**XML Report**

Group 14: Online Retailer

Design Process:

When tasked with expanding upon our project to include XML, DTD and XQueries, we first thought it important to revisit our initial UML design to se if anything would need to be adjusted in order to facilitate our XML implementation.

We first visited our class diagram as that would serve as the basis for each of our XML files. When examining it, we picked out specific classes that would provide interesting XML implementations. However, we soon realized that we would have to expand upon some of our classes in order to have fully realized. We added quite a few elements which would allow us to create ATTLIST’s in our XML files and we also considerably expanded some classes such as Payment so that we were satisfied with their level of complexity. In the end, this allowed us to explore advanced data manipulation with our XQueries and brought our report to a higher level of quality.

When it came to implementing our XQueries, we visited our main Use Case diagram. We decided to add and modify some cases here as it lets us showcase more of our XML files through interesting XQueries, with the ability to sort in “Add and Sort Selected Items to Shopping Cart” and the creation of “View and Sort Shipments”.

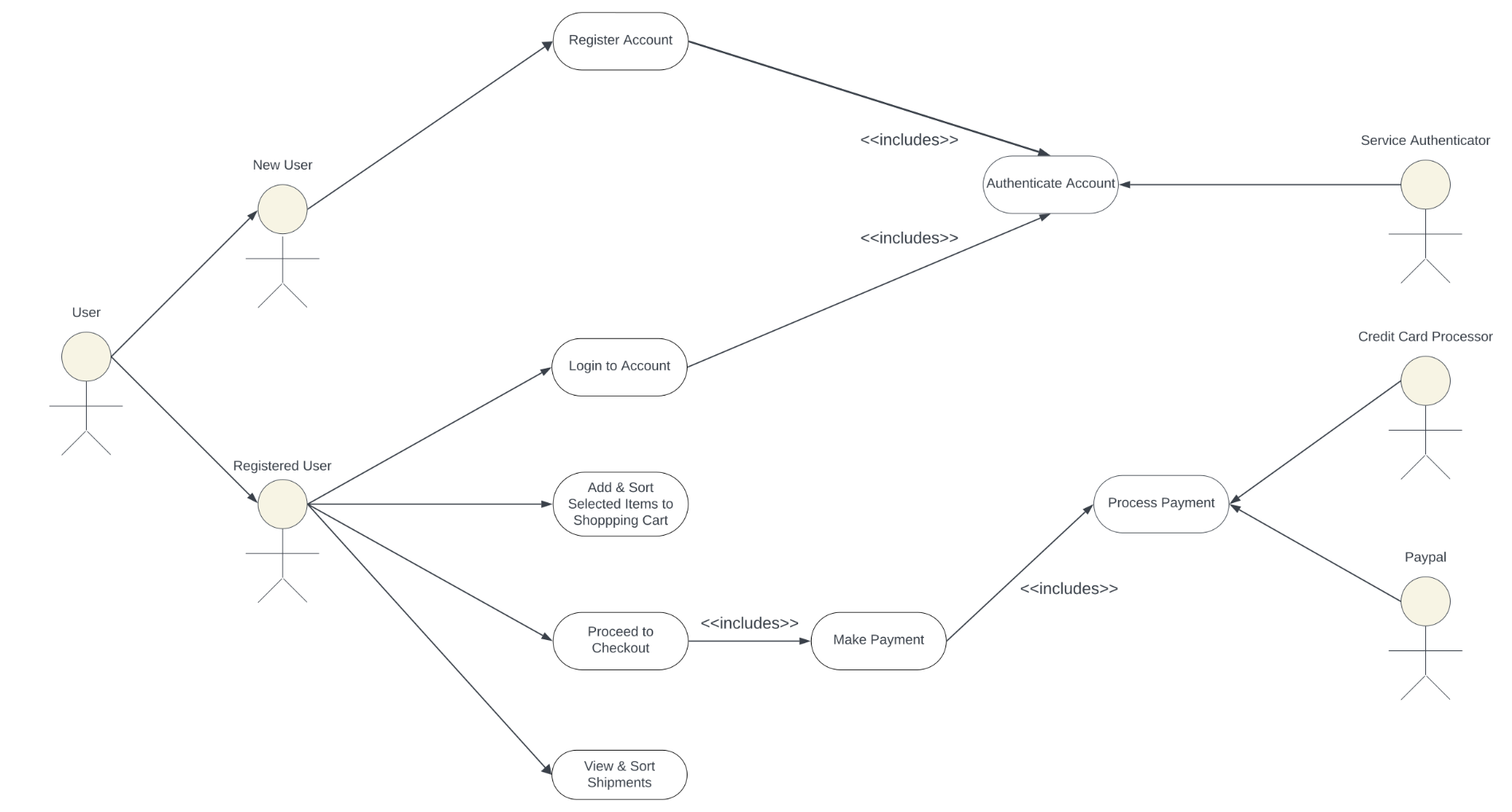
Present below are our revised diagrams:

