Custom Source - Developer/Admin Guide

TODO: in the thesis text I should also state the benefits of having a custom HTTP API source (custom authorization, self-hosting, etc.)

Overview

This document provides guidelines for developers and administrators on how to add a custom source provider to the application.

There will be two main parts:

1. **Implementing a Custom API** - Creating an HTTP API that exposes the required endpoints for the application to fetch the repository tree and file contents.
2. **New Provider Integration** - Integrating the custom HTTP API into the application by creating a new source provider type.

Option 1: Implementing a Custom API

Your HTTP API needs to provide **two endpoints**:

1. Tree Endpoint - Directory Structure

Endpoint: `GET /tree?branch={branch}`

Returns the complete directory structure of your code repository.

Response Format:

```
[
  {
    "name": "folder-name",
    "path": "whole/path/to/folder",
    "type": "folder",
    "children": [
      {
        "name": "file.ts",
        "path": "whole/path/to/folder/file.ts",
        "type": "file" | "folder",
        "children": [],
        "isExpanded": false
      }
    ],
    "isExpanded": false
  }
]
```

Field Descriptions:

```
export interface TreeNode {
         name: string; // File or folder name
         path: string; // Relative path from repository root
         type: TreeNodeType; // "file" | "folder"
         children: TreeNode[]; // Child nodes (for folders)
         isExpanded: boolean; // UI state for folder expansion
}
```

2. File Endpoint - File Contents

Endpoint: `GET /file?branch={branch}&path={filePath}`

Returns the content and metadata for a specific file.

Response Format:

```json
{
  "displayType": "text",
  "content": "The actual file content as a string",
  "downloadUrl": "https://example.com/download/path/to/file.ts",
  "previewUrl": null,
  "fileName": "file.ts"
}
```

Authentication

Both endpoints should support optional authentication via an `Authorization` header with a Bearer token:

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

Field Descriptions:

TODO: the display type could be improved to the classic "mime type" format, not the custom strings I use

```typescript
export interface ProcessedFile {
          displayType: FileDisplayType; // "text" | "image" | "pdf" | "binary"
          content: string | null; // Text content (for text files)
          downloadUrl: string | null; // URL to download the file
          previewUrl: string | null; // URL for previewing (images, PDFs)
          fileName: string; // Name of the file
}
```

Option 2: New Provider Integration

All source providers must implement the `ISourceProvider` interface:

```
export interface ISourceProvider {
  getRepositoryTree(repositoryUrl: string, branch: string, authToken?: string): Promise<TreeNod

          fetchProcessedFile(
                    repositoryUrl: string,
                    branch: string,
                    filePath: string,
                    authToken?: string
          ): Promise<ProcessedFile>;
}
```

To add a completely new provider type like for Gitlab or Bitbucket, follow these steps:

Step 1: Update the RepositoryType Enum

Add your new type to both client and manager:

Client: `client/src/types/enums/RepositoryType.ts`
Manager: `manager/shared/types/RepositoryType.ts`

```
export enum RepositoryType {
          github = "github",
          httpApi = "httpApi",
          myCustomProvider = "myCustomProvider", // Add this
}
```

Step 2: Create Your Provider Class

```
// MyCustomSourceProvider.ts
import type { ISourceProvider } from "../../types/sourceProviders/ISourceProvider";
import type { TreeNode } from "../../types/github/githubTree";
import type { ProcessedFile } from "../../types/github/githubFile";

export class MyCustomSourceProvider implements ISourceProvider {
        async getRepositoryTree(repositoryUrl: string, branch: string, authToken?: string): Pr
                // Fetch tree data from your source
                // Transform it to TreeNode[]
                // Return the tree
        }

        async fetchProcessedFile(
                repositoryUrl: string,
                branch: string,
                filePath: string,
                authToken?: string
        ): Promise<ProcessedFile> {
                // Fetch file content from your source
                // Determine the displayType
                // Process content appropriately
                // Return ProcessedFile
        }
}
```

Step 3: Register the New Provider

```
import { MyCustomSourceProvider } from "./providers/MyCustomSourceProvider";

const createProvider = (repositoryType: RepositoryType): ISourceProvider => {
        switch (repositoryType) {
                case RepositoryType.github:
                        return new GithubSourceProvider();
                case RepositoryType.httpApi:
                        return new HttpApiSourceProvider();
                case RepositoryType.myCustomProvider: // Add this
                        return new MyCustomSourceProvider();
                default:
                        throw new Error(`Unsupported repository type: ${repositoryType}`);
        }
};
```

Step 4: Add Translations

(This could be optional, there could be generic text used instead)

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

```
{
        "projectForm": {
                "myCustomProviderUrlPlaceholder": "Enter your custom provider URL"
        }
}
```