Group 17 - Online Retailing

- Pavel Petrukhin - 19328624
- Christiana Popoola - 19334280
- Pascal Raos - 19335585
- Oluwatumininu Ogunbadejo - 18328099
- David Olowookere - 19335061

What we need to change in our UML design

1. The associations we had on the class diagram stayed more or less the same, however they were represented via ids scheme in our XMLs and DTDs, which means that every class involved in XML got an extra id field and probably (depending on the associations) a list of id's to some other class or just one id, if the relationships is 1 to 1. We did not really reflect this on our class diagram because this can be assumed implicitly.
2. We deleted the moderator object from the shop, because we thought that only the moderator needs to know which shops they supervise, while it does not really matter for a particular shop to know who moderates it.
3. We also had to rethink our Ethics canvas. In this context, we understood that the sort by rating function might create bias towards sellers who got negative feedback for a reason other than the quality of their product e.g. views, gender identification and race. To tackle this issue, we agreed that an increased amount of moderation should be applied to the comments area as well as users should only be able to leave a comment if they actually bought the product. This could be checked via order reference number, for instance.

Allocation of work:

1. Pavel Petrukhin: Did Comment.xml, Comment.dtd, average rating query and sort by average rating query, formatted most of the DTDs and XMLs to make them work together.

2. Pascal Raos: Did XML and DTD documents for Seller, User and Customer, and wrote the XQuery for listAllCustomers.

3. Christiana Popoola: Wrote Product.xml, Order.xml and respective DTD documents. Wrote the XQuery functions for List products by price (ascending/descending) and List products of a specific size.

4. Oluwatumininu Ogunbadejo: XML and DTD: Shop.dtd, Shop.xml. XQuery: Get past orders, get total cost of wishlist.

5. David Olowookere: XML and DTD: Account.dtd, Account,xml. XQuery: List products which have price no larger than.

Strengths and Weaknesses of XML design and XQueries

In terms of the strengths, our XML design is in agreement with the UML design, because there is a more or less standardized way of converting UML to XML. We carefully defined our DTD files to make sure our XMLs are both flexible and valid. XML documents itself nicely and this enhanced even more by the way we formatted them. XQuery language is closer to usual programming languages as opposed to SQL, for instance, so it is easy to develop queries. The queries we developed support the essential use-cases of any online retail system.

However, there are a few drawbacks to all this, because otherwise, people would only be using XML. Firstly, XML documents might get verbose and especially with such a broad system like an online retailing website. Secondly, there are quite a few options for validating XMLs like DTDs, Schemas and etc, so it makes the learning process a bit more difficult compared to other database solutions such MongoDB and MSSQL. Finally, our XQueries are only reading the database and not modifying it anyhow, but we feel like modifying should be done via building the XML insertion in code and then pushing it into the relevant file in the database.

XML and DTD Documents

NB: ? indicates that the tag is not necessarily present in the xml file. We express references using id and refid attributes in respective tags.

User.xml:

User.xml describes the parent class of Customer and Seller, which has inherent traits the other subclasses possess, such as fullName, isLoggedIn attributes, and a reference to a linked Account by refid.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE databaseUser SYSTEM "User.dtd">

<databaseUser>
      <User id="u1">
            <User.fullName> David O.</User.fullName>
            <User.isLoggedIn> True </User.isLoggedIn>
            <User.account>
                  <Account refid="a7"/>
            </User.account>
```

```
      </User>
      <User id="u2">
            <User.fullName> Pavel P.</User.fullName>
            <User.isLoggedIn> False </User.isLoggedIn>
            <User.account>
                  <Account refid="a8"/>
            </User.account>
      </User>
      <User id="u3">

      <User.fullName> Mark Jacobs </User.fullName>
            <User.isLoggedIn> True </User.isLoggedIn>
            <User.account>
                  <Account refid="a9"/>
            </User.account>
      </User>
</databaseUser>
```

User.dtd:

User.dtd validates the format for the xml file. The '*' cardinality for User represents the presence of multiple potential users in the user database.

```
<!ELEMENT databaseUser (User*)>
<!ELEMENT User (User.fullName?,User.isLoggedIn?,User.account )>
<!ATTLIST User
 id CDATA #REQUIRED>
<!ELEMENT User.fullName (#PCDATA)>
<!ELEMENT User.isLoggedIn (#PCDATA)>
<!ELEMENT User.account (Account)>
<!ELEMENT Account EMPTY>
<!ATTLIST Account refid CDATA #REQUIRED>
```

Shop.xml:

Shop.xml describes the attributes of the name of the shop, the seller and the product being sold. It also possesses references from its subclasses, such as Product refid and Seller refid. This xml shows four different shops available on the retail website.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE databaseShop SYSTEM "Shop.dtd">
```

```
<databaseShop>
      <Shop id="s1">
            <Shop.name> AmazonReseller </Shop.name>
            <Shop.products>
                  <Product refid="p228"/>
                  <Product refid="p1757"/>
            </Shop.products>
            <Shop.owner>
                  <Seller refid="sl2"/>
            </Shop.owner>
      </Shop>
      <Shop id="s2">
            <Shop.name> AmazonReseller </Shop.name>
            <Shop.products>
                  <Product refid="p1753"/>
                  <Product refid="p1757"/>
            </Shop.products>
            <Shop.owner>
                  <Seller refid="sl2"/>
            </Shop.owner>
      </Shop>
      <Shop id="s3">
            <Shop.name>Heavy Gym</Shop.name>
            <Shop.products>
                  <Product refid="p991"/>
            </Shop.products>
            <Shop.owner>
                  <Seller refid="sl1"/>
            </Shop.owner>
      </Shop>
      <Shop id="s4">
            <Shop.name>GardenWorkshop</Shop.name>
            <Shop.products>
                  <Product refid="p123"/>
            </Shop.products>
            <Shop.owner>
                  <Seller refid="sl1"/>
            </Shop.owner>
      </Shop>
</databaseShop>
```