Revised Classes Diagram:

A computer screen with many white boxes

Description automatically generated

Revised Use Case Diagram:



XML/DTD Files:

**Product:**

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

<!--This document describes the product aspect of the class diagram in our UML design. It describes the different aspects that encompass the product-->

<!DOCTYPE Product\_Info [ <!ELEMENT Product\_Info (Product)\*>

<!--Declaring the Document Type for Product-->

<!ELEMENT Product (productID, name, price, unit, stockQuantity, isAvailable)>

<!--Declaring different elements in Product-->

<!ELEMENT productID (#PCDATA)>

<!--productID: The ID number associated with a product in the system-->

<!ELEMENT name (#PCDATA)>

<!--name: The name of the product-->

<!ELEMENT price (#PCDATA)>

<!--price: The price of the product-->

<!ELEMENT unit (#PCDATA)>

<!--unit: The specific unit number associated with the product-->

<!ELEMENT stockQuantity (#PCDATA)>

<!--stockQuantity: The amount of the product that is available-->

<!ELEMENT isAvailable (#PCDATA)>

<!--isAvailable: Boolean indicator which shows if the product is available for purchase or not-->

<!ATTLIST Product category CDATA #REQUIRED >

<!--Product category: The specific category that the product falls under in the system-->

]>

<Product\_Info>

<Product category="Electronics">

<!--category: This product falls under the Electronics category in the system-->

<productID>751384013</productID>

<!--productID: The product's ID number is 751384013-->

<name>Two Slice Premium Toaster</name>

<!--name: The product's name is "Two Slice Premium Toaster"-->

<price>23.05</price>

<!--price: The product's price is €23.05-->

<unit>AZ7842</unit>

<!--unit: The product's unit identifier is AZ7842-->

<stockQuantity>343</stockQuantity>

<!--stockQuantity: There are 343 of this product in the system-->

<isAvailable>true</isAvailable>

<!--isAvailable: The product is available to purchase-->

</Product>

</Product\_Info>

**Shopping Cart:**

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

<!--This document describes the shopping cart aspect of the class diagram in our UML design. It contains information specific to the cart and user while also being able to contain multiple products within itself.-->

<!DOCTYPE ShoppingCart\_Info [ <!ELEMENT ShoppingCart\_Info (ShoppingCart)\*>

<!--Declaring the Document Type for Shopping Cart-->

<!ELEMENT ShoppingCart (cartID, userID, productList, totalPrice)>

<!--Declaring different elements in Shopping Cart-->

<!ELEMENT cartID (#PCDATA)>

<!--cartID: The ID number associated with a user's cart in the system-->

<!ELEMENT userID (#PCDATA)>

<!--userID: The user's associated ID number in the system-->

<!ELEMENT totalPrice (#PCDATA)>

<!--totalPrice: The total price of the various products in a user's shopping cart-->

