AzureSearchService

September 4, 2026

AzureSearchService

Kind: Service

Source: `atloria-monorepo/apps/api/src/documentation/services/azure-search.service.ts`

Azure AI Search Service

Wrapper around Azure AI Search SDK.
Handles:

- Index creation and management
- Document indexing
- Full-text search with filtering
- Project-level isolation (multi-tenancy)
- Faceted search

`AzureSearchService` is a NestJS service that wraps the Azure AI Search SDK to provide a consistent API for indexing and searching documents within the backend. It manages index lifecycle (create/update), document ingestion, and full-text search with filtering and faceting. The service enforces project-level isolation (multi-tenancy) so each project's data is indexed and queried independently.

Methods

Method	Signature	Returns	Description
`initializeIndex`	`initializeIndex()`	`Promise<void>`	Initialize search index (run on app startup)
`indexDocument`	`indexDocument(document: SearchDocument)`	`Promise<void>`	Index a single document
`indexBatch`	`indexBatch(documents: SearchDocument[])`	`Promise<void>`	Index multiple documents in batch
`updateDocument`	`updateDocument(document: SearchDocument)`	`Promise<void>`	Update a document

Method	Signature	Returns	Description
`deleteDocument`	`deleteDocument(documentId: string)`	`Promise<void>`	Delete a document
`search`	`search(query: string, options: SearchOptions)`	`Promise<SearchResponse>`	Search documents with project filtering
`getSuggestions`	`getSuggestions(query: string, organizationId: string, projectId: string, limit: number)`	`Promise<string[]>`	Get search suggestions (autocomplete)
`deleteVersionDocuments`	`deleteVersionDocuments(projectId: string, docVersionIds: string[])`	`Promise<number>`	Delete all indexed documents belonging to the given doc versions of a project.
`deleteProjectDocuments`	`deleteProjectDocuments(projectId: string)`	`Promise<void>`	Delete all documents for a project

Dependencies

- `ConfigService`
- `PrismaService`

Where it refuses work

- `AzureSearchService` stops the work with an early return when `!this.searchClient`, in 2 places.
- `AzureSearchService` stops the work with an early return when `documents.length === 0`.
- `AzureSearchService` stops the work with an early return when `!sanitizedQuery`.
- `AzureSearchService` stops the work with an early return when `!facet`.
- `AzureSearchService` stops the work with an early return when `!this.searchClient || docVersionIds.length === 0`.

When something fails

- `AzureSearchService` handles failure in 10 places: it lets it reach the caller in 9, and turns it into a return value in 1.

Diagram

```
sequenceDiagram
  autonumber
  participant C as API Controller/Resolver
  participant S as AzureSearchService (NestJS)
  participant SDK as Azure AI Search SDK
  participant IDX as Search Index
  participant DOC as Indexed Documents

  C->>S: ensureIndex(projectId, schema)
  S->>SDK: createOrUpdateIndex(indexName, schema)
  SDK->>IDX: Upsert index definition
  IDX-->>SDK: OK
  SDK-->>S: OK
  S-->>C: ready

  C->>S: indexDocuments(projectId, docs)
  S->>SDK: upload/mergeOrUpload(indexName, docs)
  SDK->>DOC: Write documents
  DOC-->>SDK: Result
  SDK-->>S: Result
  S-->>C: indexed

  C->>S: search(projectId, query, filters, facets)
  S->>SDK: search(indexName, query, options)
  SDK->>IDX: Query w/ filter + facets
  IDX-->>SDK: Hits + facet counts
  SDK-->>S: Results
  S-->>C: results
```

Usage

```typescript
import { Controller, Get, Post, Body, Query } from '@nestjs/common';
import { AzureSearchService } from './azure-search.service';

@Controller('search')
export class SearchController {
  constructor(private readonly azureSearch: AzureSearchService) {}

  @Post('reindex')
  async reindex(@Body() body: { projectId: string; documents: any[] }) {
    // Ensure the tenant-specific index exists and is up to date
    await this.azureSearch.ensureIndex(body.projectId, {
      // Example index schema (shape depends on your implementation)
      name: 'documents',
      fields: [
        { name: 'id', type: 'Edm.String', key: true, filterable: true },
        { name: 'projectId', type: 'Edm.String', filterable: true },
        { name: 'title', type: 'Edm.String', searchable: true },
        { name: 'content', type: 'Edm.String', searchable: true },
        { name: 'tags', type: 'Collection(Edm.String)', filterable: true, facetable: true },
      ],
    });

    // Index documents (typically include projectId for isolation and filtering)
```

```
    await this.azureSearch.indexDocuments(body.projectId, body.documents);

    return { ok: true };
  }

  @Get()
  async search(
    @Query('projectId') projectId: string,
    @Query('q') q = '*',
    @Query('tag') tag?: string,
  ) {
    const filter = tag ? `tags/any(t: t eq '${tag}')` : undefined;

    const results = await this.azureSearch.search(projectId, q, {
      filter,
      facets: ['tags,count:10'],
      top: 20,
      skip: 0,
    });

    return results;
  }
}
```

AI Coding Instructions

- Preserve project-level isolation: always derive the index name and/or filter scope from `projectId` and avoid cross-project queries by default.
- Keep index schema changes backward-compatible where possible; when changing fields, update `ensureIndex` flows and reindex strategy accordingly.
- When adding filters/facets, validate and sanitize user inputs to avoid malformed OData filter strings and unexpected query errors.
- Use Azure SDK batching/merge-or-upload patterns for indexing to avoid payload limits and reduce partial-failure risk; surface indexing errors with actionable logs.
- Ensure integration points (env/config for endpoint, API key, index naming conventions) are centralized so deployments don’t diverge between environments.

Relationships

- DEPENDS_ON → `configservice`
- DEPENDS_ON → `PrismaService`

Referenced By

- `DocumentationModule` (MODULE_PROVIDES)
- `DocumentationModule` (MODULE_EXPORTS)
- `SearchController` (DEPENDS_ON)

- `DocumentIndexingService` (DEPENDS_ON)
- `SupportAgentService` (DEPENDS_ON)