyMockMCP: A Shopify MCP Server built to interact with Mock.shop - GitHub, otwierano: września 16, 2025, <https://github.com/ramakay/ShopifyMockMCP>
16. MCP server for Shopify api, usable on mcp hosts such as Claude and Cursor - GitHub, otwierano: września 16, 2025, <https://github.com/GeLi2001/shopify-mcp>
17. MCP UI: Breaking the text wall with interactive components (2025 ..., otwierano: września 16, 2025, <https://shopify.engineering/mcp-ui-breaking-the-text-wall>
18. Custom apps - Shopify Help Center, otwierano: września 16, 2025, <https://help.shopify.com/en/manual/apps/app-types/custom-apps>
19. wolfielabs/shopify-storefront-mcp-server - NPM, otwierano: września 16, 2025, <https://www.npmjs.com/package/%40wolfielabs%2Fshopify-storefront-mcp-server>
20. QuentinCody/shopify-storefront-mcp-server - GitHub, otwierano: września 16, 2025, <https://github.com/QuentinCody/shopify-storefront-mcp-server>
21. Supercharging E-commerce with Shopify Integration, LangChain Tools, and Function Calling - Expertise AI, otwierano: września 16, 2025, <https://www.expertise.ai/blog/shopify-integration-langchain-function-calling>
22. Challenges of building an AI Agent on top of Shopify MCP | Quickchat AI, otwierano: września 16, 2025, <https://quickchat.ai/post/challenges-building-ai-agent-shopify-mcp>
23. LangChain and MCP: Building Enterprise AI Workflows with ..., otwierano: września 16, 2025, <https://medium.com/@richardhightower/langchain-and-mcp-building-enterprise-ai-workflows-with-universal-tool-integration-e0547742233f>
24. How To Connect AI Voice Agent to Shopify Using MCP Server (2025) - YouTube, otwierano: września 16, 2025, <https://www.youtube.com/watch?v=Z2H8NcVBEZc>
25. Guide to Using the Responses API's MCP Tool - OpenAI Cookbook, otwierano: września 16, 2025, <https://cookbook.openai.com/examples/mcp/mcp_tool_guide>
26. MCP Adapters for LangChain and LangGraph, otwierano: września 16, 2025, <https://changelog.langchain.com/announcements/mcp-adapters-for-langchain-and-langgraph>
27. LlamaIndex + MCP Usage - LlamaIndex, otwierano: września 16, 2025, <https://docs.llamaindex.ai/en/stable/examples/tools/mcp/>
28. Build LLamaIndex Agents with MCP Connector | by Pedro Azevedo - Medium, otwierano: września 16, 2025, <https://medium.com/@pedroazevedo6/build-llamaindex-agents-with-mcp-connector-69df32d95508>
29. Mcp - LlamaIndex, otwierano: września 16, 2025, <https://docs.llamaindex.ai/en/stable/api_reference/tools/mcp/>
30. llama-index-tools-shopify - PyPI, otwierano: września 16, 2025, <https://pypi.org/project/llama-index-tools-shopify/>
31. Shopify - LlamaIndex, otwierano: września 16, 2025, <https://docs.llamaindex.ai/en/logan-llama_deploy_docs/api_reference/tools/shopify/>
32. Test and customize your agent - Shopify.dev, otwierano: września 16, 2025, <https://shopify.dev/docs/apps/build/storefront-mcp/testing-and-examples>
33. Get Started Using an Order Status Template With a Shopify Integration - Kore.ai, otwierano: września 16, 2025, <https://www.kore.ai/blog/get-started-using-an-order-status-template-with-a-shopify-integration>
34. Consuming Streamed LLM Responses on the Frontend: A Deep ..., otwierano: września 16, 2025, <https://tpiros.dev/blog/streaming-llm-responses-a-deep-dive/>
35. Streaming LLM Responses — Tutorial For Dummies (Using PocketFlow!) - Medium, otwierano: września 16, 2025, <https://medium.com/@zh2408/streaming-llm-responses-tutorial-for-dummies-using-pocketflow-417ad920c102>
36. What Is LLM Optimization? A Guide for Shopify Merchants - Sufio, otwierano: września 16, 2025, <https://sufio.com/blog/llm-optimization-shopify-guide/>
37. Implementing AI-Powered Search: A Step-by-Step Guide for E-commerce Platforms, otwierano: września 16, 2025, <https://wizzy.ai/blog/implementing-ai-powered-search-a-step-by-step-guide-for-e-commerce-platforms/>
38. 10 Techniques to Make Your Shopify Store Content for AI And LLMs - StoreSEO, otwierano: września 16, 2025, <https://storeseo.com/blog/make-your-shopify-store-content-for-ai-and-llms/>
39. Leveraging Multimodal LLMs for Shopify's Global Catalogue: Recap of Expo Talk at ICLR 2025, otwierano: września 16, 2025, <https://shopify.engineering/leveraging-multimodal-llms>