<!ELEMENT productList (product)+>

<!--productList: A list containing one or more products (+) as without a product, there is nothing to view inside a shopping cart-->

<!ELEMENT product (productID, name, price, unit, stockQuantity, isAvailable)>

<!--Declaring different elements in product-->

<!ELEMENT productID (#PCDATA)>

<!--productID: The ID number associated with a product in the system-->

<!ELEMENT name (#PCDATA)>

<!--name: The name of the product-->

<!ELEMENT price (#PCDATA)>

<!--price: The price of the product-->

<!ELEMENT unit (#PCDATA)>

<!--unit: The specific unit number associated with the product-->

<!ELEMENT stockQuantity (#PCDATA)>

<!--stockQuantity: The amount of the product that is available-->

<!ELEMENT isAvailable (#PCDATA)>

<!--isAvailable: Boolean indicator which shows if the product is available for purchase or not-->

<!ATTLIST product category CDATA #REQUIRED >

<!--product category: The specific category that the product falls under in the system-->

]>

<ShoppingCart\_Info>

<ShoppingCart>

<cartID>48712</cartID>

<!--cartID: This specific shopping cart's ID is 48712-->

<userID>5042</userID>

<!--userID: This user's ID is 5042-->

<productList>

<!--productList: List of the products currently in the Shopping Cart-->

<product category="Electronics">

<!--category: This product falls under the Electronics category in the system-->

<productID>751384013</productID>

<!--productID: The product's ID number is 751384013-->

<name>Two Slice Premium Toaster</name>

<!--name: The product's name is "Two Slice Premium Toaster"-->

<price>23.05</price>

<!--price: The product's price is €23.05-->

<unit>AZ7842</unit>

<!--unit: The product's unit identifier is AZ7842-->

<stockQuantity>343</stockQuantity>

<!--stockQuantity: There are 343 of this product in the system-->

<isAvailable>true</isAvailable>

<!--isAvailable: The product is available to purchase-->

</product>

<product category="Electronics">

<!--category: This product falls under the Electronics category in the system-->

<productID>5274294</productID>

<!--productID: The product's ID number is 5274294-->

<name>Jet Ink Printer</name>

<!--name: The product's name is "Jet Ink Printer"-->

<price>107.50</price>

<!--price: The product's price is €107.50-->

<unit>JB874513</unit>

<stockQuantity>236</stockQuantity>

<!--stockQuantity: There are 236 of this product in the system-->

<isAvailable>true</isAvailable>

<!--isAvailable: The product is available to purchase-->

</product>

</productList>

<totalPrice>140.55</totalPrice>

<!--totalPrice: The total price of all items in the cart is €140.55-->

</ShoppingCart>

</ShoppingCart\_Info>

**User:**

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

<!-- This document describes the User aspect of the class diagram in our UML design.It contains information specific to each user. -->

<!DOCTYPE User\_Info [ <!ELEMENT User\_Info (User\*)>

<!-- Declaring the Document Type for User Information -->

<!ELEMENT User (userId, userName, password, address, eirCode, emailAd, login)>

<!-- Declaring different elements in User -->

<!ELEMENT userId (#PCDATA)>

<!-- userId: The ID number associated with a user in the system -->

<!ELEMENT userName (#PCDATA)>

<!-- userName: The user's name -->

<!ELEMENT password (#PCDATA)>

<!-- password: The user's password -->

<!ELEMENT address (#PCDATA)>

<!-- address: The user's physical address -->

<!ELEMENT eirCode (#PCDATA)>

<!-- eirCode: The Eir code for the user's address -->

<!ELEMENT emailAd (#PCDATA)>

<!-- emailAd: The user's email address -->

<!ELEMENT login (#PCDATA)>

<!-- login: Indicates whether the user is currently logged in or not (true/false) -->

