```typescript
import { Injectable } from '@angular/core';
import { Category } from '../shared/models/category.model';
import { Subject } from 'rxjs/Subject';
import { CoreService } from '../shared/services/core.service';
import { Residence } from './residence.model';
import { Occurrence } from '../shared/models/occurrence.model';

@Injectable()
export class TimelineService {
  private _dataTabToCateg;
  private _minDate = new Date(2008, 0, 1);
  private _maxDate = new Date();
  private _absMinDate = new Date(1985, 1, 21);
  private _absMaxDate = new Date();
  private _catCount: number; // total number of categories ()
  clickedItemEvent = new Subject<any>();
  minDateChanged = new Subject<Date>();
  maxDateChanged = new Subject<Date>();
  chartReady = new Subject<boolean>();

  constructor(private coreService: CoreService) { }

  getDataTabToCateg() {
    return this._dataTabToCateg;
  }

  getMinDate() {
    return this._minDate;
  }

  getMaxDate() {
    return this._maxDate;
  }

  // absolute minimum and maximum Dates acceptable
  getAbsMinDate() {
    return this._absMinDate;
  }

  getAbsMaxDate() {
    return this._absMaxDate;
  }

  setMinDate(minDate: Date) {
    if (minDate >= this._absMinDate && minDate <= this._absMaxDate) {
      this._minDate = minDate;
    } else if (minDate < this._absMinDate) {
      this._minDate = this._absMinDate;
    } else {
      this._minDate = new Date(this._absMaxDate.getFullYear() , 0, 1);
    }
    this.minDateChanged.next(this._minDate);
  }

  setMaxDate(maxDate: Date) {
    if (maxDate >= this._absMinDate && maxDate <= this._absMaxDate) {
      this._maxDate = maxDate;
    } else if (maxDate < this._absMinDate) {
      this._maxDate = new Date(1985, 11, 31);
    } else {
      this._maxDate = this._absMaxDate;
    }
    this.minDateChanged.next(this._minDate);
  }
  // ----------------

  setClickedItemOccurrence(categoryName: string, occurrenceRef: Occurrence) {
    let clickedItem;

    clickedItem = {
      categoryName: categoryName,
      occurrenceRef: occurrenceRef
    };
    this.clickedItemEvent.next(clickedItem);
  }

  setClickedItemResidence(residenceRef: Residence) {
    this.clickedItemEvent.next({
      categoryName: 'residence',
      residenceRef: residenceRef
    });
  }

  loadTimelineDataTable(categories: Category[], residences: any): any {
    let currentRow;
    const _dataTable = [];
    this._dataTabToCateg = [];
    let rowIndex = 0;

    this._catCount = categories.length;
    _dataTable.push(['categoryDesc', 'event', 'From', 'To']);

    for (const category of categories) {
      for (const occurrence of category.occurrences) {
        if (this._isWithinDatesRange(occurrence.startDate, occurrence.endDate)) {
          currentRow = [
            category.categoryDesc,
            occurrence.title,
            this._correctDate(occurrence.startDate),
            this._correctDate(occurrence.endDate)
          ];
          this._dataTabToCateg[rowIndex] = {
            categoryId: category.categoryName,
            occurrenceId: occurrence.id
          };
          _dataTable.push(currentRow);
          rowIndex++;
        }
      }
    }
    this._loadResidences(_dataTable, residences, rowIndex);

    return _dataTable;
  }

  private _loadResidences(dataTable, residences: Residence[], rowIndex) {
    let currentRow;

    for (const residence of residences) {
      if (this._isWithinDatesRange(residence.startDate, residence.endDate)) {
        currentRow = [
          this.coreService.getLang() === 'eng' ? 'Country of residence' : 'Nazione di residenza',
          residence.nation,
          this._correctDate(residence.startDate),
          this._correctDate(residence.endDate)
        ];
        this._dataTabToCateg[rowIndex] = {
          categoryId: 'residence',
          occurrenceId: residence.id
        };
        dataTable.push(currentRow);
        rowIndex++;
      }
    }
  }

  private _correctDate(date: Date): Date {
    if (date > this._maxDate) {
      date = this._maxDate;
    } else if (date < this._minDate) {
      date = this._minDate;
    }
    return date;
  }

```

```
    private _isWithinDatesRange(eventStartDate: Date, eventEndDate: Date): boolean {
      return !((eventStartDate > this._maxDate && eventEndDate > this._maxDate) ||
        (eventStartDate < this._minDate && eventEndDate < this._minDate));
    }

  }

```