package com.nsw.backend.utils;  
  
import com.nsw.backend.validation.ValidateResult;  
import org.apache.commons.lang3.StringUtils;  
import org.apache.poi.hssf.util.HSSFColor;  
import org.apache.poi.ss.formula.FormulaParser;  
import org.apache.poi.ss.formula.FormulaRenderer;  
import org.apache.poi.ss.formula.FormulaType;  
import org.apache.poi.ss.formula.ptg.AreaPtgBase;  
import org.apache.poi.ss.formula.ptg.Ptg;  
import org.apache.poi.ss.formula.ptg.RefPtgBase;  
import org.apache.poi.ss.usermodel.\*;  
import org.apache.poi.ss.util.CellReference;  
import org.apache.poi.xssf.usermodel.\*;  
  
import java.util.Locale;  
  
public class ExcelUtils {  
  
 private static final ExcelUtils *excelUtils* = new ExcelUtils();  
  
 private static final String *DATA\_NUMBER\_FORMAT* = "\_-\* #,##0.0 \_₫\_-;-\* #,##0.0 \_₫\_-;\_-\* \"-\"?? \_₫\_-;\_-@\_-";  
  
 public ExcelUtils () {  
 }  
  
 public static synchronized ExcelUtils getInstance() {  
 return *excelUtils*;  
 }  
  
 */\*\*  
 \*  
 \** ***@param*** *sheet the sheet contains cell  
 \** ***@param*** *formula formula in string  
 \** ***@param*** *coldiff khoảng cách từ cột gốc tới cột đích: ví dụ: ô gốc là A1, ô đích là B2 => khoảng cách là 1, ngược lại khoảng cách là -1  
 \** ***@param*** *rowdiff khoảng cách từ dòng gốc tới dòng đích: ví dụ: ô gốc là A1, ô đích là B2 => khoảng cách là 1, ngược lại khoảng cách là -1  
 \** ***@return*** *new formula by references in string  
 \*/* public static String copyFormula(XSSFSheet sheet, String formula, int coldiff, int rowdiff) {  
 XSSFEvaluationWorkbook workbookWrapper =  
 XSSFEvaluationWorkbook.*create*(sheet.getWorkbook());  
 Ptg[] ptgs = FormulaParser.*parse*(formula, workbookWrapper, FormulaType.*CELL* , sheet.getWorkbook().getSheetIndex(sheet));  
  
 for (int i = 0; i < ptgs.length; i++) {  
 if (ptgs[i] instanceof RefPtgBase) { // base class for cell references  
 RefPtgBase ref = (RefPtgBase) ptgs[i];  
 if (ref.isColRelative())  
 ref.setColumn(ref.getColumn() + coldiff);  
 if (ref.isRowRelative())  
 ref.setRow(ref.getRow() + rowdiff);  
 }  
 else if (ptgs[i] instanceof AreaPtgBase) { // base class for range references  
 AreaPtgBase ref = (AreaPtgBase) ptgs[i];  
 if (ref.isFirstColRelative())  
 ref.setFirstColumn(ref.getFirstColumn() + coldiff);  
 if (ref.isLastColRelative())  
 ref.setLastColumn(ref.getLastColumn() + coldiff);  
 if (ref.isFirstRowRelative())  
 ref.setFirstRow(ref.getFirstRow() + rowdiff);  
 if (ref.isLastRowRelative())  
 ref.setLastRow(ref.getLastRow() + rowdiff);  
 }  
 }  
 formula = FormulaRenderer.*toFormulaString*(workbookWrapper, ptgs);  
 return formula;  
 }  
  
 public boolean isFormulaCell(Cell c){  
 return c.getCellTypeEnum() == CellType.*FORMULA*;  
 }  
  
 */\*\*  
 \* Take pre-cell's fomular and return suitable formula for cell in the right  
 \** ***@param*** *preCell - the left cell from current cell where youuuuuuuuuuuuuu are at  
 \* Posible converting cell over AA column  
 \* \*/* public static String cloneFormulaFromPreCell(Cell preCell) {  
 String oldFormula = preCell.getCellFormula().toUpperCase(Locale.*ROOT*).replaceAll(" ", "");  
 int length = oldFormula.length() - 1;  
 char[] formulArray = oldFormula.toCharArray();  
  
