FlowDetectorService

September 4, 2026

FlowDetectorService

Kind: Service

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

`FlowDetectorService` is a NestJS backend service responsible for analyzing graph data to identify execution or dependency flows and their downstream impact. It exposes methods for detecting a `Flow` and resolving the affected `KGEntity` nodes, enabling other API and graph features to reason about relationships within the knowledge graph.

Methods

Method	Signature	Returns
`detectFlow`	`detectFlow(projectId: string, entityId: string, flowName: string)`	`Promise<Flow>`
`detectImpact`	`detectImpact(projectId: string, entityId: string)`	`Promise<KGEntity[]>`

Dependencies

- `KnowledgeGraphService`

Diagram

```
sequenceDiagram
  participant Consumer as API / Graph Consumer
  participant Service as FlowDetectorService
  participant Graph as Graph Data Source
  participant Flow as Flow
  participant Entities as KGEntity[]

  Consumer->>Service: detectFlow()
```

```
  Service->>Graph: Query graph relationships
  Graph-->>Service: Nodes and edges
  Service->>Service: Build and validate flow
  Service-->>Consumer: Flow

  Consumer->>Service: detectImpact()
  Service->>Graph: Traverse dependent entities
  Graph-->>Service: Related KGEntity nodes
  Service-->>Consumer: KGEntity[]
```

Usage

```
import { Injectable } from '@nestjs/common';
import { FlowDetectorService } from './flow-detector.service';

@Injectable()
export class GraphAnalysisService {
  constructor(
    private readonly flowDetectorService: FlowDetectorService,
  ) {}

  async analyzeGraph() {
    const flow = await this.flowDetectorService.detectFlow();
    const impactedEntities = await this.flowDetectorService.detectImpact();

    return {
      flow,
      impactedEntities,
      impactedEntityCount: impactedEntities.length,
    };
  }
}
```

AI Coding Instructions

- Keep graph traversal and flow-construction logic inside `FlowDetectorService`; callers should consume the returned `Flow` and `KGEntity[]` rather than replicate detection logic.
- Treat `detectFlow()` and `detectImpact()` as asynchronous operations and always `await` their results.
- Ensure graph queries handle missing nodes, disconnected relationships, and cyclic dependencies without producing invalid flows or infinite traversal.
- Preserve NestJS dependency-injection patterns when adding graph repositories, query services, or logging dependencies.

- Update consumers of impact analysis when changing the shape or semantics of returned `KGEntity` instances.

Relationships

- DEPENDS_ON → `knowledgegraphservice`

Referenced By

- `GraphController` (DEPENDS_ON)
- `GraphModule` (MODULE_PROVIDES)
- `GraphModule` (MODULE_EXPORTS)