<!ATTLIST User userState CDATA #REQUIRED>

<!-- userState: Describes the state of the user (new/registered) -->

]>

<User\_Info>

<User userState="new">

<!-- User element with userState attribute "new" -->

<userId>12345</userId>

<!-- userId: This user's ID is 12345 -->

<userName>J.K.Rowling</userName>

<!-- userName: The user's name is J.K.Rowling -->

<password>12345678</password>

<!-- password: The user's password is 12345678 -->

<address>Dublin2</address>

<!-- address: The user's address is Dublin2 -->

<eirCode>D02 PN40</eirCode>

<!-- eirCode: The postal code for the user's address is D02 PN40 -->

<emailAd>trinity@tcd.ie</emailAd>

<!-- emailAd: The user's email address -->

<login>true</login>

<!-- login: Indicates the user is currently logged in -->

</User>

</User\_Info>

**Shipment:**

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

<!-- This document describes the Shipment aspect of the system.It contains detailed information about various shipments.-->

<!DOCTYPE Shipment\_Info [ <!ELEMENT Shipment\_Info (Shipment)>

<!-- Declaring the Document Type for Shipment Information -->

<!ELEMENT Shipment (shipmentList)>

<!-- Declaring the Shipment element containing a shipmentList -->

<!ELEMENT shipmentList (shipmentDetails)+>

<!-- shipmentList: A list containing one or more shipmentDetails -->

<!ELEMENT shipmentDetails (shipmentId, shippingCompanyName, currentLocation, isShipped, shippingDate, expectShippedDate, calShippingFee)>

<!-- Declaring different elements in shipmentDetails -->

<!ELEMENT shipmentId (#PCDATA)>

<!-- shipmentId: The unique identifier for each shipment -->

<!ELEMENT shippingCompanyName (#PCDATA)>

<!-- shippingCompanyName: The name of the company handling the shipment -->

<!ELEMENT currentLocation (#PCDATA)>

<!-- currentLocation: The current location of the shipment -->

<!ELEMENT isShipped (#PCDATA)>

<!-- isShipped: Indicates whether the shipment has been shipped (true/false) -->

<!ELEMENT shippingDate (#PCDATA)>

<!-- shippingDate: The date when the shipment was or will be shipped -->

<!ELEMENT expectShippedDate (#PCDATA)>

<!-- expectShippedDate: The expected date for the shipment to be shipped -->

<!ELEMENT calShippingFee (#PCDATA)>

<!-- calShippingFee: The calculated shipping fee for the shipment -->

<!ATTLIST shipmentDetails shippingSpeedCategory CDATA #REQUIRED>

<!-- shippingSpeedCategory: The category indicating the speed of shipping (e.g.Slow, Fast, superFast) -->

]>

<Shipment\_Info>

<Shipment>

<shipmentList>

<shipmentDetails shippingSpeedCategory="Fast">

<!-- ShipmentDetails for a shipment shippingSpeedCategory: Indicates the speed of shipping 'Fast' -->

<shipmentId>54321</shipmentId>

<!-- shipmentId: The unique identifier for this particular shipment is 54321 -->

<shippingCompanyName>DHL</shippingCompanyName>

<!-- shippingCompanyName: The name of the company handling this shipment is DHL -->

<currentLocation>New York, The US</currentLocation>

<!-- currentLocation: The current location of the shipment is New York, The US -->

<isShipped>false</isShipped>

<!-- isShipped: A boolean value indicating the shipment has not been shipped -->

<shippingDate>2023-11-17</shippingDate>

<!-- shippingDate: The date when the shipment was or will be shipped is 2023-11-17 -->

<expectShippedDate>2023-12-14</expectShippedDate>

<!-- expectShippedDate: The expected date for the shipment to be delivered is 2023-12-14 -->

<calShippingFee>9.99</calShippingFee>

<!-- calShippingFee: The shipping fee for this shipment is 9.99 euros -->

</shipmentDetails>

<shipmentDetails shippingSpeedCategory="superFast">

<!-- ShipmentDetails for another shipment shippingSpeedCategory: Indicates the speed of shipping 'superFast' -->

