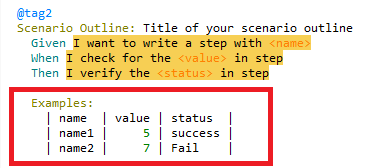
**¿Cómo Vincular Datos Externos (Excel) a nuestra Feature (.feature) ?**

Hasta el momento hemos usando nuestros datos desde la historia de usuario (.feature) o conectándonos a un archivo externo como excel (.xlsx).





ó

Sin embargo, se pueden presentar situaciones en las cuales tenemos unas ventajas y desventajas sea el modelo que se use.

Por tal razón ahora vamos a ver un modelo que integra los dos anteriores y en el cual lo que hacemos es crear nuestros datos en el archivo de Excel y básicamente antes de comenzar la ejecución vamos a cargar dichos datos en nuestra feature.

**Objetivo:**

Realizar la ejecución de nuestros escenarios con datos que serán cargados (copiados) en la feature desde un archivo externo de Excel.

**Pasos para configurar nuestro proyecto**

1. **Adicionar al archivo POM.XML las siguientes dependencias. (Nota: Si realizaste el Tip “Uso de Archivos Externos - Excel” , ya debes tener descargadas estas dependencias, así que puedes omitir este paso.)**

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>3.17</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>3.17</version>

