```typescript
import { Injectable } from '@angular/core';
import { DataRetrieverService } from '../shared/services/data-retriever.service';
import { CoreService } from '../shared/services/core.service';
import { Subject } from 'rxjs/Subject';
import { Project } from './Models/project.model';
import { HttpErrorResponse } from '@angular/common/http';
import { ProjectFile } from './Models/project-file.model';
import { Folder } from './Models/folder.model';
import { FolderProjectFile } from './Models/folder-project-file.model';

@Injectable()
export class ProjectsService {
  private _fullProjData: any[];
  private _projects: Project[];
  getProjectsAsync = new Subject<Project[]>();

  constructor(
    private _dataRetrieverService: DataRetrieverService,
    private _coreService: CoreService
  ) {
    this._projects = null;

    this._loadData();
    this._coreService.langUpdated.subscribe(
      (lang: string) => {
        this._projects = this._setProjects(this._fullProjData, lang);
        this.getProjectsAsync.next(this._projects);
      }
    );
  }

  // get Project data from the server
  private _loadData() {
    this._dataRetrieverService.importProjectData().subscribe(
      (rawProjects) => {
        this._fullProjData = rawProjects;
        this._projects = this._setProjects(rawProjects, this._coreService.getLang());
        this.getProjectsAsync.next(this._projects);
      },
      (err: HttpErrorResponse) => {
        if (err.error instanceof Error) {
          console.log('Client-side error occurred.');
        } else {
          console.log('Server-side error occurred.');
        }
      }
    );
  }

  // populate projects from complete data (all languages included)
  private _setProjects(fullData: any, lang: string): Project[] {
    const projects: Project[] = [];
    let project: Project, index = 0, projectFiles: ProjectFile[];

    for (const projectRef of fullData) {
      project = new Project(
        projectRef['id'],
        projectRef['extTableName'],
        projectRef['occId'],
        lang === 'eng' ? projectRef['title'] : projectRef['titoloIta'],
        lang === 'eng' ? projectRef['details'] : projectRef['dettagli'],
        projectRef['attachedFile'],
        projectFiles = this._setProjFiles(index, lang),
        this._setFoldersArray(projectFiles)
      );
      index ++;
      projects.push(project);
    }

    return projects;
  }

  getProjects(): Project[] {
```

```
    return this._projects;
  }

  isProjects(): boolean {
    return this._projects !== null;
  }

  private _setProjFiles(projectPos: number, lang: string): ProjectFile[] {
    let projectFile: ProjectFile;
    let projectFiles: ProjectFile[] = [];

    for (const projFileRef of this._fullProjData[projectPos].projFiles) {
      projectFile = new ProjectFile(
        projFileRef['id'],
        lang === 'eng' ? projFileRef['title'] : projFileRef['titoloIta'],
        projFileRef['attachedFile'],
        projFileRef.attachedFile.split('/').length,
      );
      projectFiles.push(projectFile);
    }

    projectFiles = this._sortProjectFiles(projectFiles);
    return projectFiles;
  }

  // populate a tree-like array to keep track of the folder structure
  private _setFoldersArray(projectFiles: ProjectFile[]) {
    let pathLength: number, currentPathArray: string[], folderLevel: number,
        folderIndex = 0;
    const folders = [];
    for (const projFile of projectFiles) {
      currentPathArray = projFile.attachedFile.split('/');
      pathLength = currentPathArray.length;
      currentPathArray.splice(currentPathArray.length - 1, 1);
      currentPathArray.splice(0, 2);
      folderLevel = 0;
      for (const folderName of currentPathArray) {
        if (!this._isExistingFolderName(folderName, projFile.attachedFile,
            folders[folderLevel], folderLevel + 2)) {
          if (folderLevel === folders.length) {
            folders[folderLevel] =
              [new Folder(folderName, this.getFilelessPath(projFile.attachedFile),
                  pathLength,
                this._setFiles(projectFiles, projFile.attachedFile, folderName,
                    folderLevel))];
          } else {
            folders[folderLevel].push(
              new Folder(folderName, projFile.attachedFile, pathLength,
                this._setFiles(projectFiles, projFile.attachedFile, folderName,
                    folderLevel))
            );
          }
        }
        folderLevel += 1;
      }
      folderIndex += 1;
    }
    return folders;
  }

  private _setFiles(
