

Desafío FactorIT

Noelia Benitez

Exámen Técnico Backend

Funciones a Codificar:	1
Detalle de Implementación	1
Marco de Trabajo	1
Documentación Básica del Código	2
Tecnologías de BackEnd	3
Dependencias del Proyecto	3
Test Unitario	4
Servicio REST	5
Aclaraciones	7

Funciones a Codificar:

La Aplicación generará en tiempo de ejecución un Excel con 2 solapas con los datos obtenidos del XML "Examen-FIT.xml"

Detalle de Implementación

La Implementación se detalla en las siguientes secciones

Marco de Trabajo

Para organizar el trabajo liste las tareas en una Planilla con el orden necesario de las mismas, identificando las tareas mencionadas como Plus o valoración para desarrollarlas luego del objetivo principal. En dicha planilla fui marcando qué tareas fueron finalizadas indicando además la fecha.

Tareas				11/15 completadas
✓	Orden	Tarea	Descripción	Fecha
✓	1	Investigar sobre Spring Boot - Spring - MVC - Apache Struts-2	Se consultó varias páginas para tener noción sobre diferencias, ventajas y desventajas. Voy a trabajar con Spring Boot	28-07-2021
✓	2	Investigar como trabaja Spring con Excel y xml	Como idea final decidí obtener los objetos a partir del xml y generar el excel	28-07-2021
✓	3	Investigar sobre API-REST	https://spring.io/guides/tutorials/rest/	28-07-2021
✓	4	Codificación-Lectura de empresas del XML	Lectura de las Empresas del XML (ubicado Resources) y generación de objetos de tipo Empresa.	28-07-2021
✓	5	Validación sobre lectura de cada tag de empresas		29-07-2021
✓	6	Lectura de movimientos del XML		29-07-2021
✓	7	Validación sobre lectura de movimientos		29-07-2021
✓	8	Generar un Excel en tiempo de ejecución (primero	se genera un Excel con los datos de una sola empresa	29-07-2021

Después de concluir con la tarea 2 separé la Codificación en la siguientes secciones:

- obtener las empresas desde el XML y generar una lista de objetos de tipo Empresa
- validar la lectura de cada tag de empresa
- obtener los movimientos desde el XML y generar una lista de objetos de tipo Movimiento
- validar la lectura de cada tag de movimiento
- generar el Excel a partir de las dos listas de objetos
- generar y Probar un Test Unitario
- generar y Probar el servicio REST

Documentación Básica del Código

Modelo:

- Clase Empresa
- Clase Movimiento
- EmpresasMovsExcelExporter: Clase encargada de Generar el Excel a partir de las listas de Empresas y Movimientos
- ExcelPOIHelper: Clase encargada de la lectura de un Excel cuya ruta se recibe como parámetro. Esta clase fue creada específicamente para el desarrollo de la prueba unitaria.
- SwaggerConfig: Clase encargada de generar la documentación Swagger.

Controlador:

- MapStaffObjectHandlerSax: Clase encargada del Parseo del XML
- CustomErrorHandlerSax: Clase encargada del manejo de Excepciones en el parseo del XML.
- ExcelControlador: Clase encargada de gestionar la petición de generación del Excel, instanciando un objeto de la clase EmpresasMovsExcelExporter con las listas de Empresas y Movimientos.
- ExcelControladorRest: Clase encargada de gestionar el servicio REST

Tecnologías de BackEnd

Java 8

Herramientas de Soporte del Proyecto:

Maven y Spring Boot

Opté por estas tecnologías porque tuve la oportunidad de trabajar con las mismas anteriormente y porque entiendo que se dispone de mucho material publicado en la red.

Dependencias del Proyecto

spring-boot-starter-web
spring-boot-starter-test
poi
poi-ooxml
jackson-dataformat-xml
springfox-swagger2
springfox-swagger-ui

Para la generación del Excel utilicé la Librería Apache POI porque es bastante completa, soporta los formatos “.xls” y “.xlsx” y porque existen varios ejemplos publicados.