 StringBuilder finalF = new StringBuilder();  
  
 for (int i = 0; i < length; i++) {  
 char ch = formulArray[i];  
 char nextCh = formulArray[i + 1];  
 if (Character.*isLetter*(ch) && Character.*isDigit*(nextCh)) {  
 finalF.append(formulArray[i]);  
 String columnInString = ch + "";  
 int count = 0;  
  
 for (int j = i - 1; j > 0; j--) {  
 count++;  
 char backCh = formulArray[j];  
 if (!Character.*isLetter*(backCh)) {  
 columnInString = oldFormula.substring(j + 1, i + 1);  
  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + 1);  
  
 finalF.replace(finalF.length() - count, finalF.length(), columnInStringForNextCol);  
 break;  
 }  
 }  
 if (count == 0) {  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + 1);  
 finalF.replace(finalF.length() - 1, finalF.length(), columnInStringForNextCol);  
 }  
 } else {  
 finalF.append(formulArray[i]);  
 }  
 }  
 finalF.append(formulArray[length]);  
 return finalF.toString();  
 }  
  
 */\*\*  
 \* Take pre-cell's fomular and return suitable formula for cell in the right  
 \** ***@param*** *otherCell - the left cell from current cell where youuuuuuuuuuuuuu are at  
 \** ***@param*** *distance - the subtraction of (currentCell index - otherCell index)  
 \* Possibly converting cell over AA column  
 \* \*/* public static String cloneFormulaFromOtherCellByDistance(Cell otherCell, int distance) {  
 String oldFormula = otherCell.getCellFormula().toUpperCase(Locale.*ROOT*).replaceAll(" ", "");  
 int length = oldFormula.length() - 1;  
 char[] formulArray = oldFormula.toCharArray();  
  
 StringBuilder finalF = new StringBuilder();  
  
 for (int i = 0; i < length; i++) {  
 char ch = formulArray[i];  
 char nextCh = formulArray[i + 1];  
 if (Character.*isLetter*(ch) && Character.*isDigit*(nextCh)) {  
 finalF.append(formulArray[i]);  
 String columnInString = ch + "";  
 int count = 0;  
  
 for (int j = i - 1; j > 0; j--) {  
 count++;  
 char backCh = formulArray[j];  
 if (!Character.*isLetter*(backCh)) {  
 columnInString = oldFormula.substring(j + 1, i + 1);  
  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + distance);  
  
 finalF.replace(finalF.length() - count, finalF.length(), columnInStringForNextCol);  
 break;  
 }  
 }  
 if (count == 0) {  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + distance);  
 finalF.replace(finalF.length() - 1, finalF.length(), columnInStringForNextCol);  
 }  
 } else {  
 finalF.append(formulArray[i]);  
 }  
 }  
 finalF.append(formulArray[length]);  
 return finalF.toString();  
 }  
  
 */\*\*  
 \* Take pre-cell's fomular and return suitable formula for next cell in the right  
 \** ***@param*** *oldFormula - the left cell from current cell where youuuuuuuuuuuuuu are at  
 \* \*/* public static String cloneFormulaFromPreFormula(String oldFormula) {  
 int length = oldFormula.length() - 1;  
 char[] formulArray = oldFormula.toCharArray();  
  
 StringBuilder finalF = new StringBuilder();  
  
 for (int i = 0; i < length; i++) {  
 char ch = formulArray[i];  
 char nextCh = formulArray[i + 1];  
 if (Character.*isLetter*(ch) && Character.*isDigit*(nextCh)) {  
 finalF.append(formulArray[i]);  
 String columnInString = ch + "";  
 int count = 0;  
  
 for (int j = i - 1; j > 0; j--) {  
 count++;  
 char backCh = formulArray[j];  
 if (!Character.*isLetter*(backCh)) {  
 columnInString = oldFormula.substring(j + 1, i + 1);  
  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + 1);  
  
 finalF.replace(finalF.length() - count, finalF.length(), columnInStringForNextCol);  
 break;  
 }  
 }  
 if (count == 0) {  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + 1);  
 finalF.replace(finalF.length() - 1, finalF.length(), columnInStringForNextCol);  
 }  
 } else {  
 finalF.append(formulArray[i]);  
 }  
 }  
 finalF.append(formulArray[length]);  
 return finalF.toString();  
 }  
  
