package com.marketcast.compute.api.simplified;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.Map;  
  
public class GetDataViewRequestForQuestionListDataModel {  
  
  
  
  
  
 private String[] guids;  
 private String reportTypeName;  
 private String requestId;  
 private String[] questionsList;  
 private Map<String, List<String>> gridQuestionsDict;  
  
 public String[] getGuids() {  
 return guids;  
 }  
  
 public void setGuids(String[] guids) {  
 this.guids = guids;  
 }  
  
 public String getReportTypeName() {  
 return reportTypeName;  
 }  
  
 public void setReportTypeName(String reportTypeName) {  
 this.reportTypeName = reportTypeName;  
 }  
  
 public String getRequestId() {  
 return requestId;  
 }  
  
 public void setRequestId(String requestId) {  
 this.requestId = requestId;  
 }  
  
  
  
  
 public String[] getQuestionsList() {  
 return questionsList;  
 }  
  
 public void setQuestionsList(String[] questionsList) {  
 this.questionsList = questionsList;  
 }  
  
 public Map<String, List<String>> getGridQuestionsDict() {  
 return gridQuestionsDict;  
 }  
  
 public void setGridQuestionsDict(Map<String, List<String>> gridQuestionsDict) {  
 this.gridQuestionsDict = gridQuestionsDict;  
 }  
  
 @Override  
 public String toString() {  
 return "GetDataViewRequestForQuestionListDataModel{" +  
 "guids=" + Arrays.*toString*(guids) +  
 ", reportTypeName='" + reportTypeName + '\'' +  
 ", requestId='" + requestId + '\'' +  
 ", questionsList=" + Arrays.*toString*(questionsList) +  
 ", gridQuestionsDict=" + gridQuestionsDict +  
 '}';  
 }  
}

// @RequestMapping(value="/dataViewRequest", method=RequestMethod.POST, consumes = MediaType.APPLICATION\_JSON\_VALUE)  
// public void getMetricListAJAX1(@RequestBody GetDataViewRequestForQuestionListDataModel adm  
// ) throws Exception {  
// //this.tokenVerifier.tokenVerifier(authToken);  
// long logId = rid++;  
// logMethodEntry("getMetricListAJAX", logId, adm);  
// List<StatResponse> crosstabsResponseList = IntStream.range(0, adm.getQuestionsList().length).parallel().mapToObj(crossTabIndex -> {  
// StatResponse currentResponse = null;  
//  
// StatRequest statRequest = new StatRequest();  
// statRequest.setRequestId(adm.getRequestId());  
// statRequest.setGuids(adm.getGuids());  
// statRequest.setReportTypeName(adm.getReportTypeName());  
// if (adm.getReportTypeName().equals("walledGardens")) {  
// statRequest.setStatNames(new String[]{"General\_Sample" });  
// } else {  
// statRequest.setStatNames(new String[]{"Case\_Count" });  
// }  
// statRequest.setFields(new String[][]{{adm.getQuestionsList()[crossTabIndex]}});  
//  
//  
// ResultsFormat resultsFormat = new ResultsFormat();  
// resultsFormat.setRows(new String[]{adm.getQuestionsList()[crossTabIndex]});  
// statRequest.setFormat(resultsFormat);  
// try {  
// currentResponse = (new ReportAPI()).getFormattedStats(statRequest);  
// return currentResponse;  
// } catch (Exception e) {  
// throw new RuntimeException(e);  
// }  
//  
// // Call getFormattedStats to get the desired results  
// }).collect(Collectors.toList());  
// // Extract and return the required data from finalStatResponse  
// }  
  
 @RequestMapping(value = "/dataViewRequest", method = RequestMethod.*POST*, consumes = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public List<String> getMetricListAJAX1(@RequestBody GetDataViewRequestForQuestionListDataModel adm) throws Exception {  
 long logId = *rid*++;  
 logMethodEntry("getMetricListAJAX", logId, adm);  
  
 // Prepare the totalSampleN StatRequest  
 StatRequest totalSampleNRequest = createTotalSampleNRequest(adm);  
 StatResponse totalSampleNResponse = (new ReportAPI()).getFormattedStats(totalSampleNRequest);  
  
