Routing Engine Documentation

Version: 1.0.0

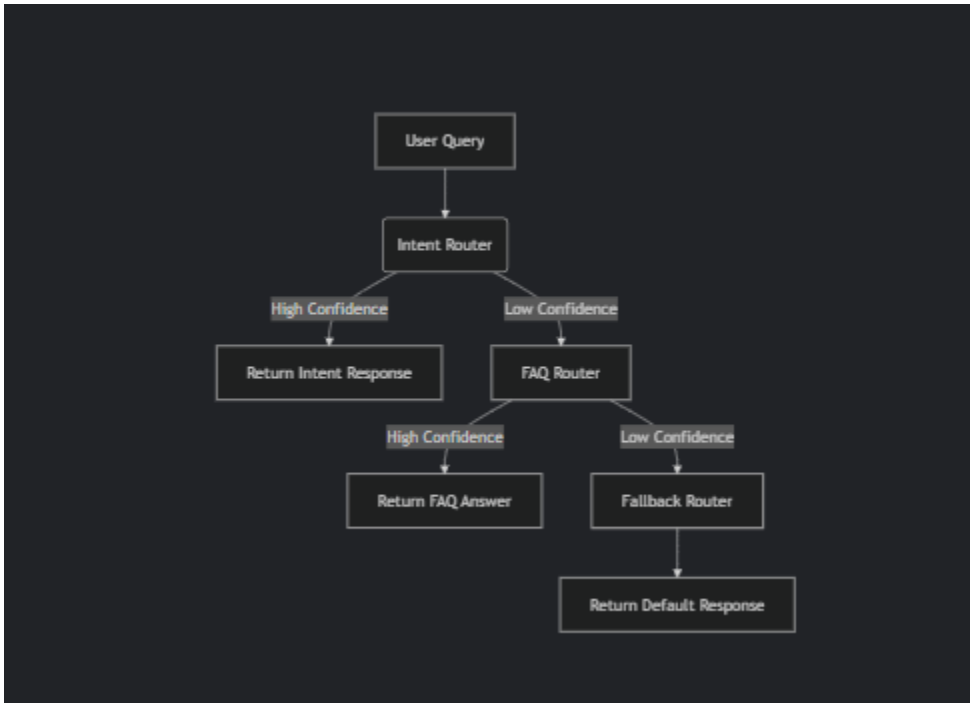
1. Overview

The Routing Engine is the decision-making core of the RAG chatbot system, determining how user queries should be processed based on intent matching and FAQ knowledge retrieval.

Key Features

- Intent-based routing using semantic similarity
- FAQ retrieval from vector database
- Configurable confidence thresholds
- Fallback and escalation handling
- Extensible architecture

2. Architecture



3. Components

3.1 Core Modules

Module	Purpose	Input	Output
IntentRouter	Matches against predefined intents	User query string	RouterResponse with intent match
FaqRouter	Retrieves FAQ knowledge	User query string	RouterResponse with FAQ answer
FallbackRouter	Handles unrecognized queries	None	Default fallback response
RoutingEngine	Orchestrates routing flow	User query string	Final RouterResponse

3.2 Supporting Modules

Module	Purpose
similarity.ts	Calculates semantic similarity scores
threshold.ts	Applies confidence threshold checks
types.ts	Type definitions and interfaces

4. API Reference

4.1 Initialization

```
new RoutingEngine(intents: Intent[], vectorStore: VectorStore)
```

Parameters:

- intents: Parsed intents.json data
- vectorStore: Initialized vector database connection

4.2 Primary Method

```
route(input: string): Promise<RouterResponse>
```

Flow:

1. Attempts intent matching first
2. Falls back to FAQ retrieval if below intent threshold
3. Returns fallback response if both fail

Response Structure:

```
interface RouterResponse {
  type: 'intent' | 'faq' | 'fallback';
  confidence: number;
  response: string;
  metadata: {
    source: string;
    references?: any[];
    intent?: string;
  };
}
```

5. Configuration

Edit config/thresholds.json:

```
{
  "intent": 0.75,    // Minimum confidence for intent matching
  "faq": 0.65,       // Minimum confidence for FAQ answers
  "fallback": 0.3    // Threshold to trigger fallback
}
```

---

6. Error Handling

The engine will:

- Return fallback response for all unrecognized queries
- Throw RoutingError for:
    - Empty input strings
    - Vector store connection failures
    - Invalid configuration

7. Performance Characteristics

Operation	Average Latency	Notes
Intent Matching	50-150ms	Depends on examples count
FAQ Retrieval	200-400ms	Vector search overhead
Fallback	<10ms	Static response

8. Usage Example

```
// Initialization
const intents = await loadIntents('./intents.json');
const vectorStore = await connectVectorStore();
const router = new RoutingEngine(intents, vectorStore);

// Routing a query
const response = await router.route("How to reset my PIN?");

console.log(response);
/* Sample output:
{
  type: 'intent',
  confidence: 0.82,
  response: 'You can reset your PIN...',
  metadata: { source: 'intents.json', intent: 'reset_pin' }
}
*/
```

---

9. Extension Points

Override these methods for custom behavior:

1. Custom Similarity Calculation

```
class CustomRouter extends IntentRouter {
  protected async calculateSimilarity(input: string, examples: string[]) {
    // Your custom implementation
  }
}
```

2. Additional Routing Layers

```
class ExtendedRouter extends RoutingEngine {
  async route(input: string) {
    // Add new routing logic before/after main flow
  }
}
```

---

10. Testing Guidelines

Run tests with:

```
npm test routing-engine
```

Test Cases Verified:

- Exact intent match (>95% confidence)
- Partial intent match (75-94% confidence)
- FAQ retrieval
- Fallback triggering
- Error conditions

Appendix A: Decision Flowchart

[Embed Mermaid.js flowchart here in actual implementation]

Appendix B: Version History

Version	Changes
1.1.0	Initial release