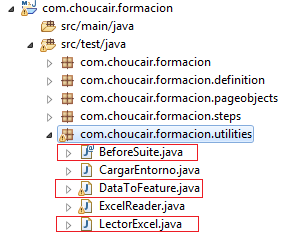
</dependency>

1. Debemos adicionar las siguientes Clases en el paquete “**com.choucair.formacion.utilities**” de nuestro proyecto.

BeforeSuite.java

DataToFeature.java

LectorExcel.java



|  |
| --- |
| BeforeSuite.java |
| package com.choucair.formacion.utilities;  import java.lang.annotation.ElementType;  import java.lang.annotation.Retention;  import java.lang.annotation.RetentionPolicy;  import java.lang.annotation.Target;  /\*\*  \* Se crea una anotación personalizada porque necesitamos copiar todos los datos del archivo de Excel al archivo .feature antes de comenzar a ejecutar  \*  \*/  //@Retention(RUNTIME)  //@Target(METHOD)  @Retention(RetentionPolicy.RUNTIME)  @Target(ElementType.METHOD)  public @interface BeforeSuite {  } |
| DataToFeature.java |
| **package** com.choucair.formacion.utilities;  **import** java.io.BufferedInputStream;  **import** java.io.BufferedReader;  **import** java.io.BufferedWriter;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **import** java.io.OutputStreamWriter;  **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** org.apache.poi.openxml4j.exceptions.InvalidFormatException;  /\*\*  \* Ingresa los datos obtenidos del archivo de Excel al archivo feature del cual se está llamando  \* **@since** 27/11/2017  \* **@author** bgaona  \*  \*/  **public** **class** DataToFeature {    /\*\*  \* Ingresa los datos obtenidos de un excel al archivo .feature del cual se está llamando, hace que se genere la tabla en el escenario  \* Outline como Data Table  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** featureFile Nombre del archivo .feature el cual se modificará, debe tener la ruta del archivo y la hoja ser usada  \* **@return**  \* **@throws** InvalidFormatException  \* **@throws** IOException  \*/    **private** **static** List<String> setExcelDataToFeature2(File featureFile)**throws** InvalidFormatException, IOException {  List<String> fileData = **new** ArrayList<String>();  **try** (BufferedReader buffReader = **new** BufferedReader(  **new** InputStreamReader(**new** BufferedInputStream(**new** FileInputStream(featureFile)), "UTF-8"))) {  String data;  List<Map<String, String>> excelData = **null**;  **boolean** foundHashTag = **false**;  **boolean** featureData = **false**;  **boolean** esUnRango = **false**;  **boolean** esMultiple = **false**;  **boolean** esRangoDefinido=**false**;  **while** ((data = buffReader.readLine()) != **null**) {  String [] dataVector = **null**;  String [] dataVectorRango=**null**;  String sheetName = **null**;  String excelFilePath = **null**;  String excelDataRange = **null**;  **int** filaSeleccionada=0;  **int** pos=0;  **if** (data.trim().contains("##@externaldata")) {  dataVector = data.trim().split("@");  excelFilePath = dataVector[2];//data.substring(StringUtils.ordinalIndexOf(data, "@", 2)+1, data.lastIndexOf("@"));  sheetName = dataVector[3];//data.substring(data.lastIndexOf("@")+1, data.length());  **if** (dataVector.length == 4)  {  esUnRango=**true**;  }  **if** (dataVector.length == 5)  {  **if**(dataVector[4].toString().contains("-"))  {  dataVectorRango = dataVector[4].trim().split("-");  //esUnRango=true;  esRangoDefinido=**true**;  filaSeleccionada= Integer.*parseInt*(dataVectorRango[pos])-1;      }  **else**  {  **if**(dataVector[4].toString().contains(","))  {  dataVectorRango = dataVector[4].trim().split(",");  esUnRango=**true**;  esMultiple=**true**;  filaSeleccionada= Integer.*parseInt*(dataVectorRango[pos])-1;    }  **else**  {    filaSeleccionada=Integer.*parseInt*(dataVector[4])-1;  }    }    }  foundHashTag = **true**;  fileData.add(data);  } **if** (foundHashTag) {  excelData = **new** LectorExcel().getData(excelFilePath, sheetName);    **for** (**int** rowNumber = filaSeleccionada; rowNumber < excelData.size()-1; rowNumber++) {  String cellData = "";  **for** (Entry<String, String> mapData : excelData.get(rowNumber).entrySet()) {  **if** (dataVectorRango==**null**)  {  **if**(rowNumber==filaSeleccionada-1 && esUnRango==**false**)  {  cellData = cellData + " |" + mapData.getValue();  }  **else**  {  cellData = cellData + " |" + mapData.getValue();  }  }  **else**  {    **if**(esRangoDefinido==**true**)  {  **if**(rowNumber<Integer.*parseInt*(dataVectorRango[1]) )  {  cellData = cellData + " |" + mapData.getValue();  }    }**else**  {  **if**(rowNumber+1==Integer.*parseInt*(dataVectorRango[pos]) && esUnRango==**true**)  {  cellData = cellData + " |" + mapData.getValue();  }    }  }  }  fileData.add(cellData + "|");  **if** (esUnRango==**false**) {  **if**(esRangoDefinido==**false**)  rowNumber=excelData.size();  }  **if**(esMultiple==**true**)  {  **if**(pos+1<dataVectorRango.length)  {  filaSeleccionada=Integer.*parseInt*(dataVectorRango[pos+1])-1;  rowNumber=filaSeleccionada-1;  pos++;  }  **else**  {  rowNumber=excelData.size()-1;  }    }  **if**(esRangoDefinido==**true**)  {  **if**(rowNumber+1==Integer.