UserDocsGeneratorService

September 4, 2026

UserDocsGeneratorService

Kind: Service

Source: `atloria-monorepo/apps/api/src/documentation/services/user-docs-generator.service.ts`

Multi-agent service for generating user documentation

Architecture:

1. Generator Agent (Azure Claude) - Creates initial documentation
2. Reviewer Agent (Azure OpenAI) - Humanizes and improves language (optional)

`UserDocsGeneratorService` is a NestJS backend service that orchestrates a multi-agent pipeline to generate end-user documentation from internal inputs (e.g., domain context, feature notes, or source artifacts). It delegates first-pass drafting to a “Generator” agent (Azure Claude) and can optionally pass the draft through a “Reviewer” agent (Azure OpenAI) to improve tone, clarity, and human readability. This service sits in the API layer and acts as the coordination point between application code and external LLM providers.

Methods

Method	Signature	Returns	Description
`setCompetitorContext`	`setCompetitorContext(competitorContext: string	null)`	`void`
`generateDocumentation`	`generateDocumentation(appMap: any, screenshots: Record<string, any[]>, workflows: DetectedWorkflow[], businessContext: {		

```
  marketing: string[];
  workflows: string[];
  brand: string[];
```

```
    processes: string[];
  }, options: {
    generatorProvider?: 'azure-claude' | 'azure-openai';
    reviewerProvider?: 'azure-claude' | 'azure-openai';
    enableReviewer?: boolean;
    competitorContext?: string; // NEW: Optional competitor docs context
  })` | `Promise<GeneratedDocumentation>` | Generate complete user documentation |
```

| `generateDocumentationWithSaving` | `generateDocumentationWithSaving(projectId: string, user: JwtPayload, appMap: any, screenshots: Record<string, any[]>, workflows: DetectedWorkflow[], businessContext: { marketing: string[]; workflows: string[]; brand: string[]; processes: string[]; }, options: { generatorProvider?: 'azure-claude' | 'azure-openai'; reviewerProvider?: 'azure-claude' | 'azure-openai'; enableReviewer?: boolean; competitorContext?: string; })` | `Promise<SyncBatchResponseDto>` | Generate documentation and save incrementally to database This method generates workflow documentation and saves each workflow immediately to the database us… |

Dependencies

- `AzureClaudeProvider`
- `AzureOpenAIProvider`
- `SyncService`

Where it refuses work

- `UserDocsGeneratorService` stops the work with an early return when `modalTrigger.includes('edit')`, in 2 places.
- `UserDocsGeneratorService` stops the work with an early return when `stateMachines.length === 0`.
- `UserDocsGeneratorService` stops the work with an early return when `!page?.stateMachines || page.stateMachines.length === 0`.
- `UserDocsGeneratorService` stops the work with an early return when `!page?.uiInteractions || page.uiInteractions.length === 0`.
- `UserDocsGeneratorService` stops the work with an early return when `screenshots.length === 0`.
- `UserDocsGeneratorService` stops the work with an early return when `enhancedInfo.length === 0`.

Diagram

```
sequenceDiagram
  autonumber
  participant Caller as API/Controller/Job
  participant Service as UserDocsGeneratorService
  participant Gen as Generator Agent (Azure Claude)
  participant Rev as Reviewer Agent (Azure OpenAI, optional)

  Caller->>Service: generateUserDocs(input, options)
  Service->>Gen: createInitialDocumentation(input)
  Gen-->>Service: draftDocs

  alt reviewer enabled
    Service->>Rev: refineLanguage(draftDocs)
    Rev-->>Service: reviewedDocs
    Service-->>Caller: reviewedDocs
  else reviewer disabled
    Service-->>Caller: draftDocs
  end
```

Usage

```
import { Injectable } from '@nestjs/common';
import { UserDocsGeneratorService } from './documentation/services/user-docs-generator.service'

@Injectable()
export class DocsJob {
  constructor(private readonly userDocs: UserDocsGeneratorService) {}

  async run() {
    const input = {
      productName: 'Atloria',
      audience: 'End users',
      features: [
        { name: 'Workspaces', notes: 'Create and organize projects and docs.' },
        { name: 'Search', notes: 'Find content across the workspace.' },
      ],
      constraints: {
        tone: 'Clear, friendly, non-technical',
        format: 'Markdown',
      },
    };

    // Optionally enable a second pass to humanize wording and improve flow.
    const options = { reviewerEnabled: true };

    const docsMarkdown = await this.userDocs.generateUserDocs(input, options);

    // Persist, return, or publish documentation content.
    return docsMarkdown;
```

```
  }
}
```

AI Coding Instructions

- Keep the service as an **orchestrator**: generation logic lives in the agent adapters/clients; this service should coordinate steps, inputs, and outputs.
- Treat the Reviewer step as **optional** and make it a clear, explicit flag; avoid silently adding reviewer calls (cost/latency).
- Normalize and validate the model inputs (tone/format/audience) before sending to providers to prevent prompt drift and inconsistent outputs.
- Handle provider failures deterministically: timeouts, retries, and fallbacks should be centralized so callers get a predictable error/result shape.

Relationships

- DEPENDS_ON → `AzureClaudeProvider`
- DEPENDS_ON → `AzureOpenAIProvider`
- DEPENDS_ON → `SyncService`

Referenced By

- `DocumentationModule` (MODULE_PROVIDES)
- `DocumentationService` (DEPENDS_ON)