RagChatModelService

September 4, 2026

RagChatModelService

Kind: Service

Source: `atloria-monorepo/apps/api/src/technical-docs/rag/chat-model.service.ts`

Generation model for grounded chat. Uses gpt-5.6-terra via chat/completions on the Azure
Foundry resource (AZURE_RESPONSES_*), which hosts the gpt-5.6 deployments (the legacy primary
Azure OpenAI resource only had gpt-5.5). terra is a reasoning model, so we send `max_completion_tokens` (not `max_tokens`) and omit `temperature` (rejected by reasoning models). Endpoint/key/deployment are overridable via RAG_CHAT_* for a dedicated resource.

`RagChatModelService` is a NestJS service that generates grounded chat completions using the **gpt-5.5** reasoning model on the **primary Azure OpenAI** resource via the `chat/completions` API. It is intentionally self-contained because the shared Responses provider targets a different Azure resource that does not host gpt-5.5. It also applies reasoning-model constraints by sending `max_completion_tokens` (not `max_tokens`) and omitting `temperature`.

Methods

Method	Signature	Returns	Description
`complete`	`complete(prompt: string, maxCompletionTokens: unknown)`	`Promise<string>`	Single-shot completion for a prompt.

Method	Signature	Returns	Description
`completeStream`	`completeStream(prompt: string, maxCompletionTokens: unknown)`	`AsyncGenerator<string, { text: string; inputTokens: number; outputTokens: number; totalTokens: number }, void>`	Streaming completion (Foundation F3): yields text deltas as they arrive, then returns the full text + usage.
`completeWithUsage`	`completeWithUsage(prompt: string, maxCompletionTokens: unknown)`	`Promise<{ text: string; inputTokens: number; outputTokens: number; totalTokens: number }>`	Completion + token usage (Tier 1 metering: chat spend must be attributable per org).

Dependencies

- `ConfigService`

Where it refuses work

- `RagChatModelService` stops the work with `Error` when `!this.client` — “RAG chat model not configured”, in 2 places.

When something fails

- `RagChatModelService` handles failure in 1 place: it discards it silently in all 1.

Diagram

```
sequenceDiagram
  autonumber
  participant Caller as API Handler / RAG Orchestrator
  participant RagSvc as RagChatModelService
  participant Azure as Azure OpenAI (Primary Resource)
  participant Model as gpt-5.5 (reasoning)

  Caller->>RagSvc: build messages (system + user + grounded context)
  RagSvc->>RagSvc: validate reasoning-model params\n(no temperature, use max_completion_tokens)
```

```
  RagSvc->>Azure: POST /chat/completions\n{ model: "gpt-5.5", messages, max_completion_tokens,
  Azure->>Model: Run reasoning completion
  Model-->>Azure: completion output
  Azure-->>RagSvc: response (choices/message)
  RagSvc-->>Caller: grounded assistant message/content
```

Usage

```
import { Injectable } from '@nestjs/common';
import { RagChatModelService } from './technical-docs/rag/chat-model.service';

@Injectable()
export class RagAnswerService {
  constructor(private readonly ragChatModel: RagChatModelService) {}

  async answer(question: string, contextChunks: string[]) {
    const groundedContext = contextChunks.map((c, i) => `Source ${i + 1}:\n${c}`).join('\n\n');

    // Typical RAG shape: system rules + user question + grounded context
    const messages = [
      {
        role: 'system',
        content:
          'You are a helpful assistant. Use ONLY the provided sources. ' +
          'If the answer is not in the sources, say you do not know.',
      },
      { role: 'user', content: `Question:\n${question}\n\nSources:\n${groundedContext}` },
    ];

    // NOTE: reasoning model usage: omit temperature; use max_completion_tokens
    const result = await this.ragChatModel.createChatCompletion({
      model: 'gpt-5.5',
      messages,
      max_completion_tokens: 800,
    });

    return {
      answer: result.choices?.[0]?.message?.content ?? '',
      raw: result,
    };
  }
}
```

AI Coding Instructions

- Keep this service **self-contained**: do not “refactor to shared Responses provider” unless the target Azure resource is confirmed to host **gpt-5.5**.

- For **reasoning models**, send `max_completion_tokens` and **do not** include `temperature` (Azure/OpenAI will reject it for reasoning models).
- Preserve the grounded-chat contract: callers should pass **messages that include the retrieved context** and system constraints (e.g., “use only sources”).
- When integrating, verify the **Azure OpenAI deployment/model name** and endpoint correspond to the **primary resource** intended for gpt-5.5 `chat/completions`.

Relationships

- DEPENDS_ON → `configservice`

Referenced By

- `TechDocsChatService` (DEPENDS_ON)
- `TechDocsRagService` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)