*parseInt*(dataVectorRango[1]))  {  rowNumber=excelData.size()-1;  pos++;  }**else**  {  pos++;  }    }      }  foundHashTag = **false**;  featureData = **true**;  **continue**;  }  **if**(data.startsWith("|")||data.endsWith("|")){  **if**(featureData){  **continue**;  } **else**{  fileData.add(data);  **continue**;  }  } **else** {  featureData = **false**;  }  fileData.add(data);  }  }  **return** fileData;  }        /\*\*  \* Lista de todos los features con sus respectivos archivo de excel que se usarán en la prueba  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** folder Carpeta donde estarán los archivo .feature  \* **@return**  \*/  **private** **static** List<File> listOfFeatureFiles(File folder) {  List<File> featureFiles = **new** ArrayList<File>();  **if** (folder.getName().endsWith(".feature")) {  featureFiles.add(folder);  }**else** {    **for** (File fileEntry : folder.listFiles()) {  **if** (fileEntry.isDirectory()) {  featureFiles.addAll(*listOfFeatureFiles*(fileEntry));  } **else** {  **if** (fileEntry.isFile() && fileEntry.getName().endsWith(".feature")) {  featureFiles.add(fileEntry);  }  }  }  }  **return** featureFiles;  }    /\*\*  \* Hace una lista con todos los features dependiendo de la ruta asignada  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** featuresDirectoryPath Ruta donde se encuentran los features que tendrán las tablas  \* **@throws** IOException  \* **@throws** InvalidFormatException  \*/  **public** **static** **void** overrideFeatureFiles(String featuresDirectoryPath)  //public void overrideFeatureFiles(String featuresDirectoryPath)  **throws** IOException, InvalidFormatException {  List<File> listOfFeatureFiles = *listOfFeatureFiles*(**new** File(featuresDirectoryPath));  **for** (File featureFile : listOfFeatureFiles) {  List<String> featureWithExcelData = *setExcelDataToFeature2*(featureFile);  **try** (BufferedWriter writer = **new** BufferedWriter(  **new** OutputStreamWriter(**new** FileOutputStream(featureFile), "UTF-8"));) {  **for** (String string : featureWithExcelData) {  writer.write(string);  writer.write("\n");  }  }  }  }  } |
| LectorExcel.java |
| **package** com.choucair.formacion.utilities;  **import** java.io.File;  **import** java.io.IOException;  **import** java.util.ArrayList;  **import** java.util.LinkedHashMap;  **import** java.util.List;  **import** java.util.Map;  **import** org.apache.poi.openxml4j.exceptions.InvalidFormatException;  **import** org.apache.poi.ss.usermodel.Cell;  **import** org.apache.poi.ss.usermodel.CellType;  **import** org.apache.poi.ss.usermodel.Row;  **import** org.apache.poi.ss.usermodel.Sheet;  **import** org.apache.poi.ss.usermodel.Workbook;  **import** org.apache.poi.ss.usermodel.WorkbookFactory;  **import** org.apache.poi.ss.util.NumberToTextConverter;  **public** **class** LectorExcel {  /\*\*  \* Obtiene los datos de un archivo de excel, teniendo en cuenta el nombre de la  \* hoja  \*  \* **@param** excelFilePath  \* Ruta del libro de excel  \* **@param** sheetName  \* Nombre de la hoja que contiene los datos  \* **@return**  \* **@throws** InvalidFormatException  \* Manejo de error por formato inválido  \* **@throws** IOException  \* Manejo de error para el proceso de entrada y salida de datos  \*/  **public** List<Map<String, String>> getData(String excelFilePath, String sheetName)  **throws** InvalidFormatException, IOException {  Sheet sheet = getSheetByName(excelFilePath, sheetName);  **return** readSheet(sheet);  }  /\*\*  \* Obtiene los datos de un archivo de excel, teniendo en cuenta el numero de la  \* hoja  \*  \* **@param** excelFilePath  \* Ruta del libro de excel  \* **@param** sheetNumber  \* Nombre de la hoja que contiene los datos  \* **@return**  \* **@throws** InvalidFormatException  \* Manejo de error por formato inválido  \* **@throws** IOException  \* Manejo de error para el proceso de entrada y salida de datos  \*/  **public** List<Map<String, String>> getData(String excelFilePath, **int** sheetNumber)  **throws** InvalidFormatException, IOException {  Sheet sheet = getSheetByIndex(excelFilePath, sheetNumber);  **return** readSheet(sheet);  }  /\*\*  \* Obtiene la hoja de trabajo donde se encuentran los datos de acuerdo a la  \* ruta del archivo  \*  \* **@param** excelFilePath  \* Ruta del libro de excel  \* **@param** sheetName  \* Nombre de la hoja que contiene los datos  \* **@return**  \* **@throws** IOException  \* **@throws** InvalidFormatException  \*/  **private** Sheet getSheetByName(String excelFilePath, String sheetName) **throws** IOException, InvalidFormatException {  Sheet sheet = getWorkBook(excelFilePath).getSheet(sheetName);  **return** sheet;  }  /\*\*  \* Obtiene los hoja de trabajo donde se encuentran los datos de acuerdo al index  \* de la hoja  \*  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** excelFilePath  \* Ruta del libro de excel  \* **@param** sheetNumber  \* Indice de tipo entero de la hoja en el libro de excel  \* **@return**  \* **@throws** IOException  \* **@throws** InvalidFormatException  \*/  **private** Sheet getSheetByIndex(String excelFilePath, **int** sheetNumber) **throws** IOException, InvalidFormatException {  Sheet sheet = getWorkBook(excelFilePath).getSheetAt(sheetNumber);  **return** sheet;  }  /\*\*  \* Devuelve el libro correspondiente a la hoja determinada con antelación  \*  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** excelFilePath  \* Ruta del archivo de excel  \* **@return**  \* **@throws** IOException  \* **@throws** InvalidFormatException  \*/  **private** Workbook getWorkBook(String excelFilePath) **throws** IOException, InvalidFormatException {  **return** WorkbookFactory.