 */\*\*  
 \* Take pre-cell's fomular and return suitable formula for next cell in the right  
 \** ***@param*** *oldFormula - the left cell from current cell where youuuuuuuuuuuuuu are at  
 \** ***@param*** *distance - the subtraction of (currentCell index - otherCell index)  
 \* \*/* public static String cloneFormulaFromOtherFormulaByDistance(String oldFormula, int distance) {  
 int length = oldFormula.length() - 1;  
 char[] formulArray = oldFormula.toCharArray();  
  
 StringBuilder finalF = new StringBuilder();  
  
 for (int i = 0; i < length; i++) {  
 char ch = formulArray[i];  
 char nextCh = formulArray[i + 1];  
 if (Character.*isLetter*(ch) && Character.*isDigit*(nextCh)) {  
 finalF.append(formulArray[i]);  
 String columnInString = ch + "";  
 int count = 0;  
  
 for (int j = i - 1; j > 0; j--) {  
 count++;  
 char backCh = formulArray[j];  
 if (!Character.*isLetter*(backCh)) {  
 columnInString = oldFormula.substring(j + 1, i + 1);  
  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + distance);  
  
 finalF.replace(finalF.length() - count, finalF.length(), columnInStringForNextCol);  
 break;  
 }  
 }  
 if (count == 0) {  
 int columnInInt = CellReference.*convertColStringToIndex*(columnInString);  
 String columnInStringForNextCol = CellReference.*convertNumToColString*(columnInInt + distance);  
 finalF.replace(finalF.length() - 1, finalF.length(), columnInStringForNextCol);  
 }  
 } else {  
 finalF.append(formulArray[i]);  
 }  
 }  
 finalF.append(formulArray[length]);  
 return finalF.toString();  
 }  
  
 public CellStyle border(CellStyle style) {  
 style.setBorderRight(BorderStyle.*THIN*);  
 style.setBorderBottom(BorderStyle.*THIN*);  
 style.setBorderLeft(BorderStyle.*THIN*);  
 style.setBorderTop(BorderStyle.*THIN*);  
 return style;  
 }  
  
 public XSSFCellStyle border(XSSFCellStyle style) {  
 style.setBorderRight(BorderStyle.*THIN*);  
 style.setBorderBottom(BorderStyle.*THIN*);  
 style.setBorderLeft(BorderStyle.*THIN*);  
 style.setBorderTop(BorderStyle.*THIN*);  
 return style;  
 }  
  
 public CellStyle borderNone(CellStyle style) {  
 style.setBorderRight(BorderStyle.*NONE*);  
 style.setBorderBottom(BorderStyle.*NONE*);  
 style.setBorderLeft(BorderStyle.*NONE*);  
 style.setBorderTop(BorderStyle.*NONE*);  
 return style;  
 }  
  
 public void borderNone(XSSFCellStyle style) {  
 style.setBorderRight(BorderStyle.*NONE*);  
 style.setBorderBottom(BorderStyle.*NONE*);  
 style.setBorderLeft(BorderStyle.*NONE*);  
 style.setBorderTop(BorderStyle.*NONE*);  
 }  
  
 public CellStyle createStyle(Workbook wb, boolean isBorder, boolean isLock, boolean isWrapText, Integer font,Integer horizontal, Integer vertical, String dataFormat){  
 CellStyle style = wb.createCellStyle();  
 if (isBorder){  
 border(style);  
 }else {  
 borderNone(style);  
 }  
 style.setLocked(isLock);  
 style.setWrapText(isWrapText);  
 vertical(vertical, style);  
 horizontal(horizontal, style);  
 font(wb, font, style);  
 if (StringUtils.*isNotEmpty*(dataFormat)){  
 style.setDataFormat(wb.createDataFormat().getFormat(dataFormat));  
 }  
 return style;  
 }  
  