<shipmentId>54322</shipmentId>

<!-- shipmentId: The unique identifier for this particular shipment is 54322 -->

<shippingCompanyName>AnPost</shippingCompanyName>

<!-- shippingCompanyName: The name of the company handling this shipment is AnPost -->

<currentLocation>Dublin 2, Ireland</currentLocation>

<!-- currentLocation: The current location of the shipment is Dublin 2, Ireland -->

<isShipped>true</isShipped>

<!-- isShipped: A boolean value indicating the shipment has been shipped -->

<shippingDate>2023-11-18</shippingDate>

<!-- shippingDate: The date when the shipment was shipped is 2023-11-18 -->

<expectShippedDate>2023-11-20</expectShippedDate>

<!-- expectShippedDate: The expected date for the shipment to be delivered is 2023-11-20 -->

<calShippingFee>12.99</calShippingFee>

<!-- calShippingFee: The shipping fee for this shipment is 12.99 euros -->

</shipmentDetails>

</shipmentList>

</Shipment>

</Shipment\_Info>

Payment:

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

<!--This document describes the payment aspect of the class diagram in our UML design. It contains information about the payment and the user.-->

<!DOCTYPE Payment\_Info [ <!ELEMENT Payment\_Info (Payment)\*>

<!--Declaring the Document Type for Payment-->

<!ELEMENT Payment (paymentMethod, transactionId, userId, amount, date, validity)>

<!--Declaring different elements in Payment-->

<!ELEMENT paymentMethod (#PCDATA)>

<!--paymentMethod: The method chosen by the customer to complete the purchase-->

<!ELEMENT transactionId (#PCDATA)>

<!--transactionID: The ID number associated with the transaction that is being completed in the system-->

<!ELEMENT userId (#PCDATA)>

<!--userId: The ID number associated with a user in the system-->

<!ELEMENT amount (#PCDATA)>

<!--amount: The amount of money that the transaction is going to take-->

<!ELEMENT date (#PCDATA)>

<!--date: The date of the transaction that is being authorised-->

<!ELEMENT validity (#PCDATA)>

<!--validity: To check if the payment has been valid-->

<!ATTLIST Payment processor CDATA #REQUIRED >

<!--Payment processor: The external system that enables financial transactions and its required for any payment to occur-->

]>

<Payment\_Info>

<Payment processor="Stripe">

<!--Payment processor: The payment processor is Stripe-->

<paymentMethod>CreditCard</paymentMethod>

<!--paymentMethod: The method chosen by the customer to complete the purchase is via credit card-->

<transactionId>92838282</transactionId>

<!--transactionID: The transaction ID is 92838282-->

<userId>12345</userId>

<!--userId: The user ID of this user is 12345-->

<amount>50.00</amount>

<!--amount: This transaction is going to take â‚¬50-->

<date>2023-11-23</date>

<!--date: This transaction is being made of 2023-11-23-->

<validity>true</validity>

<!--validity: This transaction has been deemed as valid by the system-->

</Payment>

</Payment\_Info>

CreditCard:

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

<!--This document describes the credit card aspect of the class diagram in our UML design. It contains information in relation to the credit card and the user.-->

<!DOCTYPE CreditCard\_Info [ <!ELEMENT CreditCard\_Info (CreditCard)\*>

<!--Declaring the Document Type for CreditCard-->

<!ELEMENT CreditCard (paymentMethod, transactionID, userID, amount, date, validity, cardType, cardNumber, expiryDate, cardHolderName, CVV, appApproval)>

<!--Declaring different elements in CreditCard-->

<!ELEMENT paymentMethod (#PCDATA)>

<!--paymentMethod: The method chosen by the customer to complete the purchase-->

<!ELEMENT transactionID (#PCDATA)>

<!--transactionID: The ID number associated with the transaction that is being completed in the system-->