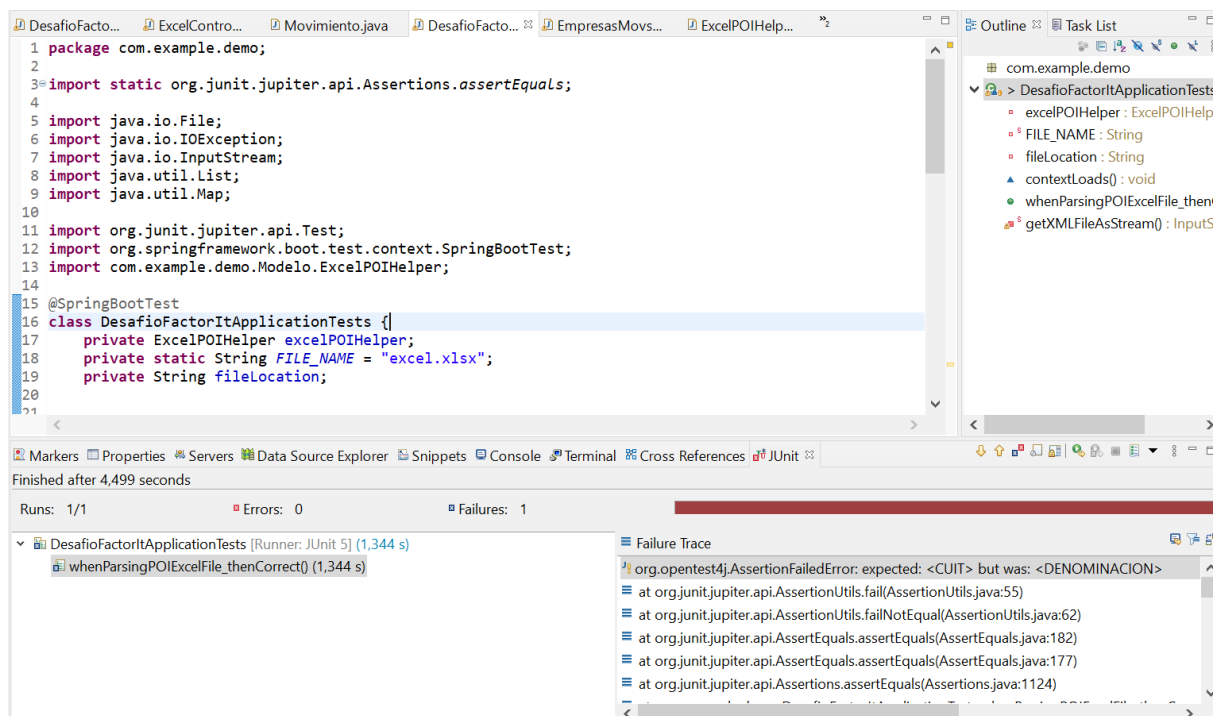
También por encontrar mayor cantidad de ejemplos opté por trabajar con la API SAX para el parseo del XML, además en la

documentación consultada se menciona que posee la ventaja de ocupar menos memoria que la opción DOM. Y porque simplemente necesitaba la opción de lectura del documento, no era necesario realizar la codificación de otra operación sobre el archivo.

Test Unitario

Se codificó una prueba con JUnit sobre el Excel generado en tiempo de ejecución.

Si se descomenta la línea 37 de la clase `DesafioFactorItApplicationTests` el resultado de la ejecución Junit Test arroja una falla:

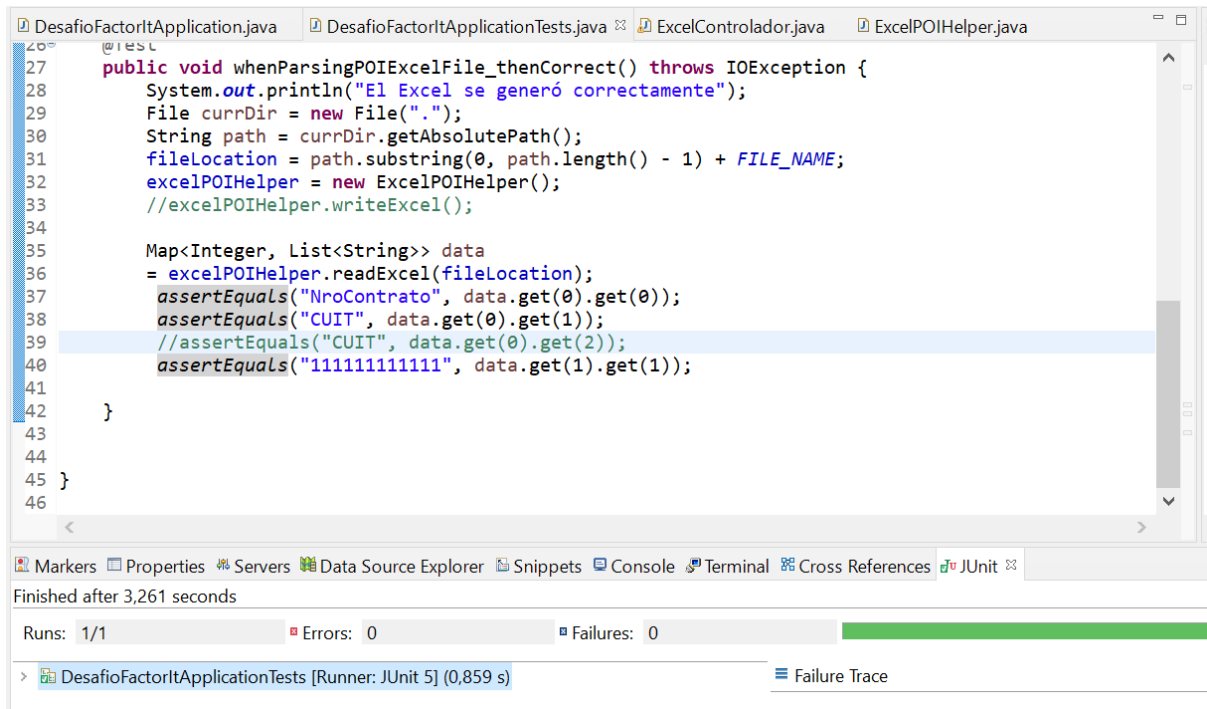


```
1 package com.example.demo;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 import java.io.File;
6 import java.io.IOException;
7 import java.io.InputStream;
8 import java.util.List;
9 import java.util.Map;
10
11 import org.junit.jupiter.api.Test;
12 import org.springframework.boot.test.context.SpringBootTest;
13 import com.example.demo.Modelo.ExcelPOIHelper;
14
15 @SpringBootTest
16 class DesafioFactorItApplicationTests {
17     private ExcelPOIHelper excelPOIHelper;
18     private static String FILE_NAME = "excel.xlsx";
19     private String fileLocation;
20
21     @Test
22     void whenParsingPOIExcelFile_thenCorrect() {
23         // ... (code commented out in the image) ...
24     }
25 }
```

Failure Trace

```
org.opentest4j.AssertionFailedError: expected: <CUIT> but was: <DENOMINACION>
    at org.junit.jupiter.api.AssertionUtils.fail(AssertionUtils.java:55)
    at org.junit.jupiter.api.AssertionUtils.failNotEqual(AssertionUtils.java:62)
    at org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:182)
    at org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:177)
    at org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:1124)
```

Con la línea 37 comentada la ejecución del Test sobre el archivo original no arroja fallas:



The screenshot shows an IDE with four tabs: `DesafioFactorItApplication.java`, `DesafioFactorItApplicationTests.java`, `ExcelControlador.java`, and `ExcelPOIHelper.java`. The `DesafioFactorItApplicationTests.java` tab is active, displaying a JUnit test method `whenParsingPOIExcelFile_thenCorrect()`. The code includes file path handling, instantiation of `ExcelPOIHelper`, reading of an Excel file, and assertions for specific data points. The test is annotated with `@Test` and `throws IOException`. Below the code editor, the IDE's status bar shows the test results: "Finished after 3,261 seconds", "Runs: 1/1", "Errors: 0", and "Failures: 0". A green progress bar indicates successful completion. The bottom status bar shows the runner: "DesafioFactorItApplicationTests [Runner: JUnit 5] (0,859 s)".

```
27 public void whenParsingPOIExcelFile_thenCorrect() throws IOException {
28     System.out.println("El Excel se generó correctamente");
29     File currDir = new File(".");
30     String path = currDir.getAbsolutePath();
31     fileLocation = path.substring(0, path.length() - 1) + FILE_NAME;
32     excelPOIHelper = new ExcelPOIHelper();
33     //excelPOIHelper.writeExcel();
34
35     Map<Integer, List<String>> data
36     = excelPOIHelper.readExcel(fileLocation);
37     assertEquals("NroContrato", data.get(0).get(0));
38     assertEquals("CUIT", data.get(0).get(1));
39     //assertEquals("CUIT", data.get(0).get(2));
40     assertEquals("11111111111", data.get(1).get(1));
41
42 }
43
44
45 }
46
```