 public XSSFCellStyle createStyle(Workbook wb, boolean isBorder, boolean isLock, boolean isWrapText, Integer font,Integer horizontal, Integer vertical, XSSFColor xssfColor, String dataFormat){  
 XSSFCellStyle style = (XSSFCellStyle)wb.createCellStyle();  
 if (isBorder){  
 border(style);  
 }else {  
 borderNone(style);  
 }  
 style.setLocked(isLock);  
 style.setWrapText(isWrapText);  
 vertical(vertical, style);  
 horizontal(horizontal, style);  
 font(wb, font, style);  
 if (StringUtils.*isNotEmpty*(dataFormat)){  
 style.setDataFormat(wb.createDataFormat().getFormat(dataFormat));  
 }  
 if (xssfColor != null){  
 style.setFillForegroundColor(xssfColor);  
 style.setFillPattern(FillPatternType.*SOLID\_FOREGROUND*);  
 }  
 return style;  
 }  
  
 private void font(Workbook wb, Integer font, CellStyle style) {  
 switch (font){  
 case 1: // normal  
 style.setFont(fontDataNormal(wb));  
 break;  
 case 2: // bold  
 style.setFont(fontDataBold(wb));  
 break;  
 case 3: // italic  
 style.setFont(fontDataBoldNormalItalic(wb));  
 break;  
 }  
 }  
  
 private void horizontal(Integer horizontal, CellStyle style) {  
 switch (horizontal){  
 case 1:// general  
 style.setAlignment(HorizontalAlignment.*GENERAL*);  
 break;  
 case 2: // left  
 style.setAlignment(HorizontalAlignment.*LEFT*);  
 break;  
 case 3: // BOTTOM  
 style.setAlignment(HorizontalAlignment.*CENTER*);  
 break;  
 case 4: // justify  
 style.setAlignment(HorizontalAlignment.*RIGHT*);  
 break;  
 case 5: // DISTRIBUTED  
 style.setAlignment(HorizontalAlignment.*FILL*);  
 break;  
 }  
 }  
  
 private void vertical(Integer vertical, CellStyle style) {  
 switch (vertical){  
 case 1:// top  
 style.setVerticalAlignment(VerticalAlignment.*TOP*);  
 break;  
 case 2: // center  
 style.setVerticalAlignment(VerticalAlignment.*CENTER*);  
 break;  
 case 3: // bottom  
 style.setVerticalAlignment(VerticalAlignment.*BOTTOM*);  
 break;  
 case 4: // justify  
 style.setVerticalAlignment(VerticalAlignment.*JUSTIFY*);  
 break;  
 case 5: // DISTRIBUTED  
 style.setVerticalAlignment(VerticalAlignment.*DISTRIBUTED*);  
 break;  
 }  
 }  
  
 private Font fontDataBold(Workbook wb){  
 Font f = wb.createFont();  
 f.setBold(true);  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)13);  
 return f;  
 }  
  
 private Font fontDataNormal(Workbook wb){  
 Font f = wb.createFont();  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)13);  
 return f;  
 }  
  
 private Font fontDataBoldNormalItalic(Workbook wb){  
 Font f = wb.createFont();  
 f.setItalic(true);  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)13);  
 return f;  
 }  
  
 */\*\*  
 \* check if input string is an integer number  
 \* vd: 5.1 -> false, 5.0 | 5 -> true  
 \* \*/* public static boolean isIntegerNumber(String input) throws NumberFormatException{  
 return !(Math.abs((long) Double.parseDouble(input) - Double.parseDouble(input)) > 0);  
 }  
  
 public CellStyle stringBorder(Workbook wb) {  
 CellStyle style = wb.createCellStyle();  
 style.setBorderRight(BorderStyle.THIN);  
 style.setBorderBottom(BorderStyle.THIN);  
 style.setBorderLeft(BorderStyle.THIN);  
 style.setBorderTop(BorderStyle.THIN);  
 style.setWrapText(true);  
 Font f = wb.createFont();  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)12);  
 style.setFont(f);  
 DataFormat dataFormat = wb.createDataFormat();  
 short format = dataFormat.getFormat("@");  
 style.setDataFormat(format);  
 return style;  
 }  
  
 public CellStyle stringBorderLock(Workbook wb) {  
 CellStyle style = wb.createCellStyle();  
 style.setBorderRight(BorderStyle.THIN);  
 style.setBorderBottom(BorderStyle.THIN);  
 style.setBorderLeft(BorderStyle.THIN);  
 style.setBorderTop(BorderStyle.THIN);  
 style.setWrapText(true);  
 Font f = wb.createFont();  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)13);  
 style.setFont(f);  
 style.setLocked(true);  
 return style;  
 }  
  
