|  |
| --- |
| Faculty of Applied Sciences and Technology |
| **XML Data Processing** |
| ITC5202 - Project |
|  |
| **Cezmi Aktepe & Tim Burns** |
| **3/6/2021** |

|  |
| --- |
| This document explains how to process Order/Item XML data …………………………. |

Table of Contents

[Question 1 : 2](#_Toc65610936)

[Question 2 and 3 : XML Structure 5](#_Toc65610937)

[Question 4 : Design XSLT 10](#_Toc65610938)

[Question 5 and 8: XPath and XSLT 11](#_Toc65610939)

[Question 6 and 7: Use JavaScript to process XML data 16](#_Toc65610940)

[Question 9 17](#_Toc65610941)

[Bonus question 20](#_Toc65610942)

[Summary 21](#_Toc65610943)

# Question 1 :

(Describe you answer. How did you prove that the document is well-formed and valid? Add screenshots)

**Tim**

A well-formed document in XML is a document that "adheres to the syntax rules specified by the XML 1.0 specification in that it must satisfy both physical and logical structures".

**Our Document is well formed as**

* All xml elements have closing tags
* They are case sensitive
* They are properly nested
* It has a root element
* Attribute values are in quotes

**Our Doc is also valid at this time**

* It has no DTD atm to check for validity but is valid when checking for syntax errors
* We will add a DTD and compare to that for validity
* In other words, it must be syntactically correct in order for it to be valid.

**All the XML elements and child elements**

* <orders>
* <order >
* <customerid>
* <status>
* <item>
* <name>
* <price>
* <qty>
* </item>
* </order>

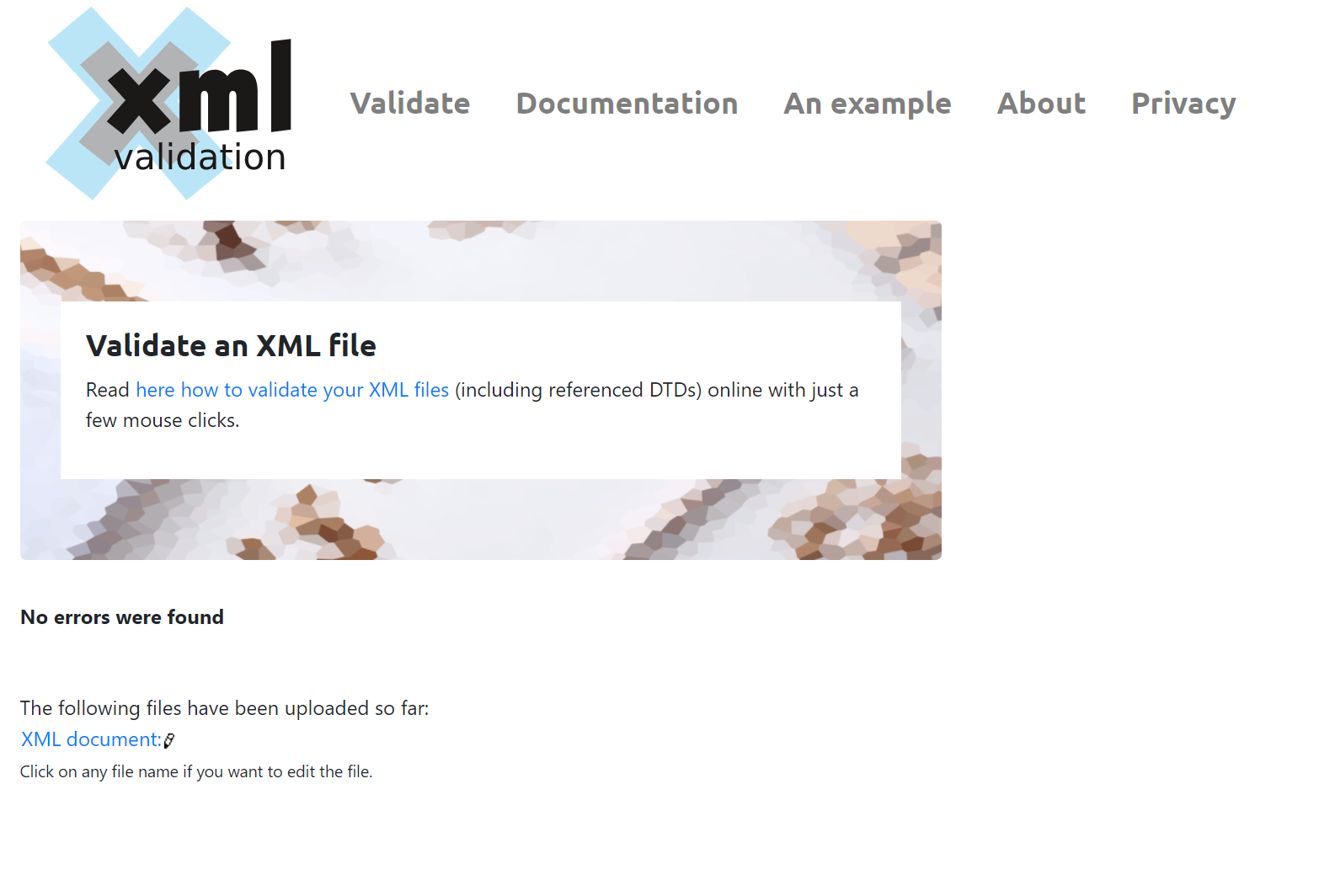
**How did you prove that the document is well-formed and valid?**

