Informe del trabajo práctico para la materia *Diseño y Procesamiento de Documentos XML*

Integrantes:

Julián Antonielli

Luciano Mosquera

Ramiro Olivera Fedi

Comisión S

Primer cuatrimestre, 2016

Tema: A

Fecha de entrega: 21/06/2016

1) SHELL Script (*trabajo.sh*)

El programa comienza a correr mediante la ejecución en consola del script en ***BASH*** *trabajo.sh*. Para su correcta ejecución deben otorgarse permisos de ejecución sobre el archivo. Se lee mediante argumento la fecha ingresada por el usuario. En esta instancia validamos la existencia del argumento, para no ejecutar las llamadas si no existía la fecha a buscar. De pasar la validación, ejecutamos mediante la herramienta ***SAXON*** los archivos *junta.qx* y *eventos.xls* pasando con los flags y parámetros necesarios para su correcta ejecución, y redirigiendo su salida a los archivos XML y JSON correspondientes.

Se debió investigar en profundidad los flags a utilizar en la llamada a la herramienta ***SAXON***. Por ejemplo, para pasar argumentos se utiliza la sintaxis `variable=valor`. Por otro lado, para la generación del ***JSON*** correspondiente se utilizaron los flags `-s` para distinguir el archivo a procesar y el flag `-xsl` para distinguir el archivo ***XSLT*** que se iba a encargar de dar formato al ***JSON***.

2) **XQuery (*junta.qx*)**

Escribimos una búsqueda ***XQuery*** para extraer la información que necesitábamos, dada una fecha en especial fijada por el usuario, que recibimos como parámetro por consola mediante la variable `$fecha` para filtrar los eventos. Se debió investigar cómo recibir por parámetros este tipo de datos, donde utilizamos una sentencia similar a `declare variable $fecha as xs:string external;`.

Al ingresar al flujo del script, se realiza una evaluación de la fecha mediante expresiones regulares. Si la fecha ingresada no coincide con ninguna de las fechas para las cuales existen datos en el archivo de eventos, pero la fecha tiene formato válido, se genera un XML con el tag de `<error>` especificando que no se encontraron eventos en la fecha ingresada. Si por otro lado, la fecha tiene un formato incorrecto, se genera el mismo tag con la leyenda *Fecha incorrecta* y explicando el formato adecuado para el procesamiento.

Si se encuentra en el archivo la fecha ingresada, se genera el archivo tal como se especificó. Se utilizó la función `match` para realizar las comparaciones entre texto de los campos con expresiones regulares, ya que permiten mayor flexibilidad y mantenibilidad. Otras funciones que se utilizaron son: `not` y `concat`, que son autodescriptivas.

Algunas de las consideraciones que contemplamos son las siguientes:

- Si no hay obras que matchean con eventos en esa fecha, se muestra el tag eventos vacio.
- Si no hay artistas que matchean, se muestra el tag artistas vacío.
- Si no hay sala dentro del lugar, se muestra el tag sala vacío.
- Si no hay título, se muestra el tag título vacío
- Si no hay descripción, se muestra el tag descripción vacio.
- Si no hay hora, se muestra el tag hora vacío.
- Si no se encuentra la sede, no se muestra ni el nombre ni el lugar.
- Si se encuentra la sede, se muestran los tags nombre y dirección, vacíos o no, según corresponda.
- Se consideró que el artista no sea NULL ni sea 0.

Finalmente se realizaron varias pruebas utilizando fechas validas e invalidas y se comprobaron los resultados mediante la validación del ***XML Schema*** propuesto por la cátedra. Utilizando la herramienta *http://www.freeformatter.com/xml-validator-xsd.html* se validaron todos los archivos, dando por terminado de manera efectiva el procesamiento del ***XQuery***.

3) XSLT (*eventos.xsl*)

A partir del archivo xml obtenido en el segundo paso, utilizamos ***XSLT*** para transformarlo a formato JSON.
Necesitamos investigar cómo se utilizaban las expresiones condicionales en ***XSLT*** (<xsl:if/>), junto con varias expresiones (test="condición") y funciones (not(), position(), last(), descritas abajo) para poder seleccionar las atributos correctos en donde debían colocarse comas (',') y saltos de línea.

La función not() devuelve true si su argumento es false, o false si su argumento es true. position() devuelve un número entero, comenzando en 1 que representa la posición del nodo actual respecto a la secuencia de contexto. Por último, last() retorna un número entero que representa la última posición de la secuencia de contexto.
Una combinación de estas 3 funciones se utilizó para chequear si algún nodo era el último en una secuencia.

Además, probamos varios atributos para el tag <xsl:output />, en particular terminamos utilizando el atributo method="text" para que archivo se muestre con el espacio y la indentación correcta.

Finalmente, validamos el archivo JSON obtenido con las herramientas

- http://jsonlint.com/
- http://www.freeformatter.com/json-validator.html

Comprobando que el archivo obtenido posee el formato correcto.