    projectFiles: ProjectFile[], currentPath: string, folderName: string,
        folderLevel: number): FolderProjectFile[] {
    let splitPath: string[];
    const files: FolderProjectFile[] = [];

    for (const projFile of projectFiles) {
      splitPath = projFile.attachedFile.split('/');
      splitPath.splice(splitPath.length - 1, 1);
      splitPath.splice(0, 2);
      if (splitPath[folderLevel] === folderName &&
        folderLevel === (splitPath.length - 1) &&
        this.isPathwayEqualUpToIndex(currentPath, projFile.attachedFile, folderLevel
```

```
            + 2)) {
141           files.push(new FolderProjectFile(projFile.id, projFile.title,
                projFile.attachedFile));
142         }
143       }
144       return files;
145     }
146
147     // check if the given folder name already exists in the array
148     private _isExistingFolderName(
149       folderName: string, folderPath: string, foldersArray: Folder[], horizontalIndex:
            number): boolean {
150       if (!foldersArray) { return false; }
151       for (const folder of foldersArray) {
152         if (folder.name === folderName &&
153           this.isPathwayEqualUpToIndex(folderPath, folder.folderPath,
                horizontalIndex)) {
154           return true;
155         }
156       }
157       return false;
158     }
159
160     // cycle the longest path horizontally while running a sorting algorithm
161     private _sortProjectFiles(projectFiles: ProjectFile[]): ProjectFile[] {
162       let horizontalIndex = 0;
163       const longestPath = this._getLongestPath(projectFiles);
164       while (horizontalIndex < longestPath) {
165         projectFiles = this._innerSortProjectFiles(projectFiles, horizontalIndex);
166         horizontalIndex += 1;
167       }
168       return projectFiles;
169     }
170
171     // get the longest existing path out of a projectFiles array
172     private _getLongestPath(projectFiles: ProjectFile[]): number {
173       let longestPath = 0;
174       for (const projectFile of projectFiles) {
175         if (projectFile.pathLength > longestPath) {
176           longestPath = projectFile.pathLength;
177         }
178       }
179       return longestPath;
180     }
181
182     // sort project file paths by folders
183     private _innerSortProjectFiles(projectFiles: ProjectFile[], horizontalIndex:
            number): ProjectFile[] {
184       return projectFiles.sort((a: ProjectFile, b: ProjectFile) => {
185         const aSplit = a.attachedFile.split('/'), bSplit = b.attachedFile.split('/');
186         let i = 0, aString = '', bString = '';
187         while (i < aSplit.length - 1) { aString += aSplit[i] + '/'; i++; }
188         i = 0;
189         while (i < bSplit.length - 1) { bString += bSplit[i] + '/'; i++; }
190         if ((aSplit[horizontalIndex] > bSplit[horizontalIndex] ||
191             aSplit.length < bSplit.length) &&
192           this.isPathwayEqualUpToIndex(aString, bString, horizontalIndex)) {
193           return 1;
194         } else if ((aSplit[horizontalIndex] < bSplit[horizontalIndex] ||
195           aSplit.length > bSplit.length) &&
196           !this.isPathwayEqualUpToIndex(aString, bString, horizontalIndex)) {
197           return -1;
198         } else {
199           return 0;
200         }
201       });
202     }
203
204     // return path without the file name at the end
205     getFilelessPath(path: string, adaptToRecursion?: boolean): string {
206       const pathSplit = path.split('/');
207       if (adaptToRecursion) {
208         pathSplit.splice(0, 2);
```

```
209       }
210
211       if (pathSplit[pathSplit.length - 1].indexOf('.') > -1) {
212         pathSplit.splice(pathSplit.length - 1, 1);
213       }
214       let i = 0, filelessString = '';
215       while (i < pathSplit.length) { filelessString += pathSplit[i] + '/'; i++; }
216       filelessString = filelessString.substring(0, filelessString.length - 1);
217       return filelessString;
218     }
219
220     // check if two pathways have the same root up to a certain index
221     isPathwayEqualUpToIndex(aPath: string, bPath: string, currentHorizontalIndex:
          number) {
222       const aSplit = aPath.split('/'), bSplit = bPath.split('/');
223       let i = 0;
224       while (aSplit[i] !== undefined &&
225         bSplit[i] !== undefined &&
226         (aSplit[i] === bSplit[i]) &&
227       (i < currentHorizontalIndex)) { i += 1; }
228       if (i === currentHorizontalIndex) {
229         return true;
230       } else {
231         return false;
232       }
233     }
234
235   }
236
```