TalkNShop

Catalog Service and MCP Integration

Design Specification

Version 0.1

Rutuja Nemane
(rutujabhagawat.nemane@sjsu.edu)

Version History

Version	Changes
0.1	Design details of Catalog service and current implementation status

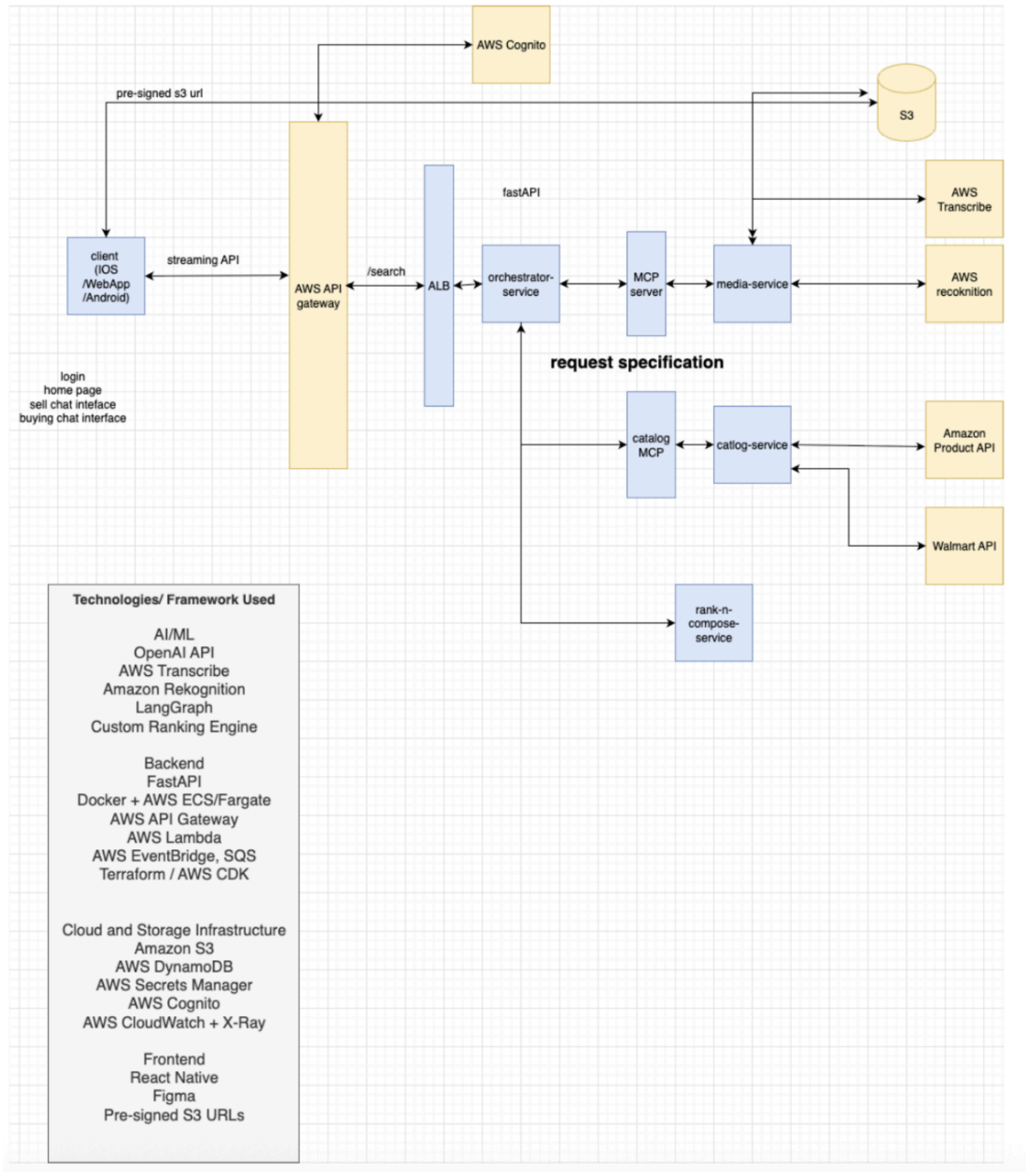
Introduction

This document details the design specification for the **Catalog Service and MCP Integration**, a key backend component of the TalkNShop platform. This service is responsible for retrieving, normalizing, and returning product data from external e-commerce APIs including Amazon (via RapidAPI and Kroger. It plays a critical role in enabling personalized, real-time product recommendations across marketplaces and forms the core search backend used by the orchestrator. The catalog service is accessed by the Orchestrator Service through an MCP (Model Context Protocol) server, which provides a standardized interface for the orchestrator to invoke catalog search functions as tools.

References

RapidAPI documentation for Product Search API
AWS Bedrock & API Gateway architecture
Kroger API Documentation (OAuth + Product Search): Link
Workbook: Link
TalkNShop System Architecture (AWS API Gateway, ALB, Orchestrator-Service integration)

Architecture Diagram



System architecture. I own the catalog service and MCP integration workflow.