*create*(**new** File(excelFilePath));  }  /\*\*  \* Retorna la lista en forma de Map de todas las filas que contiene la hoja de  \* excel, teniendo en cuenta la primera fila como los nombres de la columna  \*  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** sheet  \* Hoja de excel  \* **@return**  \*/  **private** List<Map<String, String>> readSheet(Sheet sheet) {  Row row;  **int** totalRow = sheet.getPhysicalNumberOfRows(); //-------------> se deja la última fila Se quita una de las filas que la librería considera que tiene datos debido a que deja una en blanco  List<Map<String, String>> excelRows = **new** ArrayList<Map<String, String>>();  **int** headerRowNumber = getHeaderRowNumber(sheet);  **if** (headerRowNumber != -1) {  **int** totalColumn = sheet.getRow(headerRowNumber).getLastCellNum();  **int** setCurrentRow = 1;  **for** (**int** currentRow = setCurrentRow; currentRow <= totalRow; currentRow++) {  row = getRow(sheet, sheet.getFirstRowNum() + currentRow);  LinkedHashMap<String, String> columnMapdata = **new** LinkedHashMap<String, String>();  **for** (**int** currentColumn = 0; currentColumn < totalColumn; currentColumn++) {  columnMapdata.putAll(getCellValue(sheet, row, currentColumn));  }  excelRows.add(columnMapdata);  }  }  **return** excelRows;  }  /\*\*  \* Obtiene el número de filas conceniernte a encabezado de la hoja  \*  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** sheet  \* **@return**  \*/  **private** **int** getHeaderRowNumber(Sheet sheet) {  Row row;  **int** totalRow = sheet.getLastRowNum();  **for** (**int** currentRow = 0; currentRow <= totalRow + 1; currentRow++) {  row = getRow(sheet, currentRow);  **if** (row != **null**) {  **int** totalColumn = row.getLastCellNum();  **for** (**int** currentColumn = 0; currentColumn < totalColumn; currentColumn++) {  Cell cell;  cell = row.getCell(currentColumn, Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***);  **if** (cell.getCellTypeEnum() == CellType.***STRING***) {  **return** row.getRowNum();  } **else** **if** (cell.getCellTypeEnum() == CellType.***NUMERIC***) {  **return** row.getRowNum();  } **else** **if** (cell.getCellTypeEnum() == CellType.***BOOLEAN***) {  **return** row.getRowNum();  } **else** **if** (cell.getCellTypeEnum() == CellType.***ERROR***) {  **return** row.getRowNum();  }  }  }  }  **return** (-1);  }  /\*\*  \* Obtiene la fila de acuerdo a la hoja y el número de ésta  \*  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** sheet  \* **@param** rowNumber  \* **@return**  \*/  **private** Row getRow(Sheet sheet, **int** rowNumber) {  **return** sheet.getRow(rowNumber);  }  /\*\*  \* Obtiene el valor de cada una de las celdas -------> reevaluar y dejar como texto todos los valores  \*  \* **@since** 27/11/2017  \* **@author** bgaona  \* **@param** sheet  \* **@param** row  \* **@param** currentColumn  \* **@return**  \*/  **private** LinkedHashMap<String, String> getCellValue(Sheet sheet, Row row, **int** currentColumn) {  LinkedHashMap<String, String> columnMapdata = **new** LinkedHashMap<String, String>();  Cell cell;  **if** (row == **null**) {  **if** (sheet.getRow(sheet.getFirstRowNum()).getCell(currentColumn, Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***)  .getCellTypeEnum() != CellType.***BLANK***) {  String columnHeaderName = sheet.getRow(sheet.getFirstRowNum()).getCell(currentColumn)  .getStringCellValue();  columnMapdata.put(columnHeaderName, "");  }  } **else** {  cell = row.getCell(currentColumn, Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***);  **if** (cell.getCellTypeEnum() == CellType.***STRING***) {  **if** (sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex(), Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***)  .getCellTypeEnum() != CellType.***BLANK***) {  String columnHeaderName = sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex())  .getStringCellValue();  columnMapdata.put(columnHeaderName, cell.getStringCellValue());  }  } **else** **if** (cell.getCellTypeEnum() == CellType.***NUMERIC***) {  **if** (sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex(), Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***)  .getCellTypeEnum() != CellType.***BLANK***) {  String columnHeaderName = sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex())  .getStringCellValue();  columnMapdata.put(columnHeaderName, NumberToTextConverter.*toText*(cell.getNumericCellValue()));  }  } **else** **if** (cell.getCellTypeEnum() == CellType.***BLANK***) {  **if** (sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex(), Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***)  .getCellTypeEnum()!= CellType.***BLANK***) {  String columnHeaderName = sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex())  .getStringCellValue();  columnMapdata.put(columnHeaderName, "");  }  } **else** **if** (cell.getCellTypeEnum() == CellType.***BOOLEAN***) {  **if** (sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex(), Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***)  .getCellTypeEnum() != CellType.***BLANK***) {  String columnHeaderName = sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex())  .getStringCellValue();  columnMapdata.put(columnHeaderName, Boolean.*toString*(cell.getBooleanCellValue()));  }  } **else** **if** (cell.getCellTypeEnum() == CellType.***ERROR***) {  **if** (sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex(), Row.MissingCellPolicy.***CREATE\_NULL\_AS\_BLANK***)  .getCellTypeEnum() != CellType.***BLANK***) {  String columnHeaderName = sheet.getRow(sheet.getFirstRowNum()).getCell(cell.getColumnIndex())  .getStringCellValue();  columnMapdata.put(columnHeaderName, Byte.*toString*(cell.getErrorCellValue()));  }  }  }  **return** columnMapdata;  }  } |

