import { Injectable } from '@angular/core';

import { ConstantsService } from './constants.service';

@Injectable()

export class LogicService {

sliderMaxValue = this.constantsService.getSliderMaxValue();

evaluationLevels = this.constantsService.getEvaluationLevels();

constructor(private constantsService: ConstantsService) { }

getCombinations(nodes) {

let pairwises = [];

for (let i = 0; i < nodes.length - 1; i++) {

for (let j = i + 1; j < nodes.length; j++) {

pairwises.push({ indexItem1: i, indexItem2: j })

}

}

return pairwises;

}

addComparisons(comparisons, nodesLenght) {

let updatedComparisons = [];

let originalIndex = 0;

for (let i = 0; i < nodesLenght - 1; i++) {

for (let j = i + 1; j < nodesLenght; j++) {

if (j == nodesLenght - 1) {

updatedComparisons.push(this.sliderMaxValue / 2);

}

else {

updatedComparisons.push(comparisons[originalIndex]);

originalIndex++;

}

}

}

return updatedComparisons;

}

removeComparisons(nodeToRemove, comparisons, nodesLenght) {

let updatedComparisons = [];

let originalIndex = 0;

for (let i = 0; i < nodesLenght; i++) {

for (let j = i + 1; j <= nodesLenght; j++) {

if (i != nodeToRemove && j != nodeToRemove) {

updatedComparisons.push(comparisons[originalIndex]);

}

originalIndex++;

}

}

return updatedComparisons;

}

convertSlider(sliderValue) {

let scaledSliderValue = (this.evaluationLevels \* sliderValue) / this.sliderMaxValue;

if (scaledSliderValue < 1) return 9;

else if (scaledSliderValue < 2) return 8;

else if (scaledSliderValue < 3) return 7;

else if (scaledSliderValue < 4) return 6;

else if (scaledSliderValue < 5) return 5;

else if (scaledSliderValue < 6) return 4;

else if (scaledSliderValue < 7) return 3;

else if (scaledSliderValue < 8) return 2;

else if (scaledSliderValue < 9) return 1;

else if (scaledSliderValue < 10) return 1 / 2;

else if (scaledSliderValue < 11) return 1 / 3;

else if (scaledSliderValue < 12) return 1 / 4;

else if (scaledSliderValue < 13) return 1 / 5;

else if (scaledSliderValue < 14) return 1 / 6;

else if (scaledSliderValue < 15) return 1 / 7;

else if (scaledSliderValue < 16) return 1 / 8;

else return 1 / 9;

}

convertSliderValue(sliderValue) {

let value = this.convertSlider(sliderValue);

if (value >= 1) {

return value;

}

else {

return 1/value;

}

}

prepareStackedChartData(bars, criteria, priorities) {

let chartData = {

bars: [],

criteria: [],

prioritiesMatrix: []

};

for (let i = 0; i < bars.length; i++) {

chartData.bars[i] = bars[i].name;

}

for (let i = 0; i < criteria.length; i++) {

chartData.criteria[i] = criteria[i].name;

}

chartData.prioritiesMatrix = priorities;

return chartData;

}

prepareNodesChartData(goal, criteria, alternatives, criteriaPriorities, priorities) {

let chartData = {

goal: String,

criteria: [],

alternatives: [],

criteriaPriorities: [],

prioritiesMatrix: []

};

chartData.goal = goal;

for (let i = 0; i < criteria.length; i++) {

chartData.criteria[i] = criteria[i].name;

}

for (let i = 0; i < alternatives.length; i++) {

chartData.alternatives[i] = alternatives[i].name;

}

chartData.criteriaPriorities = criteriaPriorities;

chartData.prioritiesMatrix = priorities;

return chartData;

}

prepareSimpleChartData(vertical, horizontal) {

let chartData = {

vertical: [],

horizontal: []

};

for (let i = 0; i < vertical.length; i++) {

chartData.vertical[i] = vertical[i].name;

}

chartData.horizontal = horizontal;

return chartData;

}

getEvaluatorsNames(masterName, evaluators) {

let evaluatorsNames = []

evaluatorsNames[0] = {name: masterName};

for(let i=0; i<evaluators.length; i++) {

evaluatorsNames[i+1] = {name: evaluators[i].name};

}

return evaluatorsNames;

}

}