Requirements

Functional Requirements

- Accept structured SearchRequest JSON (query, filters, pagination, sort) from orchestrator or UI
- Translate requests into API calls to Amazon (RapidAPI) and Kroger
- Normalize responses into the unified Product schema
- Support filtering (price range, platforms), sorting (price/rating), pagination
- Return SearchResponse JSON to orchestrator/UI
- Expose product details and reviews for Amazon ASINs
- Surfacing availability, rating, price, image, description, and URLs

Non-Functional Requirements

- Response time P95 < 2s for blended search (single platform < 1.5s)
- Graceful fallback when a provider fails (partial results with logging)
- Secure credential management via environment variables/.env
- Retry logic for transient errors (timeouts, 429s) inside API clients
- Structured logging with request correlation IDs (planned)
- Future: caching, rate limiting, automated health checks

Functional Overview

MCP Integration

The Catalog MCP Server acts as an adapter layer between the Orchestrator Service and the Catalog Service:

- Purpose: Allows the orchestrator to invoke catalog functions as MCP tools
- Implementation: MCP server exposes catalog search functions that the orchestrator can call
- Benefits:
    - Standardized interface for orchestrator-to-catalog communication
    - Tool-based invocation model (search_products, get_product_details, etc.)
    - Easier integration with orchestrator's tool selection logic

MCP Tools Exposed (Planned):

- search_products - Search products across platforms
- get_product_details - Get detailed product information
- get_product_reviews - Get product reviews

FastAPI-based microservice exposing:

- GET / – Web interface (HTML/JS demo) for manual searches
- GET /health – Health probe for orchestrator and load balancer
- POST /api/v1/search – Core search endpoint accepting SearchRequest

- GET /api/v1/products/{product_id} – Detailed Amazon product info (ASIN)
- GET /api/v1/products/{product_id}/reviews – Amazon product reviews

Internal flow:

1. Validate SearchRequest (query, filters, pagination, sort_by)
2. For each requested platform:
    - RapidAPI Amazon: search endpoint with query, page, sort mapping
    - Kroger: OAuth token acquisition, optional zip_code → location_id, product search with pagination
3. Transform responses into unified Product model
4. Apply filter + sort + pagination (server-side)
5. Return SearchResponse with metadata (query, total_results, page, size, timestamp, products)

Configuration/ External Interfaces

Environment Variables (config.env)

- RAPIDAPI_KEY=<RapidAPI real-time Amazon data key>
- KROGER_CLIENT_ID=<Kroger OAuth client id>
- KROGER_CLIENT_SECRET=<Kroger OAuth client secret>
- KROGER_ZIP_CODE=95112 # Default zip for location-aware pricing
- DEBUG=false

External APIs

- RapidAPI real-time-amazon-data
    - /search
    - /product-details
    - /product-reviews
- Kroger Product API
    - /v1/connect/oauth2/token
    - /v1/products
    - /v1/locations?filter.zipCode=<zip> (for location_id)

Internal Interfaces

- Orchestrator → Catalog: POST /api/v1/search with SearchRequest
- Catalog → Orchestrator/UI: SearchResponse (list of Product)
- Planned queue/topic for async price refreshes (future)

Debug

Debug endpoints (Planned)

- GET /api/v1/debug/amazon – Raw Amazon payloads for given query
- GET /api/v1/debug/kroger
- GET /api/v1/debug/kroger/locations

Logging

- INFO: request received, platform call start/end, counts per platform
- ERROR: API failures with details (status code, message)
- Planned: include request IDs and structured JSON logs

Sample logs:

```
INFO: Starting product search for platform: amazon
ERROR: Amazon API failed: TimeoutError
```

Counters

The catalog service tracks the following metrics for monitoring and performance analysis:

Request Metrics

Total Requests: Total number of search requests received
Requests Per Second (RPS): Average and peak requests per second
Target: Handle up to 50 RPS sustained, 100 RPS peak
Successful Requests: Count of successful search operations
Failed Requests: Count of failed search operations (by error type)
Request Rate by Platform: Requests per second breakdown for Amazon vs Kroger searches

Performance Metrics (Latency)

API Latency Percentiles: Response time distribution for search endpoint
P50 (Median): 50th percentile response time (target: < 1s)
P95: 95th percentile response time (target: < 2s)
P99: 99th percentile response time (target: < 3s)
Platform-Specific Latency:
Amazon API latency (RapidAPI response time) - P95: ~800ms
Kroger API latency (including OAuth) - P95: ~1200ms
End-to-End Latency: Total time from request received to response sent
Transformation Latency: Time spent in data normalization and mapping

Product Metrics

Products Fetched Per Platform:
- Total products from Amazon
- Total products from Kroger
- Average products per search query
- Products Returned: Number of products in final response (after filtering/sorting)
- Products Filtered Out: Count of products filtered by price/brand/category criteria