Markers Properties Servers Data Source Explorer Snippets Console Terminal Cross References JUnit

Finished after 3,261 seconds

Runs: 1/1 Errors: 0 Failures: 0

> DesafioFactorItApplicationTests [Runner: JUnit 5] (0,859 s) Failure Trace

Servicio REST

Después de ver varias opciones y leer bastante me pareció la mejor opción utilizar la dependencia Jackson XML para el desarrollo. Si bien no logré el objetivo de codificar un servicio que reciba el XML y devuelva como resultado el Excel decidí que al menos podría generar un método `@PostMapping` que consuma específicamente archivos XML y que genere una salida JSON, y de esta manera probar la herramienta Swagger (una de las solicitadas para exponer el servicio). A continuación dejo algunas capturas:

Caso en donde se envía como parámetro un XML conteniendo una empresa:

excel-controlador-rest Excel Controlador Rest

POST /Empresas postXmlEmpresa

Parameters Cancel

Name	Description
empresaXML required (body)	empresaXML Example Value Model <pre><?xml version="1.0" encoding="UTF-8"?> <Empresa> <codigopostal>3300</codigopostal> <cuit>11111111</cuit> <denominacion>PRUEBA</denominacion> <domicilio>DOM PRUEBA</domicilio> <nroContrato>1</nroContrato> <productor>string</productor> </Empresa></pre>

Parameter content type: application/xml

Curl

```
curl -X POST "http://localhost:8080/Empresas" -H "accept: application/json" -H "Content-Type: application/xml" -d "<?xml version='1.0' encoding='UTF-8'?>
<Empresa>\t<codigopostal>3300</codigopostal>\t<cuit>11111111</cuit>\t<denominacion>PRUEBA</denominacion>\t<domicilio>DOM
PRUEBA</domicilio>\t<nroContrato>1</nroContrato>\t<productor>string</productor></Empresa>"
```

Request URL

http://localhost:8080/Empresas

Server response

Code	Details
200	<p>Response body</p> <pre>{ "cuit": "11111111", "productor": "string", "codigopostal": "3300", "domicilio": "DOM PRUEBA", "denominacion": "PRUEBA", "nroContrato": 1 }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Sun, 01 Aug 2021 14:23:16 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Caso en donde se envía como parámetro un JSON:

Curl

```
curl -X POST "http://localhost:8080/Empresas" -H "accept: application/json" -H "Content-Type: application/xml" -d "{ \"cuit\": \"11111111\", \"productor\": \"string\", \"codigopostal\": 3300, \"domicilio\": \"DOM PRUEBA\", \"denominacion\": \"PRUEBA\", \"nroContrato\": 1}"
```

Request URL

http://localhost:8080/Empresas

Server response

Code	Details
400 <i>Undocumented</i>	<p>Error:</p> <p>Response body</p> <pre>{ "timestamp": "2021-08-01T14:31:01.762+00:00", "status": 400, "error": "Bad Request", "path": "/Empresas" }</pre> <p>Response headers</p> <pre>connection: close content-type: application/json date: Sun, 01 Aug 2021 14:31:01 GMT transfer-encoding: chunked</pre>

Aclaraciones

- no se incluyó Persistencia ni Seguridad en el desarrollo, entiendo que no formaban parte de las especificaciones.
- Por tratarse de un exámen específico sobre Backend no se desarrolló ninguna vista
- el archivo XML se encuentra en "`\DesafioFactorIT\src\main\resources`"
- el archivo excel es generado en "`\DesafioFactorIT`" con el nombre "`excel.xlsx`"
- No alcancé a formatear las columnas del Excel, como no encontré la manera de hacerlo y todavía me quedaban puntos por desarrollar decidí dejarlo de lado. Pero se puede verificar en el Excel que cada celda contiene los valores en el formato correcto: Clase "`EmpresasMovsExcelExporter`" método "`createCell`"