AquaRAG - Unified Schema + Reasoning Agent Plan

🎯 Unified Schema - Finalna Wersja

Struktura dla wszystkich typów treści:

json

```
{
  "id": "unique-identifier-001",
  "content_type": "product|knowledge|faq",
  "lang": "pl|en",
  "title": "SEO-friendly title z keywords",
  "query_text": "Zoptymalizowane keywords + synonimy + problemy",
  "full_content": "Kompletny content - wszystko co LLM potrzebuje",
  "domain": "seawater|freshwater|universal",
  "category": "aquascaping|parameters|bacteria|problems|education|equipment",
  "intent": ["maintenance", "troubleshooting", "startup", "learning"],
  "difficulty": "beginner|intermediate|expert",
  "url": "https://aquaforest.eu/pl/...",
  "tags": ["keyword1", "product-name", "chemical-compound"],
  "related_items": ["product-ids", "knowledge-ids", "faq-ids"],
  "updated_at": "2025-05-29"
}
```

📋 Przykłady dla każdego typu treści

PRODUCT

json


```
{
  "id": "af-gel-fix-001",
  "content_type": "product",
  "lang": "pl",
  "title": "AF Gel Fix – klej żelowy do koralowców",
  "query_text": "AF Gel Fix klej żelowy koralowce aquascaping mocowanie szczepek 10 sekund wiąz
  "full_content": "AF Gel Fix to szybkowiążący klej żelowy do precyzyjnego mocowania szczepek k
  "domain": "seawater",
  "category": "aquascaping",
  "intent": ["coral_mounting", "aquascaping", "repair"],
  "difficulty": "beginner",
  "url": "https://aquaforest.eu/pl/produkty/seawater/aquascaping/af-gel-fix/",
  "tags": ["klej", "10_sekund", "żel", "nietoksyczny", "precyzyjny", "koralowce"],
  "related_items": ["af-frag-rocks", "af-mini-rocks", "af-plug-rocks"],
  "updated_at": "2025-05-29"
}
```


FAQ

json


```
{
  "id": "faq-pl-001",
  "content_type": "faq",
  "lang": "pl",
  "title": "Jak obniżyć azotany (NO3) w akwarium rafowym?",
  "query_text": "azotany NO3 obniżyć zmniejszyć akwarium rafowe koralowce problem wysokie zanie
  "full_content": "PROBLEM: Wysokie azotany (NO3) to częsty problem w akwariach rafowych, powoc
  "domain": "seawater",
  "category": "problems",
  "intent": ["troubleshooting", "water_parameters", "maintenance"],
  "difficulty": "intermediate",
  "url": "https://aquaforest.eu/pl/faq/azotany-obnizenie",
  "tags": ["NO3", "azotany", "problemy", "NP_Pro", "Pro_Bio_S", "testy"],
  "related_items": ["af-np-pro", "pro-bio-s", "af-test-kit-no3"],
  "updated_at": "2025-05-29"
}
```


KNOWLEDGE

json


```
{
  "id": "knowledge-pl-001",
  "content_type": "knowledge",
  "lang": "pl",
  "title": "Kompletny przewodnik po cyklu azotowym w akwarium morskim",
  "query_text": "cykl azotowy akwarium morskie bakterie nitryfikacyjne NH3 NO2 NO3 startup dojr
  "full_content": "WPROWADZENIE: Cykl azotowy to fundament każdego zdrowego akwarium morskiego.
  "domain": "seawater",
  "category": "education",
  "intent": ["learning", "startup", "understanding"],
  "difficulty": "beginner",
  "url": "https://aquaforest.eu/pl/poradniki/cykl-azotowy",
  "tags": ["cykl_azotowy", "bakterie", "NH3", "NO2", "NO3", "startup", "Bio_S", "edukacja"],
  "related_items": ["bio-s", "af-bio-sand", "pro-bio-s", "af-test-kit-nh3-no2-no3"],
  "updated_at": "2025-05-29"
}
```

🧠 Agent RAG Reasoning - Kompletny Plan

Architektura Reasoning Agent

python

```python
class AquaRAGReasoningAgent:
    def __init__(self):
        self.max_reasoning_loops = 3
        self.confidence_threshold = 7.0
        self.pinecone_index = get_pinecone_index()
        self.llm = get_openai_client()

    def process_query(self, user_query: str, lang_hint: str = "pl"):
        # Step 1: Analyze user intent
        intent_analysis = self.analyze_user_intent(user_query)

        # Step 2: Initial search strategy
        search_strategy = self.determine_search_strategy(intent_analysis)

        # Step 3: Reasoning loop
        for attempt in range(self.max_reasoning_loops):
            results = self.execute_search(user_query, search_strategy, attempt)
            confidence = self.evaluate_results(user_query, results, intent_analysis)

            if confidence >= self.confidence_threshold:
                return self.generate_final_response(user_query, results, confidence)

            # Refine strategy for next attempt
            search_strategy = self.refine_search_strategy(
                user_query, results, confidence, attempt
            )

        # If all attempts failed, escalate to human support
        return self.escalate_to_support(user_query, intent_analysis)
```