<!ELEMENT userID (#PCDATA)>

<!--userId: The ID number associated with a user in the system-->

<!ELEMENT amount (#PCDATA)>

<!--amount: The amount of money that the transaction is going to take-->

<!ELEMENT date (#PCDATA)>

<!--date: The date of the transaction that is being authorised-->

<!ELEMENT validity (#PCDATA)>

<!--validity: To check if the credit card being used is valid for a transaction-->

<!ELEMENT cardType (#PCDATA)>

<!--cardType: The type of bank card that is being used for this transaction-->

<!ELEMENT cardNumber (#PCDATA)>

<!--cardNumber: The distinct number that belongs to every bank card-->

<!ELEMENT expiryDate (#PCDATA)>

<!--expiryDate: The date when this card cannot be used for a transaction as it has expired-->

<!ELEMENT cardHolderName (#PCDATA)>

<!--cardHolderName: Name of the owner of the bank card-->

<!ELEMENT CVV (#PCDATA)>

<!--CVV: The distinct three digit code on the back of the bank card that authorises the payment-->

<!ELEMENT appApproval (#PCDATA)>

<!--appApproval: The customer's authorisation of the transaction through the app of the bank they are making the transaction-->

<!ATTLIST CreditCard bank CDATA #REQUIRED >

<!--CreditCard bank: The bank associated with the credit card being used in the transaction and is required for the transaction to occur-->

]>

<CreditCard\_Info>

<CreditCard bank="BOI">

<!--CreditCard bank: The bank associated with the card is the Bank Of Ireland-->

<paymentMethod>CreditCard</paymentMethod>

<!--paymentMethod: The method chosen by the customer to complete the purchase is via credit card-->

<transactionID>92838282</transactionID>

<!--transactionID: The transaction ID is 92838282-->

<userID>12345</userID>

<!--userId: The user ID of this user is 12345-->

<amount>â‚¬50.00</amount>

<!--amount: This transaction is going to take â‚¬50-->

<date>2023-11-23</date>

<!--date: This transaction is being made of 2023-11-23-->

<validity>true</validity>

<!--validity: This bank card is valid to make a transaction-->

<cardType>Visa</cardType>

<!--cardType: This card is a Visa card-->

<cardNumber>4213478928791111</cardNumber>

<!--cardNumber: 4213478928791111 is the card number of the card being used-->

<expiryDate>09/28</expiryDate>

<!--expiryDate: The expiry date of the card is September 2028-->

<cardHolderName>John Smith</cardHolderName>

<!--cardHolderName: John Smith is the name of the card holder-->

<CVV>123</CVV>

<!--CVV: 123 is the CVV-->

<appApproval>yes</appApproval>

<!--appApproval: The card holder has approved the transaction inside the bank app-->

</CreditCard>

</CreditCard\_Info>

XQueries:

We created 9 XQuery files included FLWORs statements, function, selection, Array and map which we believe show the information in XML files and the processes in our case diagram and class diagram quite well.

**1: (Account Details)**

The **first** file user.xq show the user’s information corresponding to the process of login part of our online information system. This process is apart of the “Login to Account” use case.

**Code:**

declare variable $mypath:= "C:/Users/taoch/Desktop/XML/user.xml";

declare variable $mymap:= map

{

'Id':doc($mypath)/User\_Info/User/userId,

'Name':doc($mypath)/User\_Info/User/userName,

'eirCode':doc($mypath)/User\_Info/User/eirCode,

'password':doc($mypath)/User\_Info/User/password,

'address':doc($mypath)/User\_Info/User/address,

'email':doc($mypath)/User\_Info/User/emailAd,

'login\_state':doc($mypath)/user\_Info/User/login

};

let $m:=

doc($mypath)/User\_Info/User

where substring($m/emailAd,1,7) = "trinity"

return

<info>

{"This user has "}

{map:size($mymap)}

{" attributes."}

{map:find($mymap, 'Name')}

{map:get($mymap, 'eirCode')}

{"contain password?"}

{map:contains($mymap, 'password')}

{if ($mymap("login") = false()) then " A new user" else " A returning user"}

</info>

In this query, we locate the specific user by the substring of the email address first. Then we calculate the number of attributes by map size function and list the user’s basic information like name, Eircode by map function find and get. Analysing and generating brief result is the last part in our query via selection statement if and map function contains. The output result for this query is:

A white background with a black border

Description automatically generated with medium confidence

**2: (View & Sort Shipment Details)**

The **second** is named shipment.xq related to the xml file shipment.xml. And it involves to our class diagram shipment and the extra part outside the information system in the case diagram. Our purpose is to display the shipping date with descending order of shipping fees. This process is apart of the “View & Sort Shipment” use case.

**Code:**

for $i in

doc("C:/Users/taoch/Desktop/XML/shipment.xml")/Shipment\_Info/shipmentList/Shipment

let $m:=

doc("C:/Users/taoch/Desktop/XML/shipment.xml")/Shipment\_Info/shipmentList/Shipment

where $i/isShipped = true()

return

<location\_and\_price\_info>

{$i/currentLocation}

{if($m/calShippingFee > 10 and string($i/shippingCompanyName) = "AnPost") then "The price is larger than 10 is "}

{$i/calShippingFee}

</location\_and\_price\_info>,

<title lang = "en">

"Another query"

</title>,

for $x in

doc("C:/Users/taoch/Desktop/XML/shipment.xml")/Shipment\_Info/shipmentList/Shipment

order by $x/calShippingFee descending

return

<o\_shippingDate>

{"Novemmber"}

{number(substring($x/shippingDate,9))}

{$x/shipmentId}

</o\_shippingDate>

In this query, we find the data what we need by a where statement to filter the data and rearrange the shipping date by the key word order and generate the result output:

A screenshot of a computer

Description automatically generated

**3: (View Products in Shopping Cart)**

The **third** one is aimed at the process of selecting goods before the payment completed. The concrete class is ShoppingCart with the xml file ShoppingCart.xml. This process is contained within “Add & Sort Selected Items to Shopping Cart”.

**Code:**

for $j in

doc("C:\Users\taoch\Desktop\XML\ShoppingCart.xml")/ShoppingCart\_Info/ShoppingCart/productList/product

where ($j/price) > 70.5 and ($j/price) < 200.5

return

<product\_name\_and\_price>

{$j/name}

{$j/price}

</product\_name\_and\_price>

In this query, we track the core factor – price shown by the for clause and a where statement to filter the information between the particular price intervals. The output is:

A white rectangular object with blue text

Description automatically generated

**4: (Get Product Information)**

The **fourth** query Product.xq gets the unit label of a product in an information array based on the product class. This information is visible through “Add & Sort Selected Items to Shopping Cart”.

**Code:**

let $x:=

doc("C:/Users/taoch/Desktop/XML/Product.xml")/Product\_Info/Product

let $my\_array:=[$x/productID,$x/unit,$x/name, $x/name, $x/stockQuantity/price]

return

<Info>

{array:get(

$my\_array, 2

)}

</Info>

We choose to create an array inside the let clause and get the unit label information by array get function. The output shows:

A white rectangular object with blue and white lines

Description automatically generated

**5 & 6: (Get Payment Information)**

The next two queries **(5 & 6)** point at the payment process with two class in the inheritance relation. First, we do a query for the xml file payment based on the superclass payment. We expect to generate the result including one unique attribute – userID and the transaction serial number. These queries are apart of the “Make Payment” use case.