 // Prepare and execute parallel requests for each question  
 List<StatResponse> crosstabsResponseList = IntStream.*range*(0, adm.getQuestionsList().length)  
 .parallel()  
 .mapToObj(crossTabIndex -> {  
 StatRequest questionRequest = createQuestionRequest(adm, crossTabIndex);  
 try {  
 return (new ReportAPI()).getFormattedStats(questionRequest);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 })  
 .collect(Collectors.*toList*());  
  
 // Extract and return the required data from the responses  
 AtomicReference<List<String>> result = new AtomicReference<>(extractData(totalSampleNResponse, crosstabsResponseList));  
  
 return result.get();  
 }  
  
 private StatRequest createTotalSampleNRequest(GetDataViewRequestForQuestionListDataModel adm) {  
 StatRequest totalSampleNRequest = new StatRequest();  
 totalSampleNRequest.setRequestId(adm.getRequestId());  
 totalSampleNRequest.setGuids(adm.getGuids());  
 totalSampleNRequest.setReportTypeName(adm.getReportTypeName());  
 if (adm.getReportTypeName().equals("WalledGardens")) {  
 totalSampleNRequest.setStatNames(new String[]{"General\_Sample"});  
 } else {  
 totalSampleNRequest.setStatNames(new String[]{"Case\_Count"});  
 }  
 totalSampleNRequest.setFields(new String[0][]); // Empty fields for totalSampleN  
 //totalSampleNRequest.setFormat(new ResultsFormat());  
 return totalSampleNRequest;  
 }  
  
 private StatRequest createQuestionRequest(GetDataViewRequestForQuestionListDataModel adm, int crossTabIndex) {  
 StatRequest questionRequest = new StatRequest();  
 questionRequest.setRequestId(adm.getRequestId());  
 questionRequest.setGuids(adm.getGuids());  
 questionRequest.setReportTypeName(adm.getReportTypeName());  
  
 // Set appropriate stat names and fields for the question  
 if (adm.getReportTypeName().equals("WalledGardens")) {  
 questionRequest.setStatNames(new String[]{"General\_Sample"});  
 } else {  
 questionRequest.setStatNames(new String[]{"Case\_Count"});  
 }  
  
 questionRequest.setFields(new String[][]{{adm.getQuestionsList()[crossTabIndex]}});  
  
 ResultsFormat resultsFormat = new ResultsFormat();  
 resultsFormat.setRows(new String[]{adm.getQuestionsList()[crossTabIndex]});  
 questionRequest.setFormat(resultsFormat);  
  
 return questionRequest;  
 }  
  
 private List<String> extractData(StatResponse totalSampleNResponse, List<StatResponse> crosstabsResponseList) {  
 List<String> result = new ArrayList<>();  
  
 // Extract header information from totalSampleNResponse  
 List<Map<String, Object>> header = new ArrayList<>();  
 Map<String, Object> headerEntry = new HashMap<>();  
 headerEntry.put("reportName", totalSampleNResponse);  
 headerEntry.put("totalSampleN", getTotalSampleN(totalSampleNResponse));  
 header.add(headerEntry);  
  
 // Extract columns information from the first crosstabsResponse  
 List<List<String>> columns = new ArrayList<>();  
 for (StatResponse crosstabResponse : crosstabsResponseList) {  
 for (String[] column : crosstabResponse.getRows()) {  
 columns.add(Arrays.*asList*(column));  
 }  
 }  
  
 // Extract rows information from the first crosstabsResponse  
 List<List<String>> rows = new ArrayList<>();  
 for (String[] row : crosstabsResponseList.get(0).getRows()) {  
 rows.add(Arrays.*asList*(row));  
 }  
  
 // Create the final JSON structure  
 Map<String, Object> jsonStructure = new HashMap<>();  
 jsonStructure.put("header", header);  
 jsonStructure.put("columns", columns);  
 jsonStructure.put("rows", rows);  
  
 // Convert the JSON structure to a string and add it to the result list  
 result.add(jsonStructure.toString());  
  
 return result;  
 }  
  
 private Integer getTotalSampleN(StatResponse totalSampleNResponse) {  
 try {  
 // Assuming 'weighted base' is the key in basicStats for totalSampleN  
 for (String basicStat : totalSampleNResponse.getBaseN()) {  
 if ("weighted base".equals(basicStat)) {  
 return Integer.*parseInt*(basicStat);  
 }  
 }  
 } catch (NumberFormatException e) {  
 // Handle the exception appropriately  
 e.printStackTrace(); // or log it  
 }  
  
 // Return a default value or handle the case when totalSampleN is not found  
 return 0;  
 }