Custom Repository Source - Developer/Admin Guide

The Code Comments application provides you with the flexibility to integrate custom source code repositories. This guide outlines the steps required to implement and integrate a custom source provider.

Overview

By default, the application supports GitHub (using the GitHub API) and the Single File Source Provider. However, you can extend this functionality by implementing and integrating your own custom source provider.

There are two main integration options:

1. **Single File Source Provider**: Use the predefined provider with your custom static JSON file that contains repository structure and file content URLs
2. **New Provider Integration**: Implement a completely new source provider type like GitLab or Bitbucket

Option 1: Single File Source Provider

The Single File Source Provider uses a single static JSON file containing:

- Complete repository tree structure
- URLs to individual file contents

This approach combines the simplicity of a static file with the scalability of on-demand file loading.

Implementation Requirements

Your static JSON file should be accessible via HTTP/HTTPS at a URL you provide during project setup.

Example URL format: `https://your-server.com/projects/project-42/content.json`

JSON File Structure

The `content.json` file must contain:

```json
{
  "tree": [
    {
      "name": "src",
      "path": "src",
      "type": "folder",
      "children": [
        {
          "name": "app.ts",
          "path": "src/app.ts",
          "type": "file",
          "fileUrl": "https://your-server.com/projects/project-42/files/src/app.ts",
          "children": [],
          "isExpanded": false
        },
        {
          "name": "logo.png",
          "path": "src/logo.png",
          "type": "file",
          "fileUrl": "https://your-server.com/projects/project-42/files/src/logo.png",
          "previewUrl": "https://your-server.com/projects/project-42/files/src/logo.png?preview=
          "children": [],
          "isExpanded": false
        }
      ],
      "isExpanded": false
    }
  ]
}
```

Tree Node Structure

```
interface TreeNode {
  name: string;           // File or folder name
  path: string;           // Relative path from repository root
  type: "file" | "folder"; // Node type
  fileUrl?: string;       // URL to fetch file content (required for files)
  previewUrl?: string;    // Optional optimized URL for preview (thumbnails, etc.)
  children: TreeNode[];   // Child nodes (empty array for files)
  isExpanded: boolean;    // UI state (default: false)
}
```

File Content Response

When the application requests a file from the `fileUrl` , your server should return:

```
{
  "displayType": "text",
  "content": "// Your file content here\nconst app = 'Hello';",
  "downloadUrl": "https://your-server.com/projects/project-42/files/src/app.ts",
  "previewUrl": null,
  "fileName": "app.ts"
}
```

ProcessedFile Interface

```
interface ProcessedFile {
  displayType: "text" | "image" | "pdf" | "binary"; // How to display the file
  content: string | null;      // Text content (for text files, base64 for images)
  downloadUrl: string | null;  // URL to download full-size file
  previewUrl: string | null;   // URL for optimized preview (smaller images, PDF thumbnails)
  fileName: string;            // Name of the file
}
```

downloadUrl vs previewUrl

- **`downloadUrl`** : Points to the full-size file (e.g., 4K image, complete PDF document)
- **`previewUrl`** : Points to an optimized version for preview purposes (e.g., 800px thumbnail, compressed PDF)

For text files, both URLs are typically the same or `previewUrl` can be `null` .

Authentication

The initial request to `content.json` and subsequent file requests should support optional authentication via `Authorization` header:

```
Authorization: Bearer {authToken}
```

Error Handling

Your server should return appropriate HTTP status codes:

Status Code	Meaning	Client Behavior
`200 OK`	Success	Process the response
`401 Unauthorized`	Invalid or missing auth token	Prompt user for credentials
`403 Forbidden`	Access denied	Show error, don't retry
`404 Not Found`	File/resource doesn't exist	Show "file not found" message
`429 Too Many Requests`	Rate limit exceeded	Wait and retry with backoff
`500 Internal Server Error`	Server error	Show error, allow manual retry
`503 Service Unavailable`	Temporary unavailability	Retry after delay

Example Implementation

You can generate your `content.json` file using any programming language. Here's a conceptual example:

```
# Generate tree structure
find ./your-project -type f -o -type d > files.txt

# Create JSON with file URLs pointing to your static file server
# Each file should be accessible at:
# https://your-server.com/projects/project-42/files/{path}
```

Option 2: New Provider Integration

If you need to integrate a completely new provider type (e.g., GitLab, Bitbucket, Azure DevOps), follow these steps.