I used <https://www.xmlvalidation.com/>

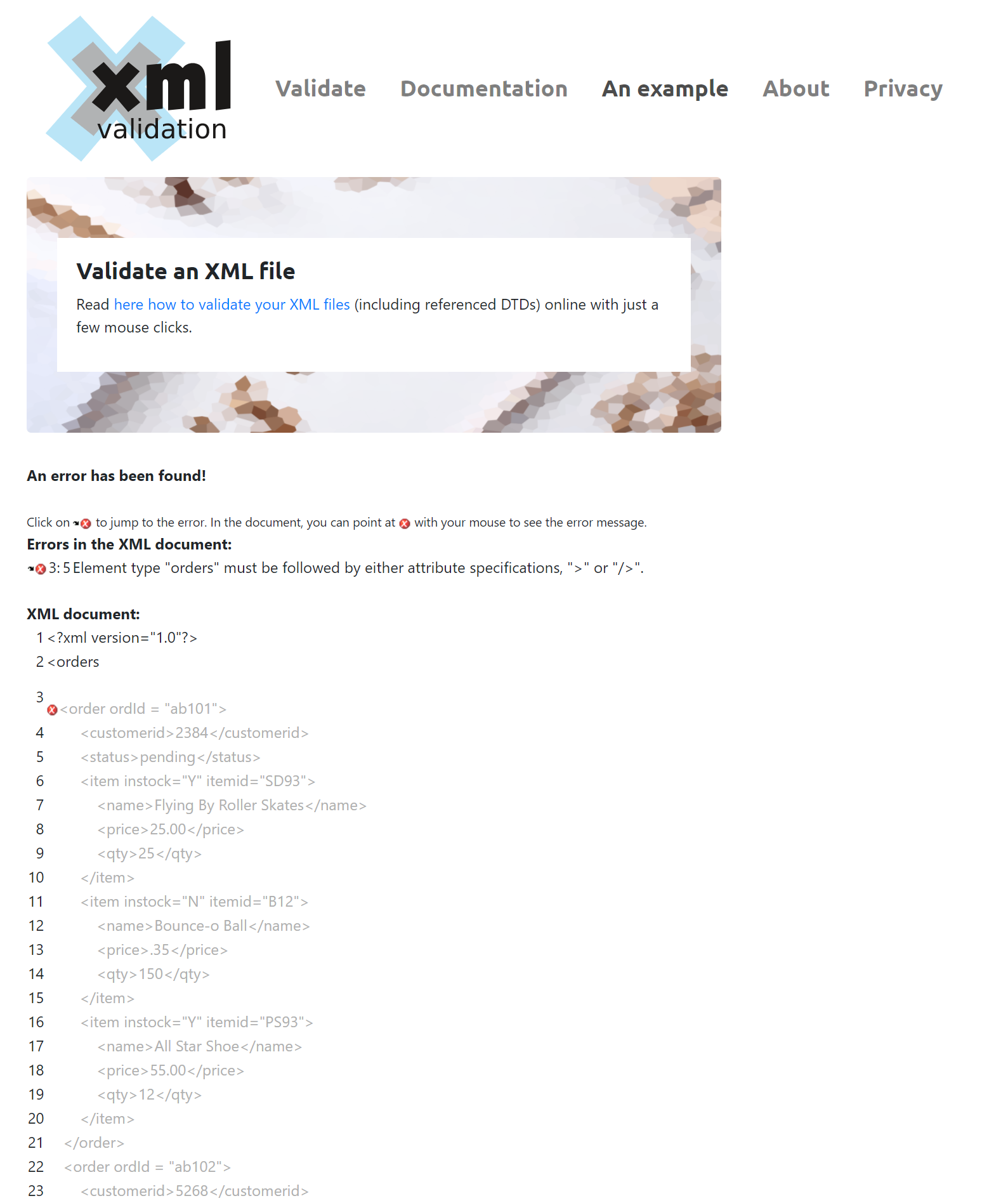
This site allows you to copy and paste your xml file and checks it to see if

* + It is well formed
  + It is valid (against any external internal schema)

Before we added our own schema, we can simply copy and paste the file and test whether it is well formed and valid. – it will be valid as there is no schema to test against.