Step 1: Intent Analysis

python

```python
def analyze_user_intent(self, user_query: str) -> dict:
    analysis_prompt = f"""
    Przeanalizuj pytanie użytkownika i określ:

    Pytanie: "{user_query}"

    Zwróć JSON z:
    1. primary_intent: learning|troubleshooting|product_info|purchase|maintenance
    2. domain_preference: seawater|freshwater|universal
    3. user_level: beginner|intermediate|expert
    4. urgency: low|medium|high
    5. expected_content_types: ["product", "knowledge", "faq"]
    6. key_concepts: [lista kluczowych pojęć]
    """

    return self.llm.analyze(analysis_prompt)

# Przykład output:
{
    "primary_intent": "troubleshooting",
    "domain_preference": "seawater",
    "user_level": "intermediate",
    "urgency": "high",
    "expected_content_types": ["knowledge", "faq", "product"],
    "key_concepts": ["azotany", "NO3", "wysokie", "obniżyć", "problem"]
}
```

Step 2: Search Strategy

python

```python
def determine_search_strategy(self, intent_analysis: dict) -> dict:
    strategy_prompt = f"""
    Na podstawie analizy intencji: {intent_analysis}

    Określ strategię wyszukiwania:
    1. primary_search_category: który content_type przeszukać najpierw
    2. fallback_categories: kolejność przeszukiwania innych kategorii
    3. filter_params: filtry do zastosowania
    4. search_keywords: zoptymalizowane słowa kluczowe
    """

    return self.llm.determine_strategy(strategy_prompt)

# Przykład strategii:
{
    "primary_search_category": "knowledge",
    "fallback_categories": ["faq", "product"],
    "filter_params": {
        "domain": "seawater",
        "category": "problems",
        "difficulty": ["beginner", "intermediate"]
    },
    "search_keywords": "azotany NO3 obniżyć wysokie problem akwarium rafowe"
}
```

Step 3: Adaptive Search Execution

python

```python
def execute_search(self, user_query: str, strategy: dict, attempt: int) -> list:
    # Attempt 0: Search primary category only
    if attempt == 0:
        results = self.pinecone_index.query(
            vector=self.embed(strategy["search_keywords"]),
            filter={
                "content_type": strategy["primary_search_category"],
                **strategy["filter_params"]
            },
            top_k=5
        )

    # Attempt 1: Expand to fallback categories
    elif attempt == 1:
        results = self.pinecone_index.query(
            vector=self.embed(strategy["search_keywords"]),
            filter={
                "content_type": {"$in": [
                    strategy["primary_search_category"],
                    strategy["fallback_categories"][0]
                ]},
                **strategy["filter_params"]
            },
            top_k=8
        )

    # Attempt 2: Broad search across all categories
    else:
        # Remove difficulty filter, expand domain if needed
        relaxed_filters = {k: v for k, v in strategy["filter_params"].items()
                           if k not in ["difficulty"]}

        results = self.pinecone_index.query(
            vector=self.embed(user_query),  # Use original query
            filter=relaxed_filters,
            top_k=10
        )

    return results
```

Step 4: Result Evaluation

python

```python
def evaluate_results(self, user_query: str, results: list, intent_analysis: dict) -> float:
    if not results:
        return 0.0

    evaluation_prompt = f"""
    ZADANIE: Oceń czy wyniki wyszukiwania odpowiadają na pytanie użytkownika.

    PYTANIE UŻYTKOWNIKA: "{user_query}"
    INTENCJA: {intent_analysis["primary_intent"]}
    POZIOM: {intent_analysis["user_level"]}

    NAJLEPSZY WYNIK:
    Tytuł: {results[0].metadata["title"]}
    Typ: {results[0].metadata["content_type"]}
    Treść: {results[0].metadata["full_content"][:500]}...

    OCEŃ na skali 1-10:
    1. Relevance (czy odpowiada na pytanie): _/10
    2. Completeness (czy informacje są kompletne): _/10
    3. Difficulty_match (czy pasuje do poziomu użytkownika): _/10
    4. Actionability (czy użytkownik wie co robić dalej): _/10

    ŚREDNIA: _/10

    UZASADNIENIE: [krótkie wyjaśnienie oceny]
    """

    evaluation = self.llm.evaluate(evaluation_prompt)
    return float(evaluation["średnia"])
```