1. En el paquete “**com.choucair.formacion**“ se debe crear la clave RunnerPersonalizado.java

|  |
| --- |
| RunnerPersonalizado.java |
| **package** com.choucair.formacion;  **import** java.lang.annotation.Annotation;  **import** java.lang.reflect.Method;  **import** org.junit.runner.Description;  **import** org.junit.runner.Runner;  **import** org.junit.runner.notification.RunNotifier;  **import** com.choucair.formacion.utilities.BeforeSuite;  **import** net.serenitybdd.cucumber.CucumberWithSerenity;  /\*\*  \* Personalización del Runner con el cual se puede determinar que busque y modifique los .feature antes de ser ejecutados  \* **@since** 27/11/2017  \*/  **public** **class** RunnerPersonalizado **extends** Runner {  /\*private Class<Cucumber> classValue;  private Cucumber cucumber;\*/    **private** Class<CucumberWithSerenity> classValue;  **private** CucumberWithSerenity cucumberWithSerenity;      **public** RunnerPersonalizado(Class<CucumberWithSerenity> classValue) **throws** Exception {  **this**.classValue = classValue;  cucumberWithSerenity = **new** CucumberWithSerenity(classValue);  }    @Override  **public** Description getDescription() {  **return** cucumberWithSerenity.getDescription();  }    **private** **void** runAnnotatedMethods(Class<?> annotation) **throws** Exception {  **if** (!annotation.isAnnotation()) {  **return**;  }  Method[] methods = **this**.classValue.getMethods();  **for** (Method method : methods) {  Annotation[] annotations = method.getAnnotations();  **for** (Annotation item : annotations) {  **if** (item.annotationType().equals(annotation)) {  method.invoke(**null**);  **break**;  }  }  }  }    @Override  **public** **void** run(RunNotifier notifier) {  **try** {  runAnnotatedMethods(BeforeSuite.**class**);  cucumberWithSerenity = **new** CucumberWithSerenity(classValue);  } **catch** (Exception e) {  e.printStackTrace();  }  cucumberWithSerenity.run(notifier);  }  } |