If there are errors (that would make it not well formed – and thus valid like spelling) you will be displayed with a error message like so showing you what the error is.



# Question 2 and 3 : XML Structure

(1) Explain the major steps that you take to create DTD. Did you create a .dtd file, or you keep the DTD declaration inside the XML file? Why?

(2) Explain the major steps that you take to create XML Schema.

(3) How did you validate them? Add screenshots.

(4) Compare the DTD and Schema and show how DTD declaration are matched with Schema.

**Part 1 (Cezmi)**

I declared DOCTYPE and I found root element. I searched all elements and attributes along with their occurrences. I defined root element and added sub-root elements then I added attributes. Finally, I added occurrences of elements.

I kept the DTD declaration inside the XML file because it seems more tidy and clean code for me.

**Part 2 and 3 (Tim)**

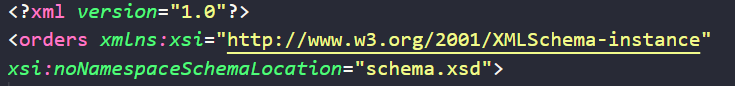
**Part 2** – schema: Explain the major steps that you take to create XML Schema.

To create a xml schema, you can use a website called <https://www.freeformatter.com/xsd-generator.html> . However, if you want to create a XML schema manually you need to do the following.

First you need to ensure that your XML doc I well formed, as mentioned above it must conform to the following

* All xml elements have closing tags
* They are case sensitive
* They are properly nested
* It has a root element
* Attribute values are in quotes

Once your XML doc is well formed you can create an XSD file where you will write your schema. You must link these two files on the XML side by using the line below, where the XSD file referenced is the file you created.



Next you can start working on the XSD file itself. Making sure you follow the same syntax as the XML doc itself.

On the XSD file you will need declare the schema in the schema element and it begins like this…

<xs:schema *attributeFormDefault*="unqualified" *elementFormDefault*="qualified" *xmlns:xs*="http://www.w3.org/2001/XMLSchema">

* You will have to close this element at the end of the doc
* </xs:schema>

Next you will define the elements starting with the root element of the XML doc. This element holds other elements but does not contain and text or attributes itself and looks like this.

<xs:element *name*="orders">

* This is also closed at the end of the doc
* </xs:element>

Next you will define the complex elements, the elements that hold data for the xml file (like text or attributes). You have to start it with the below for each parent element holding other elements

 <xs:complexType>

      <xs:sequence>

* These are also close at the end of the doc / end of the sequence
* </xs:sequence>
* </xs:complexType>

For complex elements (elements that contain data like text or attributes) you need to do the following

<xs:element *name*="order" *maxOccurs*="unbounded" *minOccurs*="0">

* And because it is a parent element it has to be closed as well at the end of place child elements inside
* This element is also not holding any text, so type is not needed