Shop.dtd:

Shop.dtd is used to validate the format for the xml document, Shop.xml. The '*' cardinality for Shop shows the presence of no shop or multiple shops in the shop database. The presence of '+' indicates that the shop has to have one or more products available.

```
<!ELEMENT databaseShop (Shop*)>
<!ELEMENT Shop (Shop.name, Shop.products+, Shop.owner)>
<!ATTLIST Shop id CDATA #REQUIRED>
<!ELEMENT Shop.name (#PCDATA)
>
<!ELEMENT Shop.products (Product*)>
<!ELEMENT Product EMPTY>
<!ATTLIST Product refid CDATA #REQUIRED>
<!ELEMENT Shop.owner (Seller)>
<!ELEMENT Seller EMPTY>
<!ATTLIST Seller refid CDATA #REQUIRED>
```

Seller.xml:

This xml Document describes a system containing the information of sellers on an online shopping system. Each seller has fullName, OnlineCheckoutID and isLoggedIn attributes with an account linked to each seller. Each seller has their own shops with their own respective products that they sell. Ref-ids are used to reference the specific data to other xml documents for Product and Shop

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE databaseSeller SYSTEM "Seller.dtd">
<databaseSeller>
      <Seller id="sl1">
            <User.fullname> Mark Jacobs </User.fullname>
            <User.isLoggedIn> True </User.isLoggedIn>
            <User.account>
                  <Account refid="a9"/>
            </User.account>
            <Seller.shops>
                  <Shop refid="s3"/>
                  <Shop refid="s4"/>
            </Seller.shops>
            <Seller.onlineCheckoutId> 22857
</Seller.onlineCheckoutId>
      </Seller>
      <Seller id="sl2">
            <User.fullname>David O.</User.fullname>
            <User.isLoggedIn>True</User.isLoggedIn>
            <User.account>
                  <Account refid="a7"/>
            </User.account>
```

```
          <Seller.shops>
               <Shop refid="s1"/>
               <Shop refid="s2"/>
          </Seller.shops>
          <Seller.onlineCheckoutId> 00157
</Seller.onlineCheckoutId>
     </Seller>
</databaseSeller>
```

Seller.dtd:

Seller.dtd is used to validate the format for the xml document. The use of '*' with regards to Seller and shops indicates the presence of potential Sellers within the database and multiple shops owned by each Seller.

```
<!ELEMENT databaseSeller (Seller*)>
<!ELEMENT Seller
(User.fullname?,User.isLoggedIn?,User.account,Seller.shops*,Seller
.onlineCheckoutId?)>
<!ATTLIST Seller id CDATA #REQUIRED >
<!ELEMENT User.fullname (#PCDATA)>
<!ELEMENT User.isLoggedIn (#PCDATA)>
<!ELEMENT User.account (Account)>
<!ELEMENT Account EMPTY>
<!ATTLIST Account refid CDATA #REQUIRED>
<!ELEMENT Seller.shops (Shop*)>
<!ELEMENT Shop EMPTY>
<!ATTLIST Shop refid CDATA #REQUIRED>
<!ELEMENT Seller.onlineCheckoutId (#PCDATA)>
```

Product.xml:

This xml document describes a system which contains the information on each product within a given shop. Each product has a product id (unique to it), name, sizes (that the product is available in), description (detailing attributes of product), price (in given currency) and references to comments (left by other users of the system). The product id is used as a reference by other other xml documents such as, shop and order.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE databaseProduct SYSTEM "Product.dtd">
<databaseProduct>
     <Product id="p1753">
```

```
           <Product.name>Denim Jacket</Product.name>
           <Product.size>S</Product.size>
           <Product.size>M</Product.size>
           <Product.size>L</Product.size>
           <Product.description>White denim jacket, metal buttons,
buttoned pockets</Product.description>
           <Product.price>37.99</Product.price>
           <Product.comments>
                 <Comment refid="c1"/>
           </Product.comments>
      </Product>
      <Product id="p1757">
           <Product.name>Bomber Jacket</Product.name>
           <Product.size>XL</Product.size>
           <Product.description>Black rain-resistant jacket,
zipped pockets</Product.description>
           <Product.price>23.99</Product.price>
           <Product.comments>
                 <Comment refid="c2"/>
                 <Comment refid="c3"/>
           </Product.comments>
      </Product>
      <Product id="p228">
           <Product.name>Echo Dot</Product.name>
           <Product.size>One Size </Product.size>
           <Product.description>Smart speaker with
Alexa</Product.description>
           <Product.price>30</Product.price>
           <Product.comments>
       <Comment refid="c4"></Comment>
       <Comment refid="c5"></Comment>
     </Product.comments>
      </Product>
      <Product id="p991">
           <Product.name>Dumbbell</Product.name>
           <Product.size>5 kg</Product.size>
           <Product.size>10 kg</Product.size>
           <Product.description>One-handed weight which comes in a
variety of sizes</Product.description>
           <Product.price>29.99</Product.price>
           <Product.comments></Product.comments>
      </Product>
      <Product id="p123">
           <Product.name>Trowel</Product.name>
           <Product.size>N/A</Product.size>
           <Product.description>used to dig up
topsoil</Product.description>
```

```
            <Product.price>19.99</Product.price>
            <Product.comments></Product.comments>
      </Product>
</databaseProduct>
```

