Custom Backend Integration Guide

The Code Comments application is designed to be backend-agnostic. While it comes with a default .NET implementation, you can implement your own backend server using any technology stack (Node.js, Python, Go, etc.) as long as it adheres to the standard REST API contract described in this guide.

Overview

The client application communicates with the backend using a standard RESTful API. When a user configures a project in the Manager application, they provide a **Server Base URL** (e.g., `https://api.myserver.com`). The client then appends standard paths to this base URL to perform operations.

Base URL Structure

All project-specific endpoints are relative to:
`{ServerBaseUrl}/api/v1/project/{projectId}`

Global endpoints (like categories) are relative to:
`{ServerBaseUrl}/api/v1`

Authentication

The client supports JWT-based authentication. If an authentication token is present (passed from the Manager app), it will be included in the `Authorization` header of every request.

```
Authorization: Bearer <your_jwt_token>
```

Your backend should validate this token if security is required.

CORS Requirements

Since the client application runs in a browser, your backend **must** support Cross-Origin Resource Sharing (CORS). You should allow requests from the domain where the client application is hosted.

Required headers:

- `Access-Control-Allow-Origin`： `*` or specific client domain
- `Access-Control-Allow-Methods`： `GET, POST, PUT, DELETE, OPTIONS`
- `Access-Control-Allow-Headers`： `Content-Type, Authorization`

API Reference

1. Get All Comments

Fetches all comments for a specific project.

- **Endpoint**: `GET /api/v1/project/{projectId}/comments`
- **Response**: Array of `CommentDto`

2. Create Comment

Creates a new root-level comment.

- **Endpoint**: `POST /api/v1/project/{projectId}/comments`
- **Body**: `CommentDto` (without ID)
- **Response**: `CommentDto` (with assigned ID)

3. Update Comment

Updates the content or properties of an existing comment.

- **Endpoint**: `PUT /api/v1/project/{projectId}/comments/{id}`
- **Body**: `CommentDto`
- **Response**: `CommentDto`

4. Delete Comment

Deletes a comment. If the comment has replies, the backend should handle them (e.g., cascade delete or mark as deleted).

- **Endpoint**: `DELETE /api/v1/project/{projectId}/comments/{id}`
- **Response**: `200 OK` or `204 No Content`

5. Reply to Comment

Creates a reply to an existing comment.

- **Endpoint**: `POST /api/v1/project/{projectId}/comments/{parentCommentId}/reply`
- **Body**: `CommentDto` (The reply object)
- **Response**: `CommentDto` (The created reply)

6. Get Comment Thread

Fetches a specific comment thread (root comment and all its descendants).

- **Endpoint**: `GET /api/v1/project/{projectId}/comments/{rootCommentId}/thread`
- **Response**: Array of `CommentDto`

7. Get Categories

Fetches the list of available comment categories.

- **Endpoint**: `GET /api/v1/category`
- **Response**: Array of `CategoryDto`

Data Models

CommentDto

```
interface CommentDto {
  id: string | null;           // Null when creating, UUID/String when reading
  categoryId: string | null;   // ID of the category
  type: CommentType;           // Type of comment
  content: string;             // Markdown content

  // Location Information
  location: LocationDto;

  // Threading
  rootCommentId: string | null;   // ID of the top-level comment
  parentCommentId: string | null; // ID of the immediate parent
  replies?: CommentDto[];         // Nested replies (optional)
}
```

LocationDto

Defines where the comment is located.

```
interface LocationDto {
  id: string | null;
  type: CommentType;
  filePath: string;           // Relative path to the file (e.g., "src/app.ts")

  // For Singleline comments
  lineNumber?: number;        // 1-based line number

  // For Multiline comments
  startLineNumber?: number;   // 1-based start line
  endLineNumber?: number;     // 1-based end line

  // For File/Project comments
  description?: string;       // Optional description
}
```

CategoryDto

```ts
interface CategoryDto {
  id: string;
  label: string;
  description: string;
}
```

CommentType (Enum)

```ts
enum CommentType {
  Singleline = "Singleline", // Comment on a specific line
  Multiline = "Multiline",   // Comment on a range of lines
  File = "File",             // Comment on the file itself
  Project = "Project"        // Global comment on the project
}
```

Example JSON Payload (Create Comment)

```json
{
  "id": null,
  "categoryId": "bug",
  "type": "Singleline",
  "content": "This function needs refactoring.",
  "location": {
    "id": null,
    "type": "Singleline",
    "filePath": "src/utils/math.ts",
    "lineNumber": 42
  },
  "rootCommentId": null,
  "parentCommentId": null
}
```