For elements that hold text you need to do the following

<xs:element *type*="xs:short" *name*="customerid"/>

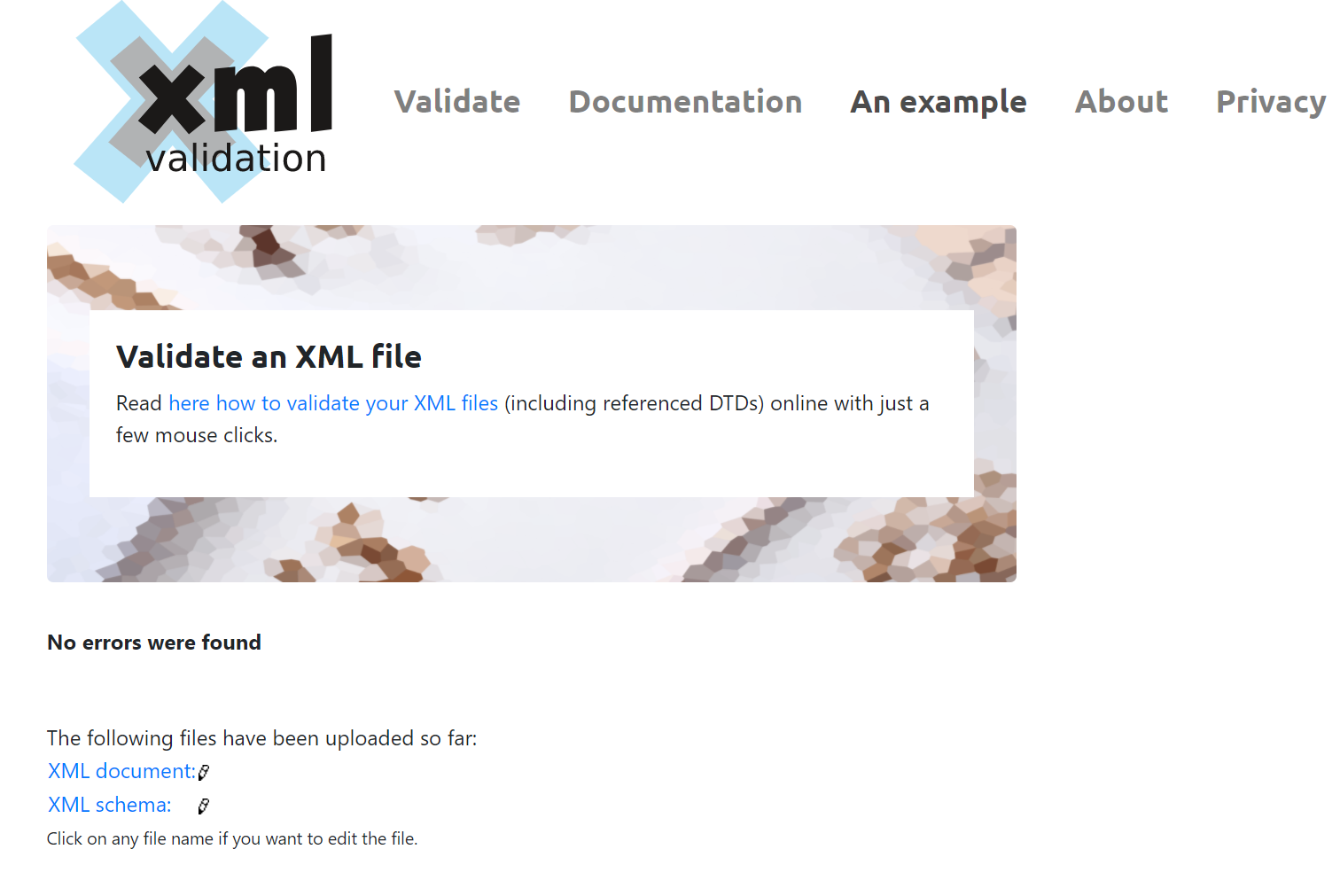
              <xs:element *type*="xs:string" *name*="status"/>

* And because these are not parent elements you can self-close them

If you follow and repeat these steps while following the syntax of the XML doc you will create a schema that correctly validates the XML doc.

**Part 3** – schema: How did you validate them? Add screenshots.

We validated it by using <https://www.xmlvalidation.com/> and validated it against external XML schema.

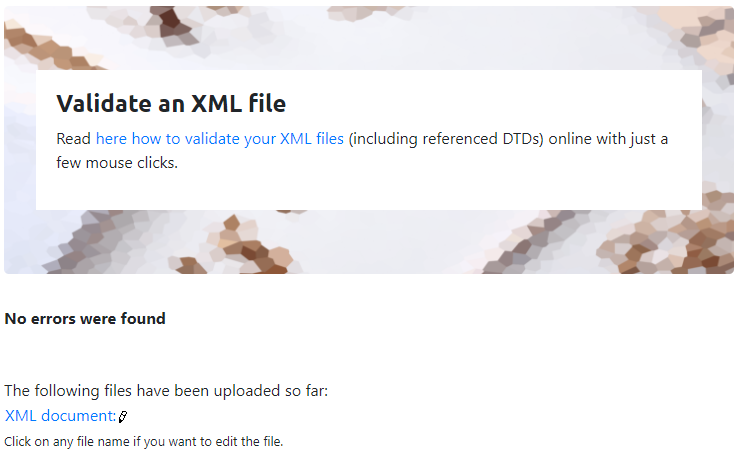


Again, it will display an error if there are any.

You can also compare its syntax to the syntax of the schema you generated to ensure it is well formed and valid.

**Part 3 (Cezmi)** - DTD

I used a web site to validate the DTD. Here is the screenshot.



**Part 4 (Cezmi and Tim)**

*Compare the DTD and Schema and show how DTD declaration are matched with Schema.*

Both have tree like structure begins with root element and then other sub-elements. Both uses ELEMENT name for the xml element tags and ATTRIBUTE name for the xml attributes. Both show data type.