 public CellStyle numberBorder(Workbook wb) {  
 CellStyle style = wb.createCellStyle();  
 style.setBorderRight(BorderStyle.THIN);  
 style.setBorderBottom(BorderStyle.THIN);  
 style.setBorderLeft(BorderStyle.THIN);  
 style.setBorderTop(BorderStyle.THIN);  
 Font f = wb.createFont();  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)12);  
 style.setFont(f);  
 DataFormat dataFormat = wb.createDataFormat();  
 short format = dataFormat.getFormat(DATA\_NUMBER\_FORMAT);  
 style.setDataFormat(format);  
 return style;  
 }  
  
 public CellStyle dateBorder(Workbook wb) {  
 CellStyle cellStyle = wb.createCellStyle();  
 cellStyle.setAlignment(HorizontalAlignment.CENTER);  
 cellStyle.setBorderTop(BorderStyle.THIN);  
 cellStyle.setBorderRight(BorderStyle.THIN);  
 cellStyle.setBorderBottom(BorderStyle.THIN);  
 cellStyle.setBorderLeft(BorderStyle.THIN);  
 cellStyle.setDataFormat(wb.createDataFormat().getFormat("dd/mm/yyyy"));  
 cellStyle.setLocked(true);  
 Font f = wb.createFont();  
 f.setFontName("Times New Roman");  
 f.setFontHeightInPoints((short)12);  
 cellStyle.setFont(f);  
 return cellStyle;  
 }  
  
 public CellStyle percentBorder(Workbook wb) {  
 CellStyle cellStyle = wb.createCellStyle();  
 cellStyle.setAlignment(HorizontalAlignment.CENTER);  
 cellStyle.setBorderTop(BorderStyle.THIN);  
 cellStyle.setBorderRight(BorderStyle.THIN);  
 cellStyle.setBorderBottom(BorderStyle.THIN);  
 cellStyle.setBorderLeft(BorderStyle.THIN);  
 cellStyle.setDataFormat(wb.createDataFormat().getFormat("0.0%"));  
 cellStyle.setLocked(true);  
 return cellStyle;  
 }  
  
 public static void markErrorOnCell(Workbook wbReportInput, ValidateResult result) {  
 Sheet failedSheet = wbReportInput.getSheetAt(result.getSheetIndex());  
 Row row = failedSheet.getRow(result.getRow());  
 if (row == null) {  
 row = failedSheet.createRow(result.getRow());  
 Cell cell = row.createCell(result.getColumn());  
 makeErrorCommentWithCell(failedSheet, cell, result.getMessage());  
 return;  
 }  
 Cell failedCell = row.getCell(result.getColumn());  
 if (failedCell == null) {  
 failedCell = row.createCell(result.getColumn());  
 }  
 makeErrorCommentWithCell(failedSheet, failedCell, result.getMessage());  
 }  
  
 public static void makeErrorCommentWithCell(Sheet failedSheet, Cell cell, String message) {  
 Drawing hpt = failedSheet.createDrawingPatriarch();  
 XSSFClientAnchor anchor = new XSSFClientAnchor();  
 int colIndex = cell.getColumnIndex() + 1;  
 int rowIndex = cell.getRowIndex() + 1;  
 anchor.setCol1(colIndex);  
 anchor.setCol2(colIndex + 4);  
 anchor.setRow1(rowIndex);  
 anchor.setRow2(rowIndex + 3);  
  
 Comment comment1 = hpt.createCellComment(anchor);  
 comment1.setString(new XSSFRichTextString(message));  
 cell.setCellComment(comment1);  
  
 CellStyle style = failedSheet.getWorkbook().createCellStyle();  
 style.cloneStyleFrom(cell.getCellStyle());  
 style.setFillForegroundColor(HSSFColor.CORAL.index);  
 style.setFillPattern(FillPatternType.SOLID\_FOREGROUND.getCode());  
 cell.setCellStyle(style);  
 }  
  
 public boolean isEmptyCellOrNull(Cell cell) {  
 return cell == null || cell.getCellTypeEnum() == CellType.BLANK;  
 }  
  
}