Product.dtd:

This document, Product.dtd is used to validate the formatting for the xml document, Product.xml. The presence of '*' with regards to Product indicates the potential presence of multiple products within the database which would belong to one particular shop. + is used to indicate that there might be 1 or more sizes for the product.

```
<!ELEMENT databaseProduct (Product* )>
<!ELEMENT Product (Product.name, Product.size+,
Product.description, Product.price,Product.comments)>
<!ATTLIST Product id CDATA #REQUIRED>
<!ELEMENT Product.name (#PCDATA)>
<!ELEMENT Product.size (#PCDATA)>
<!ELEMENT Product.description (#PCDATA)>
<!ELEMENT Product.price (#PCDATA)>
<!ELEMENT Product.comments (Comment*)>
<!ELEMENT Comment EMPTY>
<!ATTLIST Comment refid CDATA #REQUIRED>
```

Order.xml:

This xml document describes a system which contains the information on each order that a specific user has made (through their account). Each order has an order id (unique to it), name, total sum, shipping address, billing address, date order (order was made), date shipped (order was dispatched), and order status (delivered, in transit, etc..). This document also contains references to customer ref id which indicates which customer (user) made the order, and a seller ref id which indicates which seller owned the shops from which the products in the order were made.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE databaseOrder SYSTEM "Order.dtd">
<databaseOrder>
      <Order id="O735">
            <Order.dateOrdered>17.01.20</Order.dateOrdered>
            <Order.dateShipped>21.01.20</Order.dateShipped>
            <Order.orderStatus>Delivered</Order.orderStatus>
            <Order.sumTotal>65.00</Order.sumTotal>
            <Order.billingAddress>123 O'Neill Street, Co.Wicklow,
Ireland</Order.billingAddress>
```

```
            <Order.shippingAddress>123 O'Neill Street, Co.Wicklow,
Ireland</Order.shippingAddress>
            <Order.customer>
                  <Customer refid="cs1"/>
            </Order.customer>
            <Order.seller>
                  <Seller refid="sl1"/>
            </Order.seller>
      </Order>
      <Order id="O736">
            <Order.dateOrdered>15.04.20</Order.dateOrdered>
            <Order.dateShipped>17.04.20</Order.dateShipped>
            <Order.orderStatus>Delivered</Order.orderStatus>
            <Order.sumTotal>43.76</Order.sumTotal>
            <Order.billingAddress>64 Zoo Lane, Co.Kildare,
Ireland</Order.billingAddress>
            <Order.shippingAddress>64 Zoo Lane, Co.Kildare,
Ireland</Order.shippingAddress>
            <Order.customer>
                  <Customer refid="cs1"/>
            </Order.customer>
            <Order.seller>
                  <Seller refid="sl2"/>
            </Order.seller>
      </Order>
      <Order id="O737">
            <Order.dateOrdered>13.09.20</Order.dateOrdered>
            <Order.dateShipped>17.09.20</Order.dateShipped>
            <Order.orderStatus>Delivered</Order.orderStatus>
            <Order.sumTotal>29.99</Order.sumTotal>
            <Order.billingAddress>90 Canary Walk, London, United
Kingdom</Order.billingAddress>
            <Order.shippingAddress>37 Rainbow Road, Co.Tipperary,
Ireland</Order.shippingAddress>
            <Order.customer>
                  <Customer refid="cs2"/>
            </Order.customer>
            <Order.seller>
                  <Seller refid="sl1"/>
            </Order.seller>
      </Order>
      <Order id="O738">
            <Order.dateOrdered>01.12.20</Order.dateOrdered>
            <Order.dateShipped>05.12.20</Order.dateShipped>
            <Order.orderStatus>In Transit</Order.orderStatus>
            <Order.sumTotal>173.77</Order.sumTotal>
            <Order.billingAddress>57 Trinity Way, Dublin 2,
Ireland</Order.billingAddress>
            <Order.shippingAddress>104 Lagan Street, New York, United
States of America</Order.shippingAddress>
            <Order.customer>
                  <Customer refid="cs2"/>
```

```
            </Order.customer>
            <Order.seller>
                  <Seller refid="sl2"/>
            </Order.seller>
      </Order>
      <Order id="O739">
            <Order.dateOrdered>01.12.20</Order.dateOrdered>
            <Order.dateShipped>05.12.20</Order.dateShipped>
            <Order.orderStatus>In Transit</Order.orderStatus>
            <Order.sumTotal>12288.90</Order.sumTotal>
            <Order.billingAddress>Trinity College, Dublin 2,
Ireland</Order.billingAddress>
            <Order.shippingAddress>Times Square, New York, United States
of America</Order.shippingAddress>
            <Order.customer>
                  <Customer refid="cs3"/>
            </Order.customer>
            <Order.seller>
                  <Seller refid="sl2"/>
            </Order.seller>
      </Order>
</databaseOrder>
```

Order.dtd:

This document, Order.dtd is used to validate the formatting for the xml document, Order.xml. The presence of '*' with regards to Order indicates the potential presence of multiple orders within the database which would have been made by a given customer. The document also outlines format for the references to both the customer and seller ref id which would correspond to the respective users in either Customer.xml database or Seller.xml database.

