Lets play with Apache POI

Hey guys, hope everyone is safe and happy in this Covid-19 pandemic situation. After few months of silence, I am back with you for tackling some interesting issues we face in reporting software applications. Lets walk through!

I was developing a reporting module for an application , but this time it was challenging, why?

1. I needed to fill report template given in `docx` format.
2. Also, there were some tables needed to be filled with data over multiple pages.
3. Format the document after filling.
4. Finally , I needed to export `docx` to `pdf` and save or return through an API without saving the physical file.

So I am going to talk about those issues and how I could handle them with code. If you are not interested you can drop the article here, but if you are facing these now or have some time for hacking lets have some fun ;) .

Lets discuss

First of all , I asked myself why I need form filling rather than creating a new document object programmatically. Because most of my document has static data and only needed several field texts inserted. Lets say 80% is fixed and 20% needed to be filled, so having a template is easy, also I will show how easy to format the document this way.

At the beginning, I had no idea what approach I should follow, I have worked with jasper reporting before so I thought it would help me. But it is a huge burden of work because to create a jasper report I have to create `.jrxml` for each report, in my case I have ~15 report types to be implemented. So it is OBVIOUS I would not going to do that here.

I searched for tools compatible for above scenarios and found some open source and commercial tools. Then reviewed their features,

This is a popular and stable library for working with microsoft docx word files, also it can edit templates,but major issue I found with it was it can’t open directly in office 2013

package need to recover the file.

This supports office 2013 package and also popular among developers. And it has a good community support.

Who use Apache POI ?

Seems not a rich library.

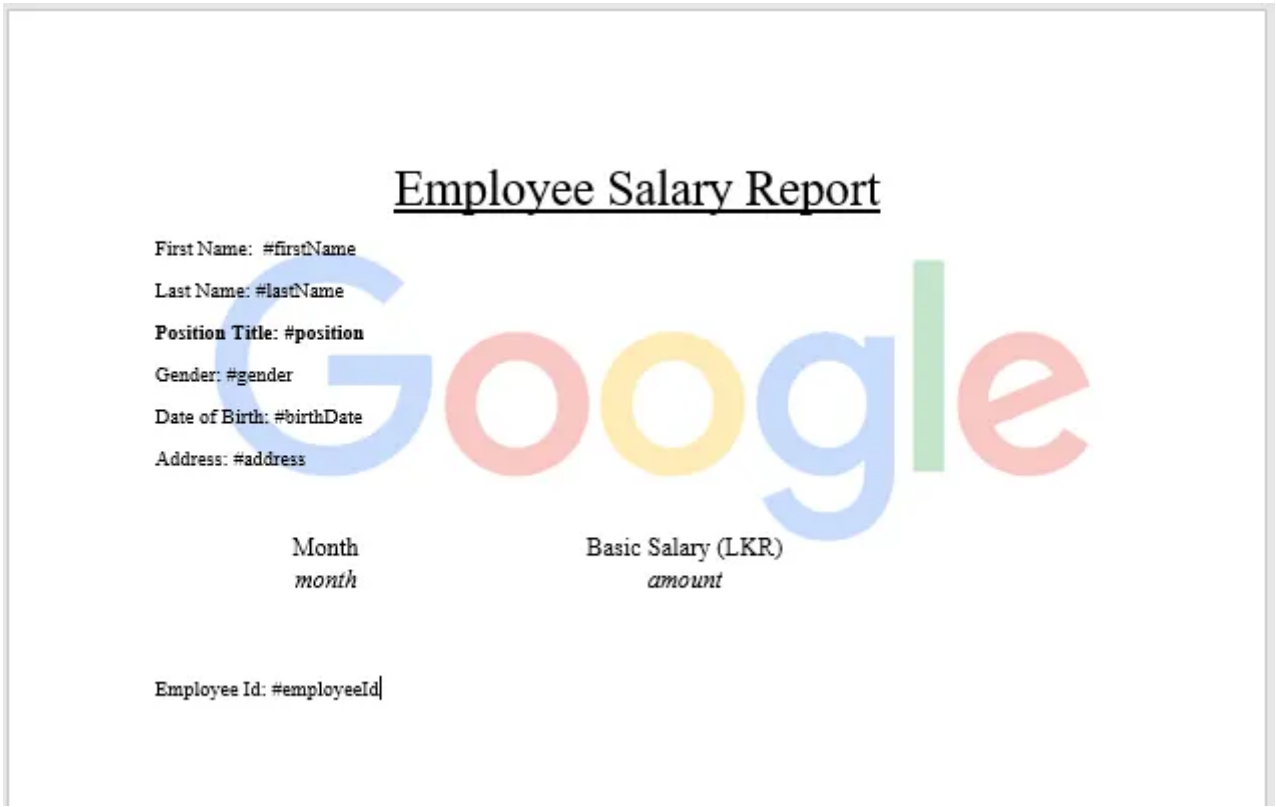
Comparison
https://java.libhunt.com/compare-apache-poi-vs-docx4j?rel=cmp-lib

And I found apache poi is a good project to start with. :)

Lets start coding

Step 1

We need to create a `docx` template with placeholders to merge our required texts. But placeholders should be unique. I am going to use prefix `#` to mark all the placeholders. Lets look how it looks.



