```typescript
import { Injectable } from '@angular/core';
import { Occurrence } from '../models/occurrence.model';
import { Certificate } from '../models/certificate.model';
import { Image } from '../models/image.model';
import 'rxjs/add/operator/map';
import { Category } from '../models/category.model';
import { ImageSet } from '../shared-modal/image-set.model';
import { Residence } from '../../timeline/residence.model';
import { DataRetrieverService } from './data-retriever.service';
import { HttpErrorResponse } from '@angular/common/http';
import { CoreService } from './core.service';
import { Subject } from 'rxjs/Subject';

@Injectable()

export class CvDataHandlerService {
  private _fullCvData: any[];
  private _categories: Category[];
  private _education: Category;
  private _workExperience: Category;
  private _volunteering: Category;
  private _hobbies: Category;
  private _residences: Residence[];
  getCategoriesAsync = new Subject<Category[]>();
  getResidencesAsync = new Subject<Residence[]>();

  // subscribe to language change to load cv info in a different language
  // load data from the server only once
  constructor(
    private _dataRetrieverService: DataRetrieverService,
    private _coreService: CoreService
  ) {
      this._categories = null;
      this._residences = null;

      this._loadData();
      this._coreService.langUpdated.subscribe(
      (lang: string) => {
        this._categories = this._setCategories(lang);
        this._residences = this._setResidences(lang);
        this.getCategoriesAsync.next(this._categories);
        this.getResidencesAsync.next(this._residences);
      }
    );
  }

  // get cv data from the server
  private _loadData() {
    this._dataRetrieverService.importCvData().subscribe(
      (categories) => {
        this._fullCvData = categories;
        this._categories = this._setCategories(this._coreService.getLang());
        this._residences = this._setResidences(this._coreService.getLang());
        this.getCategoriesAsync.next(this._categories);
        this.getResidencesAsync.next(this._residences);
      },
      (err: HttpErrorResponse) => {
        if (err.error instanceof Error) {
          console.log('Client-side error occurred.');
        } else {
          console.log('Server-side error occurred.');
        }
      }
    );
  }

  // return whether or not categories have been loaded from the server yet
  isCategories(): boolean {
    return this._categories !== null;
  }

  // return whether or not residences (nations I resided in) have been loaded from
      the server yet
  isResidences(): boolean {
    return this._residences !== null;
  }

  // populate categories from complete data (all languages included)
  private _setCategories(lang: string): Category[] {
    const categories = [];

    this._setEducation(lang);
    this._setWorkExperience(lang);
    this._setVolunteering(lang);
    this._setHobbies(lang);

    categories.push(this._education);
    categories.push(this._workExperience);
    categories.push(this._volunteering);
    categories.push(this._hobbies);

    return categories;
  }

  getCategories(): Category[] {
    return this._categories;
  }

  getCategoryByName (categoryName): Category {
    for (const categoryRef of this._categories) {
      if (categoryRef.categoryName === categoryName) {
        return categoryRef;
      }
    }
    return null;
  }

  getOccurrenceByIds(categoryName, occurrenceId): Occurrence {
    for (const category of this._categories) {
      if (category.categoryName === categoryName) {
        for (const occurrenceRef of category.occurrences) {
          if (occurrenceRef.id === occurrenceId) {
            return occurrenceRef;
          }
        }
        break;
      }
    }
    return null;
  }

  getResidenceById (residenceId): Residence {
    for (const residenceRef of this._residences) {
      if (residenceRef.id === residenceId) {
        return residenceRef;
      }
    }
    return null;
  }

  getResidences(): Residence[] {
    return this._residences;
  }

  // populate each single category ----
  private _setEducation(lang) {
    this._education = new Category(
      'education',
      lang === 'eng' ? 'education and training' : 'istruzione e formazione',
      this._setOccurrences('education', lang)
    );
  }