```
<!ELEMENT databaseOrder (Order*)>
<!ELEMENT Order (Order.dateOrdered, Order.dateShipped,
Order.orderStatus, Order.sumTotal,
      Order.billingAddress,
Order.shippingAddress,Order.customer,Order.seller)>
<!ATTLIST Order id CDATA #REQUIRED>
<!ELEMENT Order.dateOrdered (#PCDATA)>
<!ELEMENT Order.dateShipped (#PCDATA)>
<!ELEMENT Order.orderStatus (#PCDATA)>
<!ELEMENT Order.sumTotal (#PCDATA)>
<!ELEMENT Order.billingAddress (#PCDATA)>
<!ELEMENT Order.shippingAddress (#PCDATA)>
<!ELEMENT Order.customer (Customer) >
<!ELEMENT Customer EMPTY>
<!ATTLIST Customer refid CDATA #REQUIRED>
<!ELEMENT Order.seller (Seller)>
<!ELEMENT Seller EMPTY>
```

```
<!ATTLIST Seller refid CDATA #REQUIRED>
```

Customer.xml:

Customer.xml describes the database of Customers which each have a full name, isLoggedIn, and Account refid from the parent class ‘User’. Each customer then has their own preferred payment method and preferred currency.

The customer wishlist is a reference to the Product.xml file, referencing a product for each product on their wishlist. Similarly each customer has a record of their orders, with a reference to the order in the Order.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE databaseCustomer SYSTEM "Customer.dtd">
<databaseCustomer>
      <Customer id="cs1">
            <User.fullname> Sam Samuels </User.fullname>
            <User.isLoggedIn> True </User.isLoggedIn>
            <User.account>
                  <Account refid="a11"/>
            </User.account>
            <Customer.PrefCurrency>Euro</Customer.PrefCurrency>
            <Customer.PrefPayMethod> Visa </Customer.PrefPayMethod>
            <Customer.Wishlist>
                  <Product refid="p1753"/>
                  <Product refid="p1757"/>
            </Customer.Wishlist>
            <Customer.orders>
                  <Order refid="O735"/>
                  <Order refid="O736"/>
            </Customer.orders>
      </Customer>
      <Customer id="cs2">
            <User.fullname> Steve Stevenson </User.fullname>
            <User.isLoggedIn> True </User.isLoggedIn>
            <User.account>
                  <Account refid="a13"/>
            </User.account>
            <Customer.PrefCurrency> United States Dollar
</Customer.PrefCurrency>
            <Customer.PrefPayMethod> PayPal
</Customer.PrefPayMethod>
```

```
          <Customer.Wishlist>
               <Product refid="p228"/>
               <Product refid="p991"/>
               <Product refid="p123"/>
          </Customer.Wishlist>
          <Customer.orders>
               <Order refid="O737"/>
               <Order refid="O738"/>
          </Customer.orders>
     </Customer>
     <Customer id="cs3">
          <User.fullname> Vladimir Kozlov </User.fullname>
          <User.isLoggedIn> True </User.isLoggedIn>
          <User.account>
               <Account refid="a14"/>
          </User.account>
          <Customer.PrefCurrency>Rubbles</Customer.PrefCurrency>
          <Customer.PrefPayMethod> Visa </Customer.PrefPayMethod>
          <Customer.Wishlist>
               <Product refid="p228"/>
          </Customer.Wishlist>
          <Customer.orders>
      <Order refid="O739"/>
          </Customer.orders>
     </Customer>
</databaseCustomer>
```

Customer.dtd:

Customer.dtd is used to validate the format for Customer.xml. The cardinality of '*' used for Customer represents the potential for multiple customers within the customer database.

```
<!ELEMENT databaseCustomer (Customer*)>
<!ELEMENT Customer (User.fullname?,User.isLoggedIn?,User.account,

Customer.PrefCurrency?,Customer.PrefPayMethod?,Customer.Wishlist?,
Customer.orders)>
<!ATTLIST Customer
 id CDATA #REQUIRED >
<!ELEMENT User.fullname (#PCDATA)>
<!ELEMENT User.isLoggedIn (#PCDATA)>
<!ELEMENT User.account (Account)>
<!ELEMENT Account EMPTY>
<!ATTLIST Account refid CDATA #REQUIRED>
<!ELEMENT Customer.PrefCurrency (#PCDATA)>
<!ELEMENT Customer.PrefPayMethod (#PCDATA)>
```

```
<!ELEMENT Customer.Wishlist (Product*)>
<!ELEMENT Product EMPTY>
<!ATTLIST Product refid CDATA #REQUIRED>
<!ELEMENT Customer.orders (Order*)>
<!ELEMENT Order EMPTY>
<!ATTLIST Order refid CDATA #REQUIRED>
```

Comment.xml:

This document represents a comment which user left after buying the product. It has a unique id, rating attribute, description for additional details about the quality, some other info tags such as title, author and location, which are mandatory (no sign for cardinality used). This is done to make sure all comments are made by real people, who bought the product.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE databaseComment SYSTEM "Comment.dtd">
<databaseComment>
      <Comment rating="5" id="c1">
            <Comment.description>
                  <title>BORAT HERE</title>
                  <text>VERY NICE PRODUCT, BACK IN KAZAKHSTAN WE
DON'T HAVE SUCH</text>
                  <author>BORAT</author>
                  <location>ALMATY</location>
            </Comment.description>
            <Comment.upvotes>
                  10
            </Comment.upvotes>
            <Comment.downvotes>
                  2
            </Comment.downvotes>
            <Comment.userRating>
                  8
      </Comment.userRating>
            <Comment.product>
                  <Product refid="p1753"/>
            </Comment.product>
      </Comment>
      <Comment rating="3" id="c2">
            <Comment.description>
                  <title>NO GOOD WORK</title>
                  <text>VERY BAD QUALITY NO GOOD WORK</text>
                  <author>VASYA</author>
                  <location>SIBERIA</location>
            </Comment.description>
```

```
          <Comment.upvotes>
               100
          </Comment.upvotes>
          <Comment.downvotes>
               0
          </Comment.downvotes>
          <Comment.userRating>
               100
     </Comment.userRating>
          <Comment.product>
               <Product refid="p1757"/>
          </Comment.product>
     </Comment>
     <Comment rating="5" id="c3">
          <Comment.description>
               <title>SUPER NICE</title>
               <text>NO PROBLEMS AT ALL</text>
               <author>KOLYA</author>
               <location>SIBERIA</location>
          </Comment.description>
          <Comment.upvotes>
               0
          </Comment.upvotes>
          <Comment.downvotes>
               200
          </Comment.downvotes>
          <Comment.userRating>
               -200
          </Comment.userRating>
          <Comment.product>
               <Product refid="p1757"/>
          </Comment.product>
     </Comment>
     <Comment rating="4" id="c4">
          <Comment.description>
               <title>SUPER NICE, GOOD</title>
               <text>NO PROBLEMS AT ALL, though delivery was
long</text>
               <author>Grace</author>
               <location>Beijing</location>
          </Comment.description>
          <Comment.upvotes>
               123
          </Comment.upvotes>
          <Comment.downvotes>
               10
          </Comment.downvotes>
```

```
            <Comment.userRating>
                  113
            </Comment.userRating>
            <Comment.product>
                  <Product refid="p228"/>
            </Comment.product>
      </Comment>
      <Comment rating="2.3" id="c5">
            <Comment.description>
                  <title>AWFUL</title>
                  <text>Item was damaged</text>
                  <author>Bernadine</author>
                  <location>Paris, France</location>
            </Comment.description>
            <Comment.upvotes>
                  10
            </Comment.upvotes>
            <Comment.downvotes>
                  0
            </Comment.downvotes>
            <Comment.userRating>
                  10
            </Comment.userRating>
            <Comment.product>
                  <Product refid="p228"/>
            </Comment.product>
      </Comment>
</databaseComment>
```

Comment.dtd:

This file is used to validate Comment.xml, * means that the database can contain 0 or more products, no cardinality is used for other elements meaning they are required. Rating and id attributes are obviously required.

```
<!ELEMENT databaseComment (Comment* ) >
<!ELEMENT Comment
(Comment.description,Comment.upvotes,Comment.downvotes,Comment.use
rRating,Comment.product) >
<!ATTLIST Comment rating CDATA #REQUIRED>
<!ATTLIST Comment id CDATA #REQUIRED>
<!ELEMENT Comment.description (title,text,author,location)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT location (#PCDATA)>
```

```
<!ELEMENT Comment.upvotes (#PCDATA)>
<!ELEMENT Comment.downvotes (#PCDATA)>
<!ELEMENT Comment.userRating (#PCDATA)>
<!ELEMENT Comment.product (Product)>
<!ELEMENT Product EMPTY>
<!ATTLIST Product refid CDATA #REQUIRED>
```

Account.xml:

This file contains a database of accounts, with email addresses, usernames, passwords and shipping addresses of users. Each account is associated with one and only one user.

```
<?xml version='1.0' standalone="no"?>
<!DOCTYPE databaseAccount SYSTEM "Account.dtd">
<databaseAccount>
      <Account id="a7">

<Account.emailAddress>olowookd@tcd.ie</Account.emailAddress>
            <Account.username>olowookd</Account.username>
            <Account.password>notMyRealPassword</Account.password>
            <Account.shippingAddress>27 Pineapple Lane, Co.
Wexford</Account.shippingAddress>

<Account.cardDetails>12345678910111213141516</Account.cardDetails>
      </Account>
      <Account id="a8">

<Account.emailAddress>petrukhp@tcd.ie</Account.emailAddress>
            <Account.username>petrukhp</Account.username>
            <Account.password>PUTIN VODKA</Account.password>
            <Account.shippingAddress>Cremlin, Moscow,
Russia</Account.shippingAddress>
            <Account.cardDetails>CASH FOR THE
PACKAGE</Account.cardDetails>
      </Account>
      <Account id="a9">

<Account.emailAddress>flex@tcd.ie</Account.emailAddress>
            <Account.username>flexer228</Account.username>
            <Account.password>ajfjff11!</Account.password>
            <Account.shippingAddress>Vladimir,
Russia</Account.shippingAddress>