40. Guide to Using the Responses API's MCP Tool - AiDocZh, otwierano: września 16, 2025, <https://www.aidoczh.com/openai-cookbook/examples/mcp/mcp_tool_guide.html>
41. Managing your store FAQs using the Shopify ... - Shopify Help Center, otwierano: września 16, 2025, <https://help.shopify.com/en/manual/promoting-marketing/knowledge-base/managing-faqs>
42. Shopify Knowledge Base, otwierano: września 16, 2025, <https://help.shopify.com/en/manual/promoting-marketing/knowledge-base>
43. Shopify Knowledge Base - Customize FAQs used by AI agents to answer shopper questions, otwierano: września 16, 2025, <https://apps.shopify.com/shopify-knowledge-base>
44. Custom RAG vs. AI Platforms: Choice Guide | Opentrends, otwierano: września 16, 2025, <https://www.opentrends.us/en/custom-rag-vs-ai-platform-choice-guide>
45. Build or Buy RAG? Four Questions to Guide Your Approach to Retrieval Augmented Generation for GenAI - Datalere, otwierano: września 16, 2025, <https://datalere.com/articles/build-or-buy-rag-four-questions-to-guide-your-approach-to-retrieval-augmented-generation-for-genai>
46. AI Agent Chatbots vs RAG Bots | Customer Service & eCommerce - Gleen AI, otwierano: września 16, 2025, <https://alhena.ai/blog/rag-agentic-ai-in-customer-support/>
47. When has it made sense to build a custom RAG solution over using RAG as a service? : r/LangChain - Reddit, otwierano: września 16, 2025, <https://www.reddit.com/r/LangChain/comments/1auviyf/when_has_it_made_sense_to_build_a_custom_rag/>
48. The Exact Steps to Implement Custom Ai Shopify Chatbots for Customers | Width.ai, otwierano: września 16, 2025, <https://www.width.ai/post/shopify-chatbots-for-customers>
49. OpenAI RAG vs. Your Customized RAG: Which One Is Better? | by Zilliz - Medium, otwierano: września 16, 2025, <https://medium.com/@zilliz_learn/openai-rag-vs-your-customized-rag-which-one-is-better-4c65a7c6317b>
50. Enhancing Retail Product Search with RAG - YouTube, otwierano: września 16, 2025, <https://www.youtube.com/watch?v=Hskf9g9LZaE>
51. Rebel Alliance Headquarters for your Shopify Apps - Fly.io, otwierano: września 16, 2025, <https://fly.io/shopify>
52. Deploy app servers close to your users · Fly, otwierano: września 16, 2025, <https://fly.io/>
53. fly launch support for Shopify Remix applications - Fresh Produce - Fly.io Community, otwierano: września 16, 2025, <https://community.fly.io/t/fly-launch-support-for-shopify-remix-applications/23455>
54. Deploy MCP servers to Vercel, otwierano: września 16, 2025, <https://vercel.com/docs/mcp/deploy-mcp-servers-to-vercel>
55. Vercel and Shopify Integration, otwierano: września 16, 2025, <https://vercel.com/docs/integrations/ecommerce/shopify>
56. MCP with Vercel Functions, otwierano: września 16, 2025, <https://vercel.com/templates/other/model-context-protocol-mcp-with-vercel-functions>
57. Build and deploy Remote Model Context Protocol (MCP) servers to ..., otwierano: września 16, 2025, <https://blog.cloudflare.com/remote-model-context-protocol-servers-mcp/>
58. A Developer's Guide to Managing Rate Limits for Shopify's API and GraphQL - Lunar.dev, otwierano: września 16, 2025, <https://www.lunar.dev/post/a-developers-guide-managing-rate-limits-for-the-shopify-api-and-graphql>
59. Shopify API limits - Shopify.dev, otwierano: września 16, 2025, <https://shopify.dev/docs/api/usage/limits>
60. Frustrated With Shopify Integration Throttling? You Need a Cloud iPaaS Solution, otwierano: września 16, 2025, <https://www.appseconnect.com/how-to-handle-shopify-api-rate-limits-with-erp/>
61. API Error Handling in Shopify - Best Practices Every Developer Should Follow - MoldStud, otwierano: września 16, 2025, <https://moldstud.com/articles/p-api-error-handling-in-shopify-best-practices-every-developer-should-follow>
62. Helpdesk, Chat & FAQ - Gorgias | AI Customer support App for Shopify stores, otwierano: września 16, 2025, <https://apps.shopify.com/helpdesk>
63. Tidio ‑ Live Chat & AI Chatbot - Free Trial - Shopify App Store, otwierano: września 16, 2025, <https://apps.shopify.com/tidio-chat>
64. Building a Startup on Shopify's Model Context Protocol (MCP) - FRANKI T, otwierano: września 16, 2025, <https://www.francescatabor.com/articles/2025/7/13/shopify-mcp>
65. Shopping Agent by Quickchat AI - Shopify App Store, otwierano: września 16, 2025, <https://apps.shopify.com/quickchat-ai>