Project Documentation: Inventory Management GraphQL API

December 15, 2024

Contents

1 Introduction

This documentation describes the backend API for a personal inventory management System. The system organizes inventory items as *nodePoints* in a hierarchical (tree-like) structure representing their physical location, enabling querying, updating, and managing items and their historical versions via a GraphQL interface.

1.1 Project Overview

The backend provides:

- Storage and retrieval of inventory items organized as a hierarchy of *nodePoints*.
- Ability to track history (*nodePointHistory*) of changes, including additions, updates, and deletions.
- GraphQL queries and mutations to interact with the data.

2 System Architecture (Brief)

2.1 Technologies Used

- **Node.js:** Runs the server.
- **GraphQL Yoga:** Provides a GraphQL server environment.
- **PostgreSQL:** Stores `nodePoint` and `nodePointHistory` records.
- **Docker:** Manages the PostgreSQL database container.

2.2 Database Schema (Summary)

Two primary database tables support the GraphQL operations:

- **nodePoint:** Stores hierarchical items with attributes like `id`, `parent`, `title`, `description`, `data`, `version`, and `deleted` flag.
- **nodePointHistory:** Stores historical records of changes applied to nodePoints.

3 GraphQL Schema and Operations

The GraphQL schema defines both queries and mutations. Clients interact with these endpoints to:

- Retrieve nodes, hierarchical structures, and historical records.
- Create, update, and delete nodes.
- Move nodes within the hierarchy and restore previously deleted nodes.

3.1 Object Types

3.1.1 NodePoint

A `NodePoint` represents an inventory item (or location) in the hierarchy.
Fields:

- `id: Int` - Unique identifier of the node.
- `parent: Int` - Parent node's ID.
- `title: String` - Node's title or name.
- `description: String` - Optional descriptive text.
- `data: JSON` - JSON data field for additional info.
- `version: Int` - Version number incremented on updates.
- `deleted: Boolean` - Indicates whether the node is deleted (soft deletion).

3.1.2 NodePointHistory

A `NodePointHistory` entry records a historical snapshot of a node at the time of a particular action.
Fields:

- `id: Int` - Unique identifier of the history record.
- `nodePointId: Int` - ID of the associated nodePoint.
- `version: Int` - Version number of the node at that time.
- `title: String` - Title of the node at that historical moment.
- `description: String` - Description of the node at that time.
- `data: JSON` - Associated JSON data at that time.
- `action: String` - The action taken (e.g., `"update"`, `"delete"`, `"restore"`).
- `timestamp: String` - When the action occurred.

3.2 Query Operations

The following queries allow clients to retrieve node information and history:

3.2.1 fetchAll

Description: Retrieves all `nodePoint` entries, including those marked as deleted.
Query Example:

```
query {
  fetchAll {
    id
    parent
    title
    deleted
  }
}
```

Use Case: Useful for retrieving a complete list of nodes for initial data loads, audits, or administrative tasks.

3.2.2 fetchHierarchy(id: Int)

Description: Given a node's `id`, returns that node and all its descendants, excluding deleted nodes. This provides a subtree of the hierarchy.
Query Example:

```
query {
  fetchHierarchy(id: 5) {
    id
    parent
    title
    deleted
  }
}
```

Use Case: Ideal for showing a segment of the tree structure, helping users visualize where a particular item sits in the larger inventory.

3.2.3 fetchNodeHistory(nodePointId: Int)

Description: Returns historical records for a given node, including past titles, data, and actions (updates, deletes, etc.).
Query Example:

```
query {
  fetchNodeHistory(nodePointId: 10) {
    version
    title
    description
    action
    timestamp
  }
}
```

Use Case: Helps developers audit changes and understand the evolution of a node over time.

4 Mutation Operations

The mutations enable creating, updating, moving, and deleting nodes in the hierarchy, as well as managing their history.

4.1 Input Types

4.1.1 NodePointInput

- `parent: Int`
- `title: String`
- `description: String`
- `data: JSON`

4.1.2 UpdateNodePointInput

- `id: Int`
- `parent: Int`
- `title: String`
- `description: String`
- `data: JSON`

4.2 Available Mutations

4.2.1 addNodePoint(input: NodePointInput)

Description: Inserts a single new nodePoint into the database.
Mutation Example:

```
mutation {
  addNodePoint(input: {
    parent: 1,
    title: "New Item",
    description: "A newly added item",
    data: { "color": "blue" }
  }) {
    id
    title
    version
  }
}
```

Use Case: Adds a new item to the inventory hierarchy.

4.2.2 addMultipleNodePoints(nodes: [NodePointInput])

Description: Inserts multiple nodePoints in a single request, streamlining bulk data entry.
Mutation Example:

```
mutation {
  addMultipleNodePoints(nodes: [
    { parent: 1, title: "Item A", data: {} },
    { parent: 1, title: "Item B", data: {} }
  ]) {
    id
    title
  }
}
```

4.2.3 updateNodePoint(input: UpdateNodePointInput)

Description: Updates the specified nodePoint and records the previous version in `nodePointHistory`.
Mutation Example:

```
mutation {
  updateNodePoint(input: {
    id: 10,
    title: "Updated Title"
  }) {
    id
    title
    version
  }
}
```

Use Case: Allows modifying item attributes and tracking versions.

4.2.4 deleteNodePoint(id: Int)

Description: Soft-deletes the nodePoint, marking it as deleted and logging the action in the history.
Mutation Example:

```
mutation {
  deleteNodePoint(id: 10) {
    id
    deleted
  }
}
```

4.2.5 moveMultipleNodePoints(newParent: Int, nodeIds: [Int])

Description: Moves multiple specified nodePoints under a new parent, helping reorganize the hierarchy.
Mutation Example:

```
mutation {
  moveMultipleNodePoints(newParent: 2, nodeIds: [10, 11, 12]) {
    id
    parent
  }
}
```

4.2.6 restoreNodePoint(id: Int)

Description: Restores a previously soft-deleted nodePoint, incrementing its version and logging the action.
Mutation Example:

```
mutation {
  restoreNodePoint(id: 10) {
    id
    deleted
    version
  }
}
```

4.2.7 hardDeleteNodePoint(id: Int)

Description: Permanently deletes a nodePoint and all its descendants from the database, removing it entirely from the system.
Mutation Example:

```
mutation {
  hardDeleteNodePoint(id: 10) {
    id
  }
}
```

5 Usage and Setup

5.1 Building and Running the Server

Developers can run the following to start the backend and the frontend:

```
git clone git@github.com:CorbinIvon/2024fa-csci-36-inventory-system.git
cd 2024fa-csci-36-inventory-system
npm install turbo --global
npm install
cd apps/backend
```

```
docker-compose up -d
cd ../../
turbo dev
```

The GraphQL endpoint will be accessible at `http://localhost:4000/graphql`.
Note: Docker must be running, or the server will fail.

6 Additional Considerations

6.1 Error Handling

Errors generally return as GraphQL errors. For example, attempting to fetch or update a non-existent node will result in an error message within the GraphQL response.

7 Conclusions

This documentation provides an overview of the GraphQL schema and endpoints for the inventory management backend. Developers can use these queries and mutations to manage hierarchical items, track their history, and maintain a versioned audit trail.