Use Case UC2: Query Processing & Answer Prioritization

Scope: RAG Chatbot System (Backend Pipeline)

Level: Sub-function

Primary Actor: User (via System Trigger)

1. Stakeholders and Interests

- **User:** Wants the system to understand the context of their question to get a more accurate answer (Intent). Expects the final results to be highly relevant due to advanced scoring (Reranking).
- **Developer:** Wants to ensure the rule-based intent matching logic is accurate and the heuristic scoring formula provides tangible improvements over the base TF ranking.
- **Orchestrator (System Controller):** Expects to receive the final, highly-prioritized list of document chunks (`List<Hit>`) for final answer generation.

2. Preconditions

- The user's question (`question`) has been received. The initial, TF-based top 10 results (`List<Hit>`) have been successfully retrieved from the index by the `KeywordRetriever` (UC1).

3. Success Guarantee (Postconditions)

The system analyzes the question to determine the user's intent (`Intent`). The `Hit` list is re-scored and re-sorted based on heuristic bonuses (Proximity and Title/Document Importance). The final, optimized list of relevant document chunks is passed to the next stage (AnswerAgent).

4. Main Success Scenario (Basic Flow)

1. Orchestrator initiates the `IntentDetector` strategy using the raw user question.
2. **IntentDetector:** The `RuleBasedIntentDetector` normalizes the question and checks for defined keywords (e.g., "hoca", "staj", "kayıt") against its internal `RULE_SET`.
3. **IntentDetector Çıktısı:** `RuleBasedIntentDetector` returns the determined `Intent` (e.g., `STAFF_LOOKUP` or `POLICY_FAQ`).
4. **Note:** The determined Intent is used by the QueryWriter (UC1, Step 4) to add booster terms, *which influences the initial scores.*
5. Orchestrator receives the initial top results (`List<Hit>`) from the Retriever (UC1).
6. Orchestrator initiates the `Reranker` strategy with the initial `List<Hit>`.
7. **Reranker:** The `BasicReranker` iterates through each `Hit` in the list.

8. **Reranker:** It calculates the **Proximity Bonus** (e.g., based on chunk length) and **Title Bonus** (e.g., based on a low `docId` hash value).

9. **Reranker:** The new score is calculated as: $NewScore = OldScore + ProximityBonus + TitleBonus$. This new score overwrites `hit.score`.

10. **Reranker Sıralama:** The list is re-sorted based on the new, higher-resolution `score` (Descending).

11. **Output:** The Orchestrator receives the final, prioritized `List<Hit>` for the final answer generation.

5. Extensions (Alternative Flows)

- **3a.** Intent Detection Fails (UNKNOWN Intent):The `RuleBasedIntentDetector` fails to find any keyword matches in the `RULE_SET` for the user's question.
    - **3a1.** The `RuleBasedIntentDetector` returns the default intent, `UNKNOWN`.
    - **3a2.** (Flow returns to Step 4) The `QueryWriter` will proceed without appending any **booster terms**, resulting in a general keyword query.
- **5a. Retriever Returns Empty List**
    - **5a1.** The Orchestrator receives an empty `List<Hit>` and bypasses the Reranker entirely.
    - **5a2.** The system logs a warning and proceeds to the final step where the AnswerAgent will deliver a "No answer found" message.

- **8a. No Reranker Bonus Applied**
    - **8a1.** The `BasicReranker` calculates the new score as: $NewScore = OldScore + 0 + 0$. The initial TF score is preserved.
    - **8a2.** (Flow returns to Step 10) The list proceeds to re-sorting, but the relative order of such un-boosted hits is maintained by the tie-breaking rules of the initial TF score.
- **9a. Reranker Fails to Re-sort**
    - **9a1.**The system logs a critical error about the sorting failure.
    - **9a2.** The original, TF-sorted List<Hit> (from UC1) is returned, and a failure flag is logged for the Reranking step.

6. Special Requirements

- **Heuristic Impact:** The `BasicReranker` implementation must demonstrably improve the relevance of the top 3 results compared to the purely TF-based ranking.

- **Rule Maintainability:** The Intent rules must be easily modifiable by editing the internal `RULE_SET` in `RuleBasedIntentDetector` without altering the core logic.
- **Input Dependency:** The Reranking step (`BasicReranker`) must not require the original query terms, relying only on the data available in the `Hit` objects (score, body, docId).

7. Technology and Data Variations

- The Intent detection relies on simple string matching against Turkish/English keywords defined in a static internal map. Reranking is performed using custom heuristic formulas defined within the `BasicReranker` class. The final sorting uses Java's built-in `Comparator` interface. The `docId` check in Reranker uses the hash value of the original document ID, as the string ID is not directly available in `Hit`.