<Account.cardDetails>9999999900001111</Account.cardDetails>
      </Account>
      <Account id="a11">
```

```
<Account.emailAddress>sam.samuels@gmail.com</Account.emailAddress>
           <Account.username>sssam</Account.username>
           <Account.password>qwerty123!</Account.password>
           <Account.shippingAddress>Dublin,
Ireland</Account.shippingAddress>

<Account.cardDetails>9999222333221111</Account.cardDetails>
     </Account>

     <Account id="a13">

<Account.emailAddress>steve.steveson@gmail.com</Account.emailAddre
ss>
           <Account.username>sssteave</Account.username>
           <Account.password>trumpmylove1!</Account.password>
           <Account.shippingAddress>Alabama,
USA</Account.shippingAddress>

<Account.cardDetails>99992223332200011</Account.cardDetails>
     </Account>

  <Account id="a14">

<Account.emailAddress>kozloff@gmail.com</Account.emailAddress>
           <Account.username>kozloff</Account.username>
           <Account.password>uiiiieee1123</Account.password>
           <Account.shippingAddress>SPB,
Russia</Account.shippingAddress>

<Account.cardDetails>9989923332200011</Account.cardDetails>
     </Account>

</databaseAccount>
```

Account.dtd:

This file is used to validate Account.xml. * states that there might be zero or more accounts in the database, whereas no cardinality is used for other elements, meaning they are required as well as the id attribute.

```
<!ELEMENT databaseAccount (Account*)>
<!ELEMENT Account
(Account.emailAddress,Account.username,Account.password,Account.sh
ippingAddress,Account.cardDetails)>
<!ATTLIST Account id CDATA #REQUIRED>
<!ELEMENT Account.emailAddress (#PCDATA)>
```

```
<!ELEMENT Account.username (#PCDATA)>
<!ELEMENT Account.password (#PCDATA)>
<!ELEMENT Account.shippingAddress (#PCDATA)>
<!ELEMENT Account.cardDetails (#PCDATA)>
```

AverageRating.xq

NB: There are two queries in this file.
So here we have two queries, one takes the id of the product and returns its average rating and the second one takes a shop and lists products by their average rating in ascending/descending order. The query supports Search/View items use-case, which requires different kinds of sortings. Average rating function returns a number, while sort by average rating returns an array of compound elements, consisting of product name and average rating associated with it sorted in the specified order.

```
(:this file contains two queries getAverageRating and sortProducts by
average
  rating:)
declare function local:getAverageRating($productId as xs:string){
  let $commentIds:= for $p in doc("../XML/Product.xml")/
    databaseProduct/Product where $productId = $p/@id return $p/
      Product.comments/Comment

  let $numericalRatings:= for $commentId in $commentIds/@refid
    for $c in doc("../XML/Comment.xml")/databaseComment/Comment
      where $commentId = $c/@id return$c/xs:double(@rating)

  return avg($numericalRatings)
};

declare function local:getProduct($productId as xs:string){
  for $p in doc("../XML/Product.xml")/
    databaseProduct/Product where $productId = $p/@id return $p
};

declare function local:sortProductsByAverageRating($shopId as
xs:string,$order
  as xs:string){
  let $shop:= for $sh in doc ("../XML/Shop.xml")/databaseShop/Shop
  where $shopId = $sh/@id return $sh

  let $results:= for $p in $shop/Shop.products/Product
      return <entry>{local:getProduct($p/@refid)/Product.name,

<avgRating>{local:getAverageRating($p/@refid)}</avgRating>}</entry>

  let $sortedList:= if($order = "asc") then
    for $rs in $results order by $rs/avgRating  ascending return $rs
    else if($order = "dsc") then
```

```
      for $rs in $results order by $rs/avgRating      descending return
$rs

  return $sortedList
};

<results>{
<justAvgRating>{local:getAverageRating("p228")}</justAvgRating>,
<listDescending>{local:sortProductsByAverageRating("s1","dsc")}</listDesc
ending>,
<listAscending>{local:sortProductsByAverageRating("s1","asc")}</listAscen
ding>}
</results>
```

```
<results>
  <justAvgRating>3.15</justAvgRating>
  <listDescending>
    <entry>
      <Product.name>Bomber Jacket</Product.name>
      <avgRating>4</avgRating>
    </entry>
    <entry>
      <Product.name>Echo Dot</Product.name>
      <avgRating>3.15</avgRating>
    </entry>
  </listDescending>
  <listAscending>
    <entry>
      <Product.name>Echo Dot</Product.name>
      <avgRating>3.15</avgRating>
    </entry>
    <entry>
      <Product.name>Bomber Jacket</Product.name>
      <avgRating>4</avgRating>
    </entry>
  </listAscending>
</results>
```

GetAllCustomers.xq

The xquery functions work on the Seller and Customer xml files to return a list of customers associated with a particular seller. This query is used in ManageShop use-case, so that sellers can see who bought things in their shops.

