Nest Kudos GraphQL API – Developer Guide

1. Introduction

This guide is intended for developers working on or integrating with the Nest Kudos GraphQL API. It provides setup instructions, API usage examples, project structure overview, and guidelines for extending or maintaining the codebase.

2. Prerequisites

- Node.js v20.19.4 or higher

- npm (Node package manager)

- Optional: VS Code or any TypeScript-compatible IDE

3. Setup Instructions

1. Clone the repository:

    git clone: https://github.com/vishruth1997/nest-kudos-api

2. Navigate to the project folder:

    cd nest-kudos-api

3. Install dependencies:

    npm install

4. Start the server:

    npm start

5. Open GraphQL Playground at:

    http://localhost:4000/graphql

4. Project Structure

The codebase is organized into the following files:

- • index.ts – Entry point for setting up Apollo Server, Express, and WebSockets
- • schema.ts – GraphQL type definitions with documented fields
- • resolvers.ts – Query, mutation, and subscription logic with role-based access control
- • mockData.ts – In-memory mock data for users and recognitions
- • auth.ts – Utility for simulating authentication and fetching the current user
- • types.ts – Enum definitions for roles and visibility levels

5. API Usage Examples

• Query users:

```
query {
  users {
    id
    name
    role
    team
  }
}
```

• Query recognitions:

```
query {
  recognitions(userId: "2") {
    message
    emoji
    visibility
  }
}
```

• Send recognition:

```
mutation {
  sendRecognition(
    senderId: "1"
    recipientId: "2"
    message: "Great work!"
    emoji: "☺"
    visibility: PUBLIC
  ) {
    id
    message
  }
}
```

• Subscribe to new recognitions:

```
subscription {
  newRecognition {
    message
    emoji
    sender { name }
    recipient { name }
  }
}
```

6. Developer Notes

- Use `getCurrentUser(userId)` to simulate user-based access in queries and mutations.

- All role-based access control is enforced within resolvers.

- You can extend the Recognition type with badges, reactions, and comments easily in schema.ts.

- Consider replacing in-memory data with a real database (e.g., PostgreSQL) for production.

- For production environments, implement real authentication (e.g., JWT or OAuth).