Step 5: Strategy Refinement

python

```python
def refine_search_strategy(self, user_query: str, previous_results: list,
                           confidence: float, attempt: int) -> dict:
    refinement_prompt = f"""
    POPRZEDNIA PRÓBA #{attempt} dała confidence: {confidence}/10

    PYTANIE: "{user_query}"
    POPRZEDNIE WYNIKI: {[r.metadata["title"] for r in previous_results[:3]]}

    CO POPRAWIĆ w następnej próbie?
    1. Czy zmienić kategorię wyszukiwania?
    2. Czy rozszerzyć/zawęzić filtry?
    3. Czy zmienić słowa kluczowe?
    4. Czy szukać w innych domenach?

    ZWRÓĆ poprawioną strategię wyszukiwania.
    """

    return self.llm.refine_strategy(refinement_prompt)
```

Step 6: Final Response Generation

python

```python
def generate_final_response(self, user_query: str, results: list, confidence: float) -> dict:
    # Combine information from multiple results if needed
    combined_context = self.combine_results(results[:3])

    response_prompt = f"""
    KONTEKST z bazy wiedzy Aquaforest:
    {combined_context}

    PYTANIE UŻYTKOWNIKA: "{user_query}"

    ODPOWIEDZ:
    1. Bezpośrednio na pytanie użytkownika
    2. Używając informacji z kontekstu
    3. W języku polskim
    4. Konkretnie i praktycznie
    5. Załącz linki do produktów/artykułów jeśli relevantne

    Na końcu dodaj: "Więcej informacji: [URL z najlepszego wyniku]"
    """

    response = self.llm.generate(response_prompt)

    return {
        "answer": response,
        "confidence": confidence,
        "sources": [r.metadata["url"] for r in results[:3]],
        "reasoning_attempts": self.current_attempt + 1
    }
```

Step 7: Support Escalation

python

```python
def escalate_to_support(self, user_query: str, intent_analysis: dict) -> dict:
    escalation_info = {
        "message": "Przepraszam, nie mogę znaleźć odpowiedniej odpowiedzi na Twoje pytanie. Prz
        "support_form": True,
        "query_analysis": intent_analysis,
        "suggested_contact": self.determine_support_channel(intent_analysis)
    }

    # Log failed query for dataset improvement
    self.log_failed_query(user_query, intent_analysis)

    return escalation_info
```

🚀 Korzyści Reasoning Agent

1. Inteligentne wyszukiwanie

- Model sam decyduje gdzie szukać na podstawie pytania
- Adaptacyjne strategie: od wąskich do szerokich filtrów
- Cross-category search gdy potrzeba

2. Self-evaluation

- Model ocenia jakość własnych wyników
- Automatyczne refinement gdy confidence < 7/10
- Transparent scoring dla debugging

3. Graceful degradation

- 3 próby z różnymi strategiami
- Escalation do human support gdy AI nie wystarczy
- Logging failed queries dla dataset improvement

4. Business benefits

- Wyższa satysfakcja użytkowników (lepsze odpowiedzi)
- Mniej escalacji do support team
- Dane o gaps w knowledge base
- A/B testing różnych strategii reasoning

📊 Metryki i monitoring

KPIs do śledzenia:

- **Confidence score distribution** - ile % queries dostaje score >7
- **Reasoning attempts** - średnia liczba prób przed sukcesem
- **Escalation rate** - % queries które idą do human support
- **User satisfaction** - thumbs up/down feedback
- **Category distribution** - które content_types są najczęściej używane

Continuous improvement:

- Weekly analysis failed queries → new knowledge entries
- A/B test different confidence thresholds
- Optimize search strategies based on success rates
- Expand knowledge base w najbardziej problematycznych obszarach

---

🎯 Implementation Timeline

Week 1-2: Core Reasoning Engine

- Implement intent analysis
- Build search strategy logic
- Create evaluation system

Week 3-4: Adaptive Features

- Add strategy refinement
- Implement escalation flows
- Build monitoring dashboard

Week 5-6: Optimization

- A/B test różnych strategii
- Fine-tune confidence thresholds
- Performance optimization

Result: Najinteligentniejszy customer support chatbot w branży akwarystycznej! 🔥