```
declare function local:getAllCustomers($SellerId as xs:string){
  let $customerIds:= for $o in
doc("../XML/Order.xml")/databaseOrder/Order
  where $o/Order.seller/Seller/@refid = $SellerId return
$o/Order.customer/Customer/@refid

  let $result:=for $csId in distinct-values($customerIds)
```

```
    for $cs in
doc("../XML/Customer.xml")/databaseCustomer/Customer
      where $cs/@id = $csId return $cs

  return $result

};

local:getAllCustomers("sl1")
```

```
<Customer id="cs1">
  <User.fullname>Sam Samuels</User.fullname>
  <User.isLoggedIn>True</User.isLoggedIn>
  <User.account>
    <Account refid="a11"/>
  </User.account>
  <Customer.PrefCurrency>Euro</Customer.PrefCurrency>
  <Customer.PrefPayMethod>Visa</Customer.PrefPayMethod>
  <Customer.Wishlist>
    <Product refid="p1753"/>
    <Product refid="p1757"/>
  </Customer.Wishlist>
  <Customer.orders>
    <Order refid="O735"/>
    <Order refid="O736"/>
  </Customer.orders>
</Customer>
<Customer id="cs2">
  <User.fullname>Steve Stevenson</User.fullname>
  <User.isLoggedIn>True</User.isLoggedIn>
  <User.account>
    <Account refid="a13"/>
  </User.account>
  <Customer.PrefCurrency>United States Dollar</Customer.PrefCurrency>
  <Customer.PrefPayMethod>PayPal</Customer.PrefPayMethod>
  <Customer.Wishlist>
    <Product refid="p228"/>
    <Product refid="p991"/>
    <Product refid="p123"/>
  </Customer.Wishlist>
  <Customer.orders>
    <Order refid="O737"/>
    <Order refid="O738"/>
  </Customer.orders>
</Customer>
```

LessThanPrice.xq

This query returns products with price less than the specified limit, it is used in Search/View use-case.

```
declare function local:getCheaperThan($priceLimit){
  for $p in doc("../XML/Product.xml")/databaseProduct/Product
  where $p/Product.price<$priceLimit
  order by $p/Product.price ascending
  return <result>{$p/Product.name, $p/Product.price}</result>
};
```

```
local:getCheaperThan(30.00)
```

```
<result>
  <Product.name>Trowel</Product.name>
  <Product.price>19.99</Product.price>
</result>
<result>
  <Product.name>Bomber Jacket</Product.name>
  <Product.price>23.99</Product.price>
</result>
<result>
  <Product.name>Dumbbell</Product.name>
  <Product.price>29.99</Product.price>
</result>
```

PastOrders.xq

This xquery function, PastOrders.xq returns the past orders according to the date of purchase whether in ascending or descending order. It works on the Customer xml file to get the customer's order, and then it works on the Order xml file to get the date of each order. This query supports LogIn use-case, because at the time after the login user can have a look at their past order history.

```
declare function local:parseDate($date as xs:string){
  let $tokenized:= tokenize($date,"\.")

  return xs:integer($tokenized[3]||$tokenized[2]||$tokenized[1])
};

declare function local:getPastOrders($customerId as xs:string, $sort as
xs:string) {

    let $orders:= for $o in doc("../XML/Order.xml")/databaseOrder/Order
              where $o/Order.customer/Customer/@refid = $customerId
return
              <entity>

{$o,<parsedDate>{local:parseDate($o/Order.dateOrdered)}</parsedDate>}
              </entity>


    let $sortedOrders:= if($sort = "asc") then
```

```
    for $o in $orders order by $o/parsedDate    ascending return $o
    else if($sort = "dsc") then
      for $o in $orders order by $o/parsedDate  descending return $o

    return $sortedOrders

};


<results>
{
<parsedDate>{local:parseDate("01.12.20")}</parsedDate>,
<listDescending>{local:getPastOrders("cs2","dsc")}</listDescending>,
<listAscending>{local:getPastOrders("cs2","asc")}</listAscending>
}
</results>
```

```
<results>
  <parsedDate>201201</parsedDate>
  <listDescending>
    <entity>
      <Order id="0738">
        <Order.dateOrdered>01.12.20</Order.dateOrdered>
        <Order.dateShipped>05.12.20</Order.dateShipped>
        <Order.orderStatus>In Transit</Order.orderStatus>
        <Order.sumTotal>173.77</Order.sumTotal>
        <Order.billingAddress>57 Trinity Way, Dublin 2, Ireland</Order.billingAddress>
        <Order.shippingAddress>104 Lagan Street, New York, United States of America</Order.shippingAddress>
        <Order.customer>
          <Customer refid="cs2"/>
        </Order.customer>
        <Order.seller>
          <Seller refid="sl2"/>
        </Order.seller>
      </Order>
      <parsedDate>201201</parsedDate>
    </entity>
    <entity>
      <Order id="0737">
        <Order.dateOrdered>13.09.20</Order.dateOrdered>
        <Order.dateShipped>17.09.20</Order.dateShipped>
        <Order.orderStatus>Delivered</Order.orderStatus>
        <Order.sumTotal>29.99</Order.sumTotal>
        <Order.billingAddress>90 Canary Walk, London, United Kingdom</Order.billingAddress>
        <Order.shippingAddress>37 Rainbow Road, Co.Tipperary, Ireland</Order.shippingAddress>
        <Order.customer>
          <Customer refid="cs2"/>
        </Order.customer>
        <Order.seller>
          <Seller refid="sl1"/>
        </Order.seller>
      </Order>
      <parsedDate>200913</parsedDate>
    </entity>
  </listDescending>
```

ProdPriceAscDscSize.xq

NB: There are two queries in this file

The xquery ProdPriceAscDscSize.xq, works to provide the user with an ordered list of products based on the price of each product. The function can be used to produce a list of products with prices ascending or descending based on the input (asc or dsc). The function

takes in a reference to the shop ref id, which would provide us with the list of products to be ordered.

In this file, we also have a query which works to provide the user with a list of products (within a given shop) that are available in the given size. The function takes in a reference to the shop ref id and searches for any product that has the specified size (XL, L, M, etc…). The function that lists all the products that fulfill the stated criterion and returns the list to the customer in no specified order. This query supports Search/View items use-case.