  private _setWorkExperience(lang) {
    this._workExperience = new Category(
      'workExperience',
      lang === 'eng' ? 'work experience' : 'esperienza lavorativa',
      this._setOccurrences('workExperience', lang)
    );
  }

  private _setVolunteering(lang) {
    this._volunteering = new Category(
      'volunteering',
      lang === 'eng' ? 'volunteering' : 'volontariato',
      this._setOccurrences('volunteering', lang)
    );
  }

  private _setHobbies(lang) {
    this._hobbies = new Category(
      'hobbies',
      lang === 'eng' ? 'hobbies' : 'hobby',
      this._setOccurrences('hobbies', lang)
    );
  }

  // --------

  // populate cv occurrences of a category
  private _setOccurrences(categoryName: string, lang: string) {
    const occurrences: Occurrence[] = [];
    let i = 0;

    for (const occurrence of this._fullCvData[categoryName]) {
      const localOccurrence: Occurrence = new Occurrence(
        occurrence.id,
        this._strToDate(occurrence.startDate),
        this._strToDate(occurrence.endDate),
        occurrence.attachedFile,
        lang === 'eng' ? occurrence.title : occurrence.titoloIta,
        occurrence.score,
        occurrence.company,
        lang === 'eng' ? occurrence.location : occurrence.luogo,
        lang === 'eng' ? occurrence.description : occurrence.descrizione,
        occurrence.addToCv,
        this._setCertificates(categoryName, i, lang),
        this._setImages(categoryName, i)
      );
      i++;
      occurrences.push(localOccurrence);
    }

    return occurrences;
  }

  // populate certificates of an occurrence
  private _setCertificates(categoryName: string, occurrenceId: number, lang:
      string): Certificate[] {
    const certificates: Certificate[] = [];

    if (this._fullCvData[categoryName][occurrenceId].certificates !== null) {
      for (const forCertificate of
          this._fullCvData[categoryName][occurrenceId].certificates) {
        const certificate = new Certificate(
          forCertificate.id,
          forCertificate.attachedFile,
          lang === 'eng' ? forCertificate['enDesc'] : forCertificate['itDesc'],
          forCertificate.score
        );
        certificates.push(certificate);
      }
    }
    return certificates;
  }

  // populate images for an occurrence
  private _setImages(categoryName: string, occurrenceId: number): Image[] {
    const images: Image[] = [];

    if (this._fullCvData[categoryName][occurrenceId].images !== null) {
      for (const image of this._fullCvData[categoryName][occurrenceId].images) {
        images.push(image);
      }
    }

    return images;
  }

  // populate residences (places I resided in) from complete data (all languages
      included)
  private _setResidences(lang: string): Residence[] {
    let residences;

    residences = this._innerSetResidences(lang);
    return residences;
  }

  // method shared setResidences and _reloadResidences to populare the residences
      array
  private _innerSetResidences(lang: string) {
    let residence: Residence;

    const residences = [];
    for (const residenceRef of this._fullCvData['residences']) {
      residence = new Residence(
        residenceRef['id'],
        lang === 'eng' ?  residenceRef['nation'] : residenceRef['nazione'],
        this._strToDate(residenceRef['startDate']),
        this._strToDate(residenceRef['endDate'])
      );
      residences.push(residence);
    }
    return residences;
  }

  // transforms a given string date into Date type (year-month-day format)
  private _strToDate(strDate: string): Date {
    const splitStringArray = strDate.split('-');
    return new Date(+splitStringArray[0], +splitStringArray[1] - 1,
        +splitStringArray[2] + 1);
  }

  // returns images and occurrence description for a given Category/Occurrence pair
  searchImageSet(categoryId: string, occurrenceId: string): ImageSet {
    const imageSet = new ImageSet('', []);

    for (const category of this._categories) {
      if (category.categoryName === categoryId) {
        for (const occurrence of category.occurrences) {
          if (occurrence.id === occurrenceId) {
            imageSet.desc = occurrence.title;
            imageSet.images = occurrence.images;
            break;
          }
        }
        break;
      }
    }
    return imageSet;
  }

}

```