<!DOCTYPE *orders*[

    <!ELEMENT orders ( order+ )>

    <!ELEMENT order ( customerid, status, item+ )>

    <!ELEMENT customerid ( #PCDATA )>

    <!ELEMENT status ( #PCDATA )>

    <!ELEMENT item ( name, price, qty )>

    <!ELEMENT name ( #PCDATA )>

    <!ELEMENT price ( #PCDATA )>

    <!ELEMENT qty ( #PCDATA )>

    <!ATTLIST order ordId CDATA #IMPLIED>

    <!ATTLIST item instock CDATA #REQUIRED >

    <!ATTLIST item itemid CDATA #REQUIRED >

]>

<xs:schema *attributeFormDefault*="unqualified" *elementFormDefault*="qualified" *xmlns:xs*="http://www.w3.org/2001/XMLSchema">

  <xs:element *name*="orders">

    <xs:complexType>

      <xs:sequence>

        <xs:element *name*="order" *maxOccurs*="unbounded" *minOccurs*="0">

          <xs:complexType>

            <xs:sequence>

              <xs:element *type*="xs:short" *name*="customerid"/>

              <xs:element *type*="xs:string" *name*="status"/>

              <xs:element *name*="item" *maxOccurs*="unbounded" *minOccurs*="0">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element *type*="xs:string" *name*="name"/>

                    <xs:element *type*="xs:float" *name*="price"/>

                    <xs:element *type*="xs:short" *name*="qty"/>

                  </xs:sequence>

                  <xs:attribute *type*="xs:string" *name*="instock" *use*="optional"/>

                  <xs:attribute *type*="xs:string" *name*="itemid" *use*="optional"/>

                </xs:complexType>

              </xs:element>

            </xs:sequence>

            <xs:attribute *type*="xs:string" *name*="ordId" *use*="optional"/>

          </xs:complexType>

        </xs:element>

      </xs:sequence>

    </xs:complexType>

  </xs:element>

</xs:schema>

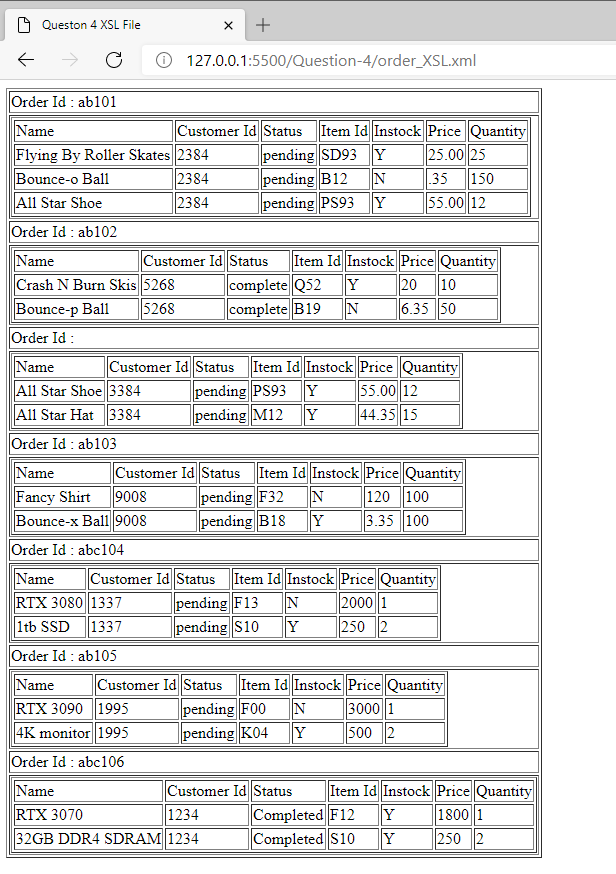
* From the two above examples you can see that DTD and Schema both follow the say syntax / structure, which is defined by the XML doc.
* They are used to test against that same xml doc to ensure any data adheres to the established syntax
* Both of these follow the same outline as the XML doc, starting with the root element and declaring all its child elements in a hierarchal order.
* Thus, they are matched as the schema syntax and DTD syntax will be the same (in terms of its logic when placing elements)

# Question 4 : Design XSLT

(Describe the major steps for designing the XSLT. Add screenshot of the output)

**Cezmi**

* Create xsl file and begin to add xsl:template and other sub-elements.
* Add html codes which will be used to display the xml file content.
* Use various xsl codes to extract data from xml and insert into the html tags.
* Add xsl link to xml file.



