GraphController

September 4, 2026

GraphController

Kind: Controller

Source: `atloria-monorepo/apps/api/src/graph/graph.controller.ts`

SECURITY (cross-tenant): the knowledge graph is a single shared Neo4j store, so every operation
here is scoped to an org-verified `projectId`. The ResourceOrgGuard resolves that projectId to
its owning organization and 403s if it isn't the caller's — the id arrives via the request body
(writes) or a required `projectId` query param (reads). The service then filters every Cypher by
that projectId, so no route can read, mutate, or delete another tenant's graph.
`document/:id/ extract` is scoped by the DOCUMENT instead (org-resolvable), and persists into the document's own
project.

`GraphController` exposes API endpoints for reading, mutating, and extracting knowledge-graph data stored in the shared Neo4j graph. Every graph operation is tenant-isolated through an org-verified `projectId`: `ResourceOrgGuard` validates ownership before the controller delegates to the graph service, which applies the project filter to Cypher queries. Document extraction is authorized through the document resource and writes extracted graph data into that document’s project.

Diagram

```
graph LR
  Client[API Client] --> Controller[GraphController]
  Controller --> Guard[ResourceOrgGuard]

  Guard -->|Verify projectId ownership| Project[Project / Organization]
  Guard -->|403 if cross-tenant| Client
```

```
  Guard --> Service[GraphService]
  Service -->|Filter all Cypher by projectId| Neo4j[(Shared Neo4j Store)]

  Client --> Extract["POST /graph/document/:id/extract"]
  Extract --> Document[Document Resource]
  Document -->|Resolve owning org and project| Service
```

Usage

```
// Read graph data for a project.
// The projectId is required so ResourceOrgGuard can verify organization ownership.
const response = await fetch(
  `${API_URL}/graph?projectId=${encodeURIComponent(projectId)}`,
  {
    headers: {
      Authorization: `Bearer ${accessToken}`,
    },
  },
);

if (!response.ok) {
  throw new Error(`Unable to load graph: ${response.statusText}`);
}

const graph = await response.json();

// Extract graph entities/relationships from a document.
// Authorization is resolved from the document, and results are persisted
// into the document's associated project.
await fetch(`${API_URL}/graph/document/${documentId}/extract`, {
  method: "POST",
  headers: {
    Authorization: `Bearer ${accessToken}`,
    "Content-Type": "application/json",
  },
});
```

AI Coding Instructions

- Require an org-verifiable `projectId` for every graph read and write route; reads receive it through query parameters and writes through the request body.
- Preserve `ResourceOrgGuard` on project-scoped endpoints—never rely only on client-provided project IDs for tenant isolation.
- Ensure every new GraphService Cypher query filters nodes and relationships by `projectId`, including update and delete operations.

- For `document/:id/extract`, authorize through the document resource rather than a caller-supplied project ID, then persist results to the document's own project.
- Return authorization failures as `403` when the requested project or document belongs to another organization.

Relationships

- MODULE_DECLARES → `createEntity`
- MODULE_DECLARES → `createRelation`
- MODULE_DECLARES → `getEntity`
- MODULE_DECLARES → `getRelated`
- MODULE_DECLARES → `getRelationships`
- MODULE_DECLARES → `findPath`
- MODULE_DECLARES → `search`
- MODULE_DECLARES → `extractEntities`
- MODULE_DECLARES → `detectFlow`
- MODULE_DECLARES → `detectImpact`
- MODULE_DECLARES → `getStatistics`
- MODULE_DECLARES → `deleteEntity`
- DEPENDS_ON → `knowledgegraphservice`
- DEPENDS_ON → `EntityExtractorService`
- DEPENDS_ON → `FlowDetectorService`

Referenced By

- `GraphModule` (MODULE_DECLARES)