Implementation

Design Notes

- FastAPI with async endpoints, uvicorn for dev server
- aiohttp for non-blocking external HTTP calls
- RapidAPIAmazonClient handles search/details/reviews + transformation
- KrogerAPIClient handles OAuth, location lookup, search, transformation
- Unified Product Schema (models/search.py):
    - id (prefixed e.g., amazon_B07ZPKBL9V, kroger_0001111040101)
    - title, brand, price, currency, platform, platform_id
    - rating, review_count, image_url, availability
    - description, url, shipping_info
- Filters: price_min, price_max, brands, categories, platforms
- Sorting: price_asc, price_desc, rating_desc, fallback to relevance
- Pagination: page/size limiting to 100 items per request
- CORS enabled for demo UI
- Planned: caching (Redis), circuit breaker, pooling, metrics exporter

Subtasks

- Amazon API Client (Completed)
- Kroger API Client with OAuth token manager (Completed)
- Unified Product Schema Model (Completed)
- Endpoint implementation for /search (Completed)
- Debug endpoints (In Progress)
- Redis caching layer (Planned in Sprint 10)
- Rate limiting and retry wrappers (Planned in Sprint 10)

Testing

General Approach

- Manual Postman tests hitting /api/v1/search with real queries
- Manual UI testing via GET /
- Verification of schema compliance, error handling, multi-platform search
- Controlled failures: invalid API keys, network timeouts, OAuth errors
- Logging inspection for debugging Kroger price/link issues

Unit Tests

- Search flow (Amazon only, Kroger only, combined)
- Price filtering, sorting via manual payload inspection
- Kroger zip/location troubleshooting (95112, 95110)
- Error scenarios: missing API keys, invalid OAuth scope, token refresh failure

Examples:

```
@pytest.mark.asyncio

async def test_search_products_success():

   """Test successful product search"""

   client = RapidAPIAmazonClient(api_key="test_key")

   # Mock API response

   results = await client.search_products(query="headphones", page=1)

   assert len(results) == 1

   assert results[0].title == "Test Headphones"

   assert results[0].price == 99.99

   assert results[0].platform == "amazon"
```

```
def test_search_endpoint_success():

   """Test search endpoint with valid request"""
```

```
response = client.post(

    "/api/v1/search",

    json={

        "query": "headphones",

        "filters": {"platforms": ["amazon"], "price_min": 50.0},

        "pagination": {"page": 1, "size": 10}

    }

)

assert response.status_code == 200

assert "products" in response.json()

assert len(response.json()["products"]) <= 10
```

Integration Tests (Planned)

- End-to-end search via orchestrator contract
- Load testing for concurrency
- Latency benchmarking for Amazon/Kroger separately and combined
- Regression suite for future API client changes

Functional Tests

- Input: “headphones under $100”
    - Check result count > 0
    - Validate schema keys: title, price, rating, etc.
- Input: “invalid token”
    - Expect 401 error from Kroger

Appendix

Sample Response:

```
{
  "query": "headphones",
  "total_results": 150,
  "page": 1,
  "size": 20,
  "timestamp": "2025-01-15T10:30:00Z",
  "products": [
    {
      "id": "amazon_B0A1234567",
      "title": "Sony WH-CH520 Wireless Headphones",
      "brand": "Sony",
      "price": 89.99,
      "currency": "USD",
      "platform": "amazon",
      "platform_id": "B0A1234567",
      "rating": 4.5,
      "review_count": 1250,
      "image_url": "https://m.media-amazon.com/images/I/...",
      "availability": "in_stock",
      "description": "Up to 50 hours battery life ...",
      "url": "https://www.amazon.com/dp/B0A1234567"
    },
    {
      "id": "kroger_0001111040101",
      "title": "Kroger Vitamin D Whole Milk Gallon",
      "brand": "Kroger",
      "price": 3.89,
      "currency": "USD",
      "platform": "kroger",
      "platform_id": "0001111040101",
      "rating": 2.9,
      "review_count": 71,
      "image_url": "https://www.kroger.com/product/images/large/front/0001111040101",
      "availability": "in_store",
      "description": "Kroger® Vitamin D Whole Milk Gallon",
      "url": "https://www.kroger.com/p/kroger-vitamin-d-whole-milk-gallon/0001111040101"
    }
  ]
}
```

MCP Tool Schema Example

```
{
  "name": "search_products",
  "description": "Search for products across Amazon and Kroger platforms",
  "inputSchema": {
    "type": "object",
    "properties": {
      "query": {
        "type": "string",
        "description": "Search query string"
      },
      "filters": {
        "type": "object",
        "properties": {
          "price_min": {"type": "number"},
          "price_max": {"type": "number"},
          "platforms": {"type": "array", "items": {"type": "string"}}
        }
      },
      "pagination": {
        "type": "object",
        "properties": {
          "page": {"type": "integer"},
          "size": {"type": "integer"}
        }
      },
      "sort_by": {"type": "string"}
    },
    "required": ["query"]
  }
}
```