KnowledgeGraphService

September 4, 2026

KnowledgeGraphService

Kind: Service

Source: `atloria-monorepo/apps/api/src/graph/services/knowledge-graph.service.ts`

KnowledgeGraphService — tenant-scoped knowledge-graph access.

SECURITY (cross-tenant): every method is scoped by `projectId`, which the controller obtains
from an org-verified route/body/query param (ResourceOrgGuard, kind: 'project'). Neo4j `:Entity`
nodes are a single shared label across all tenants, so EVERY Cypher here filters on `{ projectId: $projectId }` — start nodes, traversed neighbours, and delete targets alike — so a
caller can never read, mutate, or delete a node outside their own project. Nodes follow the same
convention the kg-sync writer uses: a composite `id = "${projectId}-${entityId}"`, a raw `entityId`, and a `projectId` property (see kg-sync.service.ts).

Label / relationship-type interpolation is validated against the EntityType / RelationType enums
before it reaches Cypher — these strings become labels/rel-types (which cannot be parameterised),
so a whitelist is the guard against Cypher injection.

`KnowledgeGraphService` provides tenant-scoped access to the Neo4j knowledge graph for a single project. It creates, queries, traverses, searches, and deletes entities and relationships while ensuring every Cypher operation filters by `projectId`. Entity labels and relationship types are validated against application enums before being interpolated into Cypher.

Methods

Method	Signature	Returns
`createEntity`	`createEntity(projectId: string, entity: Partial<KGEntity>)`	`Promise<KGEntity>`
`createRelation`	`createRelation(projectId: string, relation: KGRelation)`	`Promise<void>`
`getEntity`	`getEntity(projectId: string, entityId: string)`	`Promise<KGEntity>`
`findRelated`	`findRelated(projectId: string, entityId: string, depth: unknown, type: RelationType)`	`Promise<KGEntity[]>`
`findPath`	`findPath(projectId: string, fromId: string, toId: string)`	`Promise<Path`
`searchEntities`	`searchEntities(projectId: string, query: string, type: EntityType)`	`Promise<KGEntity[]>`
`getEntityRelationships`	`getEntityRelationships(projectId: string, entityId: string)`	`Promise<any[]>`
`deleteEntity`	`deleteEntity(projectId: string, entityId: string)`	`Promise<void>`
`getStatistics`	`getStatistics(projectId: string)`	`Promise<any>`

Dependencies

- `Neo4jService`

Where it refuses work

- `KnowledgeGraphService` stops the work with `BadRequestException` when `!type || !Object.values(EntityType).includes(type as EntityType)`.
- `KnowledgeGraphService` stops the work with `BadRequestException` when `!type || !Object.values(RelationType).includes(type as RelationType)`.
- `KnowledgeGraphService` stops the work with `NotFoundException` when `!result || result.length === 0` — “Entity not found in this project”.
- `KnowledgeGraphService` stops the work with `NotFoundException` when `result.length === 0`.
- `KnowledgeGraphService` stops the work with `NotFoundException` when `deleted === 0`.

- `KnowledgeGraphService` stops the work with an early return when `!Number.isFinite(d) || d < 1`.

Diagram

```
sequenceDiagram
  participant Controller
  participant Guard as ResourceOrgGuard
  participant Service as KnowledgeGraphService
  participant Neo4j

  Controller->>Guard: Validate project route/body/query parameter
  Guard-->>Controller: Verified projectId
  Controller->>Service: findRelated(projectId, entityId, options)
  Service->>Service: Validate entity label / relation type
  Service->>Neo4j: Cypher query scoped by projectId
  Note over Neo4j: Filters start nodes, neighbours,<br/>and targets by projectId
  Neo4j-->>Service: Matching entities / paths
  Service-->>Controller: Tenant-safe graph result
```

Usage

```
import { Controller, Get, Param, Query } from '@nestjs/common';
import { KnowledgeGraphService } from '../graph/services/knowledge-graph.service';

@Controller('projects/:projectId/graph')
export class GraphController {
  constructor(
    private readonly knowledgeGraphService: KnowledgeGraphService,
  ) {}

  @Get('entities/:entityId/related')
  async getRelatedEntities(
    @Param('projectId') projectId: string,
    @Param('entityId') entityId: string,
    @Query('depth') depth = '1',
  ) {
    // projectId should be verified by ResourceOrgGuard before this call.
    return this.knowledgeGraphService.findRelated(
      projectId,
      entityId,
      { depth: Number(depth) },
    );
  }

  @Get('search')
  async search(
    @Param('projectId') projectId: string,
```

```
    @Query('q') query: string,
  ) {
    return this.knowledgeGraphService.searchEntities(projectId, query);
  }
}
```

AI Coding Instructions

- Always pass the organization-verified `projectId` into service methods; never derive tenant scope from an untrusted entity ID or client-provided graph node ID.
- Preserve `projectId` filtering in every Cypher clause, including traversal neighbours, optional matches, relationship targets, and delete operations.
- Use the established composite entity ID format, `${projectId}-${entityId}`, alongside the raw `entityId` and `projectId` node properties.
- Validate any dynamic entity label or relationship type against `EntityType` or `RelationType` before interpolating it into Cypher; Neo4j labels and relationship types cannot be query parameters.
- Keep graph writes compatible with the `kg-sync` writer conventions so synchronized and manually created nodes remain queryable together.

Relationships

- DEPENDS_ON → `Neo4jService`