# Question 5 and 8: XPath and XSLT

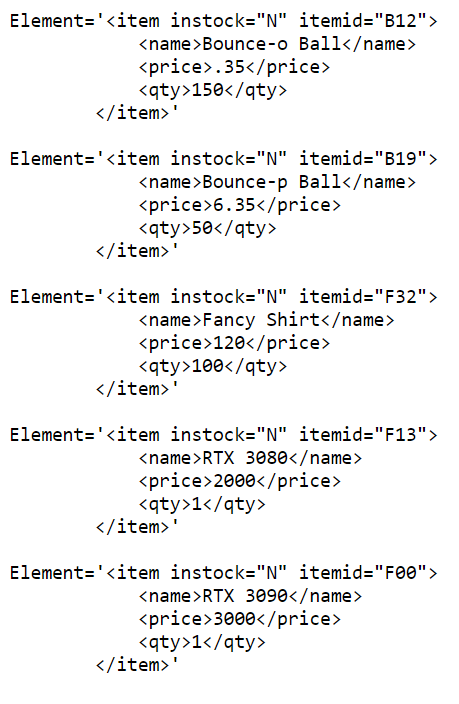
(Describe the major steps for designing the XPath and XSLT. How did you test the XPath? How did you use XPath in the XSLT?

Add screenshot of the XPath testing and the output of XSLT)

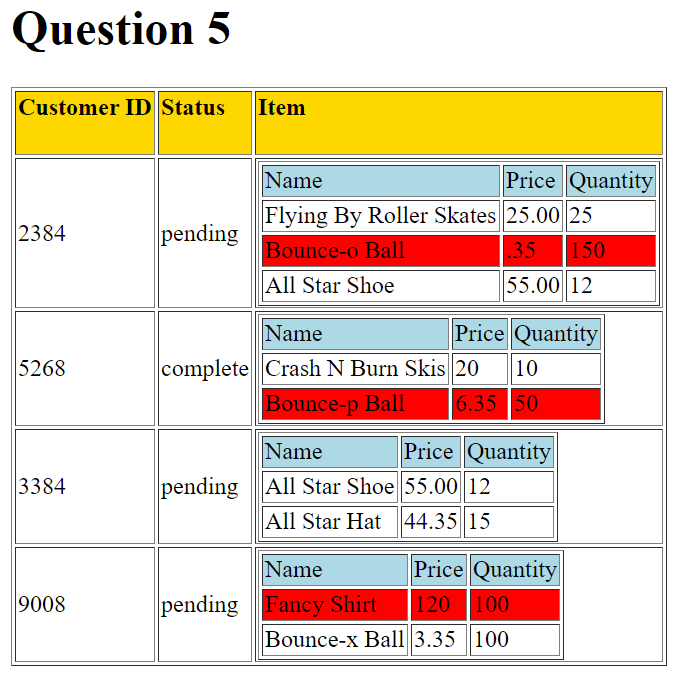
**Tim**

**XPath & XSLT Outputs**

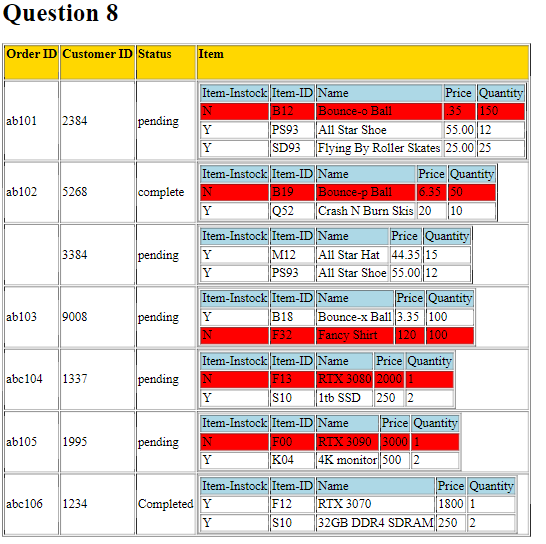
**XPATH (updated question 8 screen shot)**

* **//orders/order/item[@instock='N']**
* 

**Question 5 XSLT**



**Question 8 XSLT**



**(Describe the major steps for designing the XPath and XSLT. How did you test the XPath? How did you use XPath in the XSLT?**

**Designing XPath -** //orders/order/item[@instock='N']

When designing the XPath, you must consider the syntax or hierarchy of the XML doc. It is good practice to start with two // and with the root element of the XML doc itself.

* Like //orders

Next you need to work your way through the hierarchy to get the desired element / data you wish to display. For each step in the hierarchy in which you are moving requires a /

* Like //orders/order

The above example would display all the data held in the order element. You continue this pattern until you get to the item element that holds an attribute we want to test against.

* //orders/order/item

The above would return all the data held in the item elements for all orders, however we need to only show the ones that are not in stock. To do this we need to open up with square brackets and place the attribute we wish to test for inside, using @ to signify the attribute and placing the value we are testing for inside quotes.

* Like //orders/order/item[@instock='N']

The above will return all elements that are not in stock.

**Designing the XSLT**

When designing the XSLT docs for questions 5 and 8 you must take into consideration the structure of the XML doc as well. This is because you need to access/ reference them in a similar way to using an XPath. The first thing you need to do is link the XML doc to the XSLT doc you have created (blank xslt).