```
(:this file contains two queries sortProductsByPrice,
listProductOfSpecifiedSize:)

declare function local:sortProductsByPrice($shopId as xs:string, $order
as xs:string) {
    let $productIds:= for $shs in
doc("../XML/Shop.xml")/databaseShop/Shop
        where $shopId = $shs/@id return $shs/Shop.products/Product/@refid

    let $products:= for $productId in $productIds
          for $p in doc ("../XML/Product.xml")/databaseProduct/Product
            where $productId = $p/@id return $p

    let $sortedProducts:= if($order = "asc") then
    for $p in $products order by $p/Product.price     ascending return $p
    else if($order = "dsc") then
      for $p in $products order by $p/Product.price   descending return
$p

    return $sortedProducts


};

declare function local:listProductOfSpecifiedSize($shopId as xs:string,
$size as xs:string) {
    let $productIds:= for $shs in
doc("../XML/Shop.xml")/databaseShop/Shop
        where $shopId = $shs/@id return $shs/Shop.products/Product/@refid

    let $products:= for $productId in $productIds
          for $p in doc ("../XML/Product.xml")/databaseProduct/Product
            where $productId = $p/@id return $p
    let $results:= for $p in $products
                where exists(index-of($p/Product.size, $size)) return $p


    return $results

};

<results>
{
```

```
<listDescending>{local:sortProductsByPrice("s1","dsc")}</listDescending>,
<listAscending>{local:sortProductsByPrice("s1","asc")}</listAscending>,
<availableInSize>{local:listProductOfSpecifiedSize("s2",
"M")}</availableInSize>
}
</results>
```

```
<results>
  <listDescending>
    <Product id="p228">
      <Product.name>Echo Dot</Product.name>
      <Product.size>One Size</Product.size>
      <Product.description>Smart speaker with Alexa</Product.description>
      <Product.price>30</Product.price>
      <Product.comments>
        <Comment refid="c4"/>
        <Comment refid="c5"/>
      </Product.comments>
    </Product>
    <Product id="p1757">
      <Product.name>Bomber Jacket</Product.name>
      <Product.size>M</Product.size>
      <Product.size>XL</Product.size>
      <Product.description>Black rain-resistant jacket, zipped pockets</Product.description>
      <Product.price>23.99</Product.price>
      <Product.comments>
        <Comment refid="c2"/>
        <Comment refid="c3"/>
      </Product.comments>
    </Product>
  </listDescending>
  <listAscending>
    <Product id="p1757">
      <Product.name>Bomber Jacket</Product.name>
      <Product.size>M</Product.size>
      <Product.size>XL</Product.size>
      <Product.description>Black rain-resistant jacket, zipped pockets</Product.description>
      <Product.price>23.99</Product.price>
      <Product.comments>
        <Comment refid="c2"/>
        <Comment refid="c3"/>
      </Product.comments>
    </Product>
    <Product id="p228">
      <Product.name>Echo Dot</Product.name>
      <Product.size>One Size</Product.size>
      <Product.description>Smart speaker with Alexa</Product.description>
      <Product.price>30</Product.price>
      <Product.comments>
        <Comment refid="c4"/>
```

```
  <availableInSize>
    <Product id="p1753">
      <Product.name>Denim Jacket</Product.name>
      <Product.size>S</Product.size>
      <Product.size>M</Product.size>
      <Product.size>L</Product.size>
      <Product.description>White denim jacket, metal buttons, buttoned pockets</Product.description>
      <Product.price>37.99</Product.price>
      <Product.comments>
        <Comment refid="c1"/>
      </Product.comments>
    </Product>
    <Product id="p1757">
      <Product.name>Bomber Jacket</Product.name>
      <Product.size>M</Product.size>
      <Product.size>XL</Product.size>
      <Product.description>Black rain-resistant jacket, zipped pockets</Product.description>
      <Product.price>23.99</Product.price>
      <Product.comments>
        <Comment refid="c2"/>
        <Comment refid="c3"/>
      </Product.comments>
    </Product>
  </availableInSize>
</results>
```

TotalCostOfWishlist.xq

This xquery function works on Customer and Product xml files. The Customer xml file returns the wishlist of a customer while the Product xml returns the prices of each product in the wishlist. And the function as a whole sums up all the prices of the products. This query supports Search/View items use-case and in particular Save/Bookmark/Wishlist option.

```
declare function local:getTotalCostOfWishlist($customerId as
xs:string) {

   let $products:= for $w in
doc("../XML/Customer.xml")/databaseCustomer/Customer
        where $customerId = $w/@id return
$w/Customer.Wishlist/Product/@refid

   let $prices:= for $id in $products
                     for $p in
doc("../XML/Product.xml")/databaseProduct/Product
                          where $id = $p/@id
                                  return
xs:double($p/Product.price/string())


   return sum($prices)
};

local:getTotalCostOfWishlist("cs1")
```

```
61.980000000000004
```