SourceProvider Interface

All source providers must implement the `SourceProvider` interface:

SourceProvider Interface

All source providers must implement the `SourceProvider` interface:

```
export interface SourceProvider {
  /**
   * Fetches the complete file tree/directory structure from the source
   * @param repositoryUrl - The URL or identifier of the repository/source
   * @param branch - The branch, version, or snapshot identifier
   * @param authToken - Optional authentication token (GitHub PAT, API key, etc.)
   * @returns Promise resolving to an array of TreeNode representing the file structure
   */
  getRepositoryTree(repositoryUrl: string, branch: string, authToken?: string): Promise<TreeNod

  /**
   * Fetches and processes a single file from the source
   * @param repositoryUrl - The URL or identifier of the repository/source
   * @param branch - The branch, version, or snapshot identifier
   * @param filePath - The relative path to the file within the repository
   * @param authToken - Optional authentication token (GitHub PAT, API key, etc.)
   * @returns Promise resolving to a ProcessedFile containing the file content and metadata
   */
  fetchProcessedFile(
    repositoryUrl: string,
    branch: string,
    filePath: string,
    authToken?: string
  ): Promise<ProcessedFile>;
}
```

Step 1: Update the RepositoryType Enum

Add your new type to the enum in these locations:

Client: `client/src/types/shared/RepositoryType.ts`

Manager: `manager/shared/types/RepositoryType.ts`

Server: `server/Types/Enums/RepositoryType.cs`

```
// TypeScript
export enum RepositoryType {
  github = "github",
  singleFile = "singleFile",
  gitlab = "gitlab",  // Add your new type
}
```

```
// C#
namespace server.Types.Enums
{
  public enum RepositoryType
  {
    github,
    singleFile,
    gitlab  // Add your new type
  }
}
```

Step 2: Create Your Provider Class

Location: `client/src/services/providers/GitlabSourceProvider.ts`

```
import type { SourceProvider } from "../../types/interfaces/SourceProvider";
import type { TreeNode } from "../../types/domain/TreeContent";
import type { ProcessedFile } from "../../types/domain/FileContent";

export class GitlabSourceProvider implements SourceProvider {
  async getRepositoryTree(repositoryUrl: string, branch: string, authToken?: string): Promise<T
    // Fetch tree data from GitLab API
    // Transform it to TreeNode[]
    // Return the tree
  }

  async fetchProcessedFile(
    repositoryUrl: string,
    branch: string,
    filePath: string,
    authToken?: string
  ): Promise<ProcessedFile> {
    // Fetch file content from GitLab API
    // Determine the displayType based on file extension
    // Process content appropriately (decode base64, etc.)
    // Return ProcessedFile
  }
}
```

Step 3: Register the New Provider

Location: `client/src/services/sourceProviderFactory.ts`

```
import { GithubSourceProvider } from "./providers/GithubSourceProvider";
import { SingleFileSourceProvider } from "./providers/SingleFileSourceProvider";
import { GitlabSourceProvider } from "./providers/GitlabSourceProvider";

const createProvider = (repositoryType: RepositoryType): SourceProvider => {
  switch (repositoryType) {
    case RepositoryType.github:
      return new GithubSourceProvider();
    case RepositoryType.singleFile:
      return new SingleFileSourceProvider();
    case RepositoryType.gitlab:  // Add this
      return new GitlabSourceProvider();
    default:
      throw new Error(`Unsupported repository type: ${repositoryType}`);
  }
};
```

Step 4: Add UI Support

Update `manager/app/components/ProjectForm.vue` to include your new provider in the UI selector:

```
const repositoryTypeOptions = [
  { value: RepositoryType.github, label: "GitHub", icon: "mdi:github" },
  { value: RepositoryType.singleFile, label: "Static File", icon: "mdi:file-code" },
  { value: RepositoryType.gitlab, label: "GitLab", icon: "mdi:gitlab" },  // Add this
];
```

Step 5: Add Translations

Update `manager/i18n/locales/en.json`:

```
{
  "projectForm": {
    "gitlabRepoUrlPlaceholder": "Enter GitLab repository URL"
  }
}
```