<?xml-stylesheet*type*="text/xsl"*href*="Q5.xsl"?>

In the above you place this line after declaring the XML doc. It tells the xml doc that styling will be applied and what xsl doc to reference. Now you can begin working on the XSLT doc which starts with the following lines.

<?xml*version*="1.0"*encoding*="UTF-8" ?>

<xsl:stylesheet *version*="1.0" *xmlns:xsl*="http://www.w3.org/1999/XSL/Transform">

These declare/define the xslt doc and then you can begin building it from here. The next this is to start adding elements, starting with xsl:template followed by its child elements. In the xslt doc you can use html vocab (elements) to define the style you want to display the data in. For example, you may want to create a head and body section like in and html doc.

<html>

         <head></head>

         <body></body

</html>

Next you can begin adding html elements like table and its sub elements (like tr td) to design the structure that the xml data will be displayed in. For example

<table *border*='1'>

                    <tr>

                        <td>

                            <h4>Customer ID</h4>

                        </td>

                        <td>

                            <h4>Status</h4>

                        </td>

                        <td>

                            <h4>Item</h4>

                        </td>

                    </tr>

Once you have designed the structure of your xslt doc you can begin extracting data from the XML doc using xsl codes within the html elements. You will have to understand the structure of the xml doc in order to extract the proper data. Additionally, you will need to leverage different xslt codes/elements to do work for you like for loops and when otherwise statements as will as defining what data to extract. For example, the following xslt elements…

<xsl:for-each *select*="//order">

<xsl:value-of *select*="customerid"/>

<xsl:choose>

<xsl:when *test*="@instock='N'">

<xsl:otherwise>

Once you have done the above in the proper way your data will be display as desired.

**Testing XPath**

Testing the XPath is done using <https://www.freeformatter.com/xpath-tester.html#ad-output>

However, you can test it yourself by comparing your XPath to the syntax of the XML file, as you to go through each element in the hierarchy in order without skipping over elements

* So, you would not do this //orders/item[@instock='N']
* This would give you errors as item is a child of order not orders

**Using XPath**

I was able to use a portion of the XPath (**item[@instock='N'])** in my XSLT as the comparer in a when otherwise statement/clause to give rows red back ground if they are not in stock and leave the row back ground as white if they are in stock. Additionally, it was useful to have an understanding of the XPath to know how to traverse the hierarchy within the XML doc.

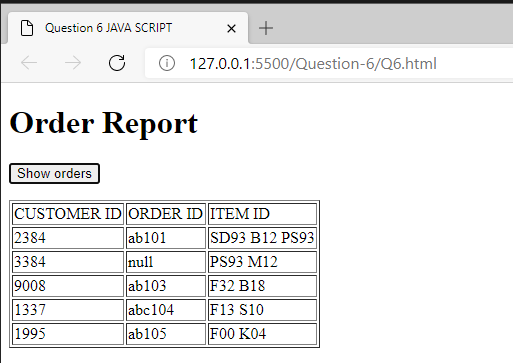
# Question 6 and 7: Use JavaScript to process XML data

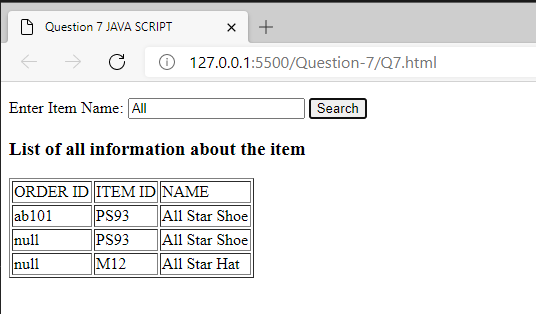
(Describe the major steps for designing the JavaScript function(s), how you test this program, add some screenshots of the output)

**Cezmi**

* Create an external script file and link this file with the html.
* Create codes to connect and extract information from xml file.
* Create a function to write extracted and filtered data into the html file.
* Use loops and if conditionals to search data in the xml file and filter data.
* Use html button to activate function.
* After all, I used live server to test codes.