Lets add maven dependencies to `pom.xml`

```
<dependency>
     <groupId>org.apache.poi</groupId>
     <artifactId>poi</artifactId>
     <version>4.1.2</version>
</dependency>

<dependency>
     <groupId>org.apache.poi</groupId>
     <artifactId>poi-ooxml</artifactId>
     <version>4.1.2</version>
</dependency>

<dependency>
     <groupId>org.apache.poi</groupId>
     <artifactId>poi-scratchpad</artifactId>
     <version>4.1.2</version>
</dependency>

<dependency>
     <groupId>fr.opensagres.xdocreport</groupId>
     <artifactId>fr.opensagres.poi.xwpf.converter.pdf</artifactId>
     <version>2.0.2</version>
</dependency>
```

Now we need to create a method to replace all the occurrences of placeholders with required texts.

```
private void replaceTextFor(XWPFDocument doc, HashMap map) {
        doc.getParagraphs().forEach(p -> p.getRuns().forEach(run -> {
            String text = run.text();
            map.forEach((findText, replaceText) -> {
                if (text.contains((String) findText)) {
                    run.setText(text.replace((String) findText, (String)
replaceText), 0);
                }
            });
        }));
    }
```

Lets pass XWPFDocument object of the template.docx and the required HashMap.

```
//get employee by id from database. this is dummy data.
EmployeeDetails employeeDetails = EmployeeDetails.builder()
                .firstName("Ranil")
```

```
                .lastName("Perera")
                .address("100,Temple Street,Colombo")
                .dob(LocalDate.now().minusYears(26))
                .employeeId(id)
                .gender("Male")
                .position("Software Engineer")
                .build();

String resourcePath = "template.docx";
Path templatePath =
Paths.get(DocumentHelper.class.getClassLoader().getResource(resourcePath).t
oURI());

XWPFDocument doc = new XWPFDocument(Files.newInputStream(templatePath));

HashMap<String, String> map = new HashMap<>();
map.put(VariableTypes.FIRST_NAME.getName(),
employeeDetails.getFirstName());
map.put(VariableTypes.LAST_NAME.getName(), employeeDetails.getLastName());
map.put(VariableTypes.POSITION.getName(), employeeDetails.getPosition());
map.put(VariableTypes.GENDER.getName(), employeeDetails.getGender());
map.put(VariableTypes.DATE_OF_BIRTH.getName(),
employeeDetails.getDob().toString());
map.put(VariableTypes.ADDRESS.getName(), employeeDetails.getAddress());
map.put(VariableTypes.EMPLOYEE_ID.getName(),
employeeDetails.getEmployeeId().toString());

replaceTextFor(doc, map);
```

Step 2

Now we need to fill tables.

Here we use a small trick to avoid the formatting table rows. What we are going to have is we create template row and copy that to new row and populate, the template row formatting will be added to the newly created row. Finally we will delete the template row. Cool right!

```
public void replaceSalaryTable(XWPFDocument doc, List<SalaryRecord>
salaryRecordList) {
        XWPFTable table = doc.getTableArray(0);//get the first table in the
template,replace index of the table you need
        int templateRowId = 1;//this is the template row id
        XWPFTableRow rowTemplate = table.getRow(templateRowId);

        salaryRecordList.forEach(salaryRecord -> {

            CTRow ctrow = null;
            try {
```

```
                ctrow =
CTRow.Factory.parse(rowTemplate.getCtRow().newInputStream());
            } catch (XmlException | IOException e) {
                e.printStackTrace();
            }

            XWPFTableRow newRow = new XWPFTableRow(ctrow, table);//create a
new row from template row

          //populate new row

newRow.getCell(0).getParagraphArray(0).getRuns().get(0).setText(salaryRecor
d.getMonth(), 0);

newRow.getCell(1).getParagraphArray(0).getRuns().get(0).setText(salaryRecor
d.getAmount(), 0);

            table.addRow(newRow);//add the row to table
        });

        table.removeRow(templateRowId);//remove the template row
    }
```

Lets inject data to this method,

```
//get data from database. this is dummy data.
List<SalaryRecord> salaryRecordList = Arrays.asList(
                SalaryRecord.builder().month("Jan
2020").amount(String.valueOf(1200.30)).build(),
                SalaryRecord.builder().month("Feb
2020").amount(String.valueOf(1200.30)).build(),
                SalaryRecord.builder().month("Mar
2020").amount(String.valueOf(1200.30)).build(),
                SalaryRecord.builder().month("Apr
2020").amount(String.valueOf(1200.30)).build(),
                SalaryRecord.builder().month("May
2020").amount(String.valueOf(1500.70)).build(),
                SalaryRecord.builder().month("Jun
2020").amount(String.valueOf(1500.70)).build()
        );

replaceSalaryTable(doc, salaryRecordList);
```

If you need to fill multiple tables , create multiple methods or a method factory.

Step 3

Lets talk about how to format the document. As we use the template docx file , we can add all the formatting to the docx file. Apache POI supports to format with code. Look at

following.

Adding tables : https://www.javatpoint.com/apache-poi-word-table

Styling : https://www.javatpoint.com/apache-poi-word-style

Aligning : https://www.javatpoint.com/apache-poi-word-aligning

Adding image : https://www.skptricks.com/2018/09/add-images-to-word-document-using-apache-poi-in-java.html

Step 4

After creating the files, now we need to export it to `pdf`. We need to use an out-of-the-box solution to generate pdf from apache poi tool. We have already added `fr.opensagres.poi.xwpf.converter.pdf` artifact to our project. Lets use this.

```
private void savePdf(String filePath, XWPFDocument doc) {
        try {
            PdfOptions options = PdfOptions.create();
            OutputStream out = new FileOutputStream(new File(filePath));
            PdfConverter.getInstance().convert(doc, out, options);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

If you don't want to export it to a physical file, lets create a `ByteArray` to pass it through API response.

```
ByteArrayOutputStream out = new ByteArrayOutputStream();

PdfOptions options = PdfOptions.create();
PdfConverter.getInstance().convert(doc, out, options);
out.close();

return out;
```

Find the all codes here

This is very simple , but, will ease your life a lot. Hope you enjoyed.