1. Debe modificar la clase RunnerTags.java con el siguiente código, es decir, elimina el que tiene actualmente y coloca el siguiente.

|  |
| --- |
| RunnerTags.java |
| **package** com.choucair.formacion;  **import** java.io.IOException;  **import** org.apache.poi.openxml4j.exceptions.InvalidFormatException;  **import** org.junit.runner.RunWith;  **import** com.choucair.formacion.utilities.BeforeSuite;  **import** com.choucair.formacion.utilities.DataToFeature;  **import** cucumber.api.CucumberOptions;  **import** cucumber.api.SnippetType;  @CucumberOptions (features = "src/test/resources/features/CRM/ZohoCRM.feature",tags= {"@CrearUsuarioExcel"}, snippets= SnippetType.***CAMELCASE***)  @RunWith(RunnerPersonalizado.**class**)  **public** **class** RunnerTags {  @BeforeSuite  **public** **static** **void** test() **throws** InvalidFormatException, IOException {  DataToFeature.*overrideFeatureFiles*("./src/test/resources/features/CRM/ZohoCRM.feature");  }  } |

**Nota: Debes tener en cuenta las 2 líneas resaltadas en amarillo, ya que deben modificarse en cada ejecución.**

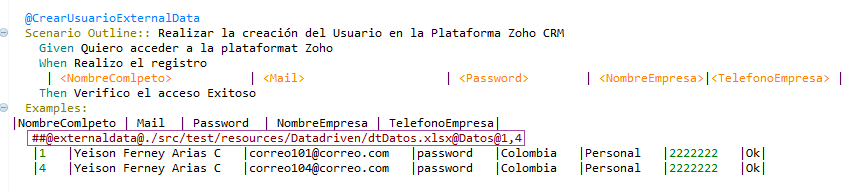
1. Para finalizar, al momento de crear nuestra **feature,** vamos a adicionar la siguiente línea de código. En esta especificamos la ruta de nuestro DataDriven de Excel, El nombre de la hoja e incluso los casos que deseamos ejecutar.

Esta línea debería ir donde anteriormente ubicábamos nuestra tabla de datos, es decir en la etiqueta Examples:

##@externaldata@./src/test/resources/Datadriven/dtDatos.xlsx@Datos@1

|  |
| --- |
| * **./src/test/resources/Datadriven/dtDatos.xlsx**: Ruta de nuestro archivo de Excel con los datos * **Datos**: Nombre de la hoja que contiene la información * **1 :** En el último parámetro podemos realizar 4 tipos de verificaciones al momento de querer ejecutar nuestro casos:   1. Colocar solo un numero de caso. Ej: 1 : trae a la feature el caso número 1 de nuestro data driven de Excel.   2. Colocar un rango de casos. Ej: 1-4 : trae a la feature el rango de casos del 1 al 4, es decir, 1,2,3 y 4.   3. Casos específicos. Ej: 1,2,4 : Trae a la feature los casos 1,2 y 4.   4. Sin parámetro, es decir no colocamos el ultimo @ : Trae todos los casos de nuestro Data Driven de Excel.   **Nota Importante**: Para que esta última parametrización en la cual especifico los casos a cargar me funcione correctamente, debo colocar en mi data driven de Excel para la columna 1 el número del caso. |

Escenario:



**¡¡ Ahora adelante, realiza la implementación de este cambio a tu solución¡!**