Here are the screen shots of output





# Question 9

(Describe the major steps for designing the JavaScript function(s), how you test this program, add some screenshots of the output)

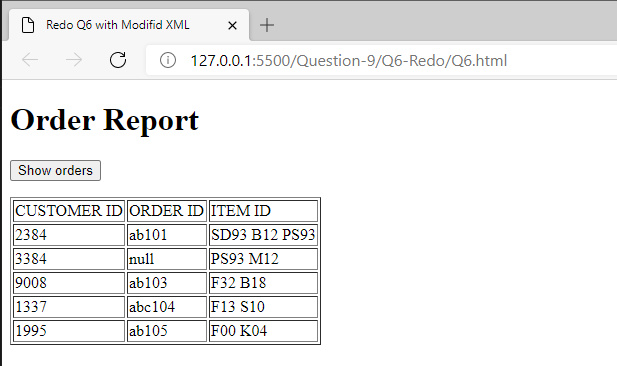
**Cezmi & Tim**

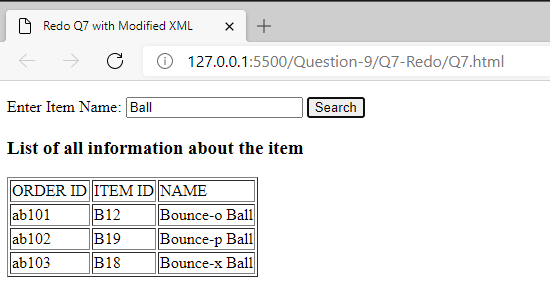
**A.**

**Redo Question 6 and Question 7 (Cezmi)**

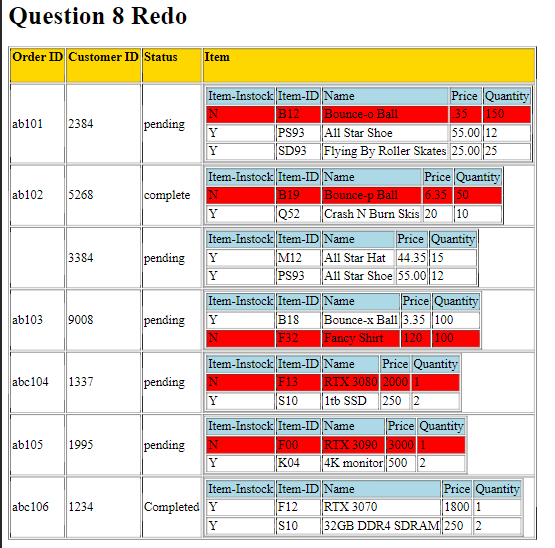
* Create an external script file and link this file with the html.
* Create codes to connect and extract information from xml file.
* Create a function to write extracted and filtered data into the html file.
* Use loops and if conditionals to search data in the xml file and filter data.
* Use html button to activate function.
* After all, I used live server to test codes.

Here are the screen shots of output





**Redo Question 8 (screen shot) (Tim)**



**B.** From your understanding as developer, explain your idea about coding/processing XML element vs attribute. Which one is easier to develop with JS or XSLT, process customerid as element-data or attribute-data? Explain your answer based on your observation through this project

**Tim (Cezmi agrees)**

I think it is easier manipulate data from an xml file using JavaScript as compared XSLT data manipulation. This is because JavaScript is easier for me to understand and manipulate data and can be written directly on the HTML page or on a separate JS file itself. This provides the developer more options in how to display and store the data. Instead of converting xml to xslt and then using html elements with predefined xsl codes, you can simply build the html and js in the same file and be more precise with what your functions are doing.

For example, when considering displaying the customerid as element-data or attribute-data I believe it is easier to develop and extract with JS as you simply reference that element by tag and use customerId[i].innerHTML for customer as an element (in a for loop) and order[i].getAttribute('customerid') respectively, allowing you to directly access that data without having to step down through the hierarchy (where errors can arise). Where as with xslt you need work you way through the entire hierarchy to access that data you want, for example moving from orders to /order and then accessing @customerid.

In general I found it more difficult working with XSLT to display that data in the way I wanted and I found the way you can manipulate it to be quite limited, whereas JS allows you to do everything xslt would do and more.

# Bonus question

(Describe the major steps for designing the XSLT/JavaScript, how you test this program, add some screenshots of the output)

# Summary

(Describe how did you divide the work, share your feedback about this project like new points that you learn, challenges, …)

We divided the work for the first six question between odds and evens, I took the first 3 odds and Cezmi the first three evens. Once part two was released, we decided to divide it based on the questions and work we have already done. Since Cezmi started working with JavaScript, and I xslt, we kept the work divided in that way. So Cezmi worked on question 7, I worked on 8 and we work on our respective parts for 9. We also decided that I would work on putting everything together in the word doc and organizing the project. In short, we decided to divide the work as evenly as possible, each keeping to their respective strengths.

For feedback, we had learned a lot. I learned how to work with attributes in XSLT and when otherwise statements, and how best to display the data in a clear and concise way. Cezmi learned various JavaScript functions and how to best work with loops inside loops.

For me, some challenges I faced where getting the proper rows to highlight red when not in stock and remain white when in stock, as well as working with the placement of when otherwise / learning there is no real if else for xslt. For Cezmi the major challenge was extracting the data need in question 7.