**Code:**

let $i:=

doc("C:\Users\taoch\Desktop\XML\Payment.xml")/Payment\_Info/Payment

let $my\_Array:=[$i/paymentMethod, $i/userId, $i/transactionId]

return

<transactionId>

{

array:head($my\_Array),

array:foot($my\_Array)

}

</transactionId>

An embedded array is set in the let clause that includes three element: paymentMethod, userId and transactionId. Then we use the head and foot function to generate the following output:

A white rectangular object with a black border

Description automatically generated

**7: (Get Credit Card Information)**

In our **seventh** query, we query the CreditCard.xml file which represents one of the payment methods in our case diagram. This query is contained within the “Make Payment” use case visible in our diagrams.

**Code:**

for $x in

doc("C:\Users\taoch\Desktop\XML\CreditCard.xml")/CreditCard\_Info/CreditCard

return

<CreditCard\_Info>

{$x/cardNumber}

{$x/appApproval}

{$x/cardType}

{$x/CVV}

</CreditCard\_Info>

We select to list all of the natural attributes from the subclass CreditCard and the output result is:

A white background with a blue and red border

Description automatically generated with medium confidence

**8: (Gets Information Across Payment & Credit Card)**

The last three queries **(8, 9 & 10)** combine with at least two different xml files each. The first one is called union\_CreditCard\_And\_Payment.xq to connect CreditCard.xml and Payment.xml with the key unique attribute userId. So, this query is still in the payment procedure of out online system consists of two class Payment and CreditCard. We expect to generate the cardHolderName, cardType and TransactionId. This first query is associated with the “Make Payment” use case.

**Code:**

for $i in

doc("C:\Users\taoch\Desktop\XML\CreditCard.xml")/CreditCard\_Info/CreditCard

let $j:=

doc("C:\Users\taoch\Desktop\XML\Payment.xml")/Payment\_Info/Payment

where $i/payment.userId = $j/userId

return

<payment\_method\_creditcard>

{$i/cardHolderName}

{$i/cardType}

{$j/transactionId}

</payment\_method\_creditcard>

One for clause and one let clause with the where statement to judge the equal relation of the userId from two xml files. The result output is:

A white background with blue and red lines

Description automatically generated

**9: (Getting Price Associated With ProductID)**

The next one is for querying the price of product by entering the productID. The corresponding classes are Product and ShoppingCart. This query is contained within “Add & Sort Selected Items to Shopping Cart”.

**Code:**

declare function local:simple\_lookup($t as xs:string)

{

for $s in

doc("C:\Users\taoch\Desktop\XML\Product.xml")/Product\_Info/Product.category

union

doc("C:\Users\taoch\Desktop\XML\ShoppingCart.xml")/ShoppingCart\_Info/ShoppingCart/productList/product

where $t = string($s/productID)

return

<price>

{$s/string(price)}

</price>

};

local:simple\_lookup("751384013")

In this query, we use a local function with a parameter string denoted by t which is the productid we enter in the system. And if the search can match the data in the database, it will return the product price. The example output is:

A white rectangular object with a black border

Description automatically generated

**10: (Total Price of Product’s Stock)**

The last query is called triple\_query.xq. We pursue the result which can show different information between three different xml files with some connected points from them. This final query combines two use cases, both “Add & Sort Selected Items to Shopping Cart” and “Make Payment”.

**Code:**

declare variable $mypath1:= "C:/Users/taoch/Desktop/XML/Product.xml";

declare variable $mypath2:= "C:/Users/taoch/Desktop/XML/CreditCard.xml";

declare variable $mypath3:= "C:/Users/taoch/Desktop/XML/ShoppingCart.xml";

declare variable $productID:= doc($mypath1)/Product\_Info/Product/productID;

declare variable $stockQuantity:= doc($mypath1)/Product\_Info/Product/stockQuantity;

declare variable $price:= doc($mypath1)/Product\_Info/Product/price;

declare variable $my\_array1:= [$productID, $stockQuantity, $price];

for $i in

doc($mypath2)/CreditCard\_Info/CreditCard

where $i/appApproval = "yes"

return

<total\_price\_info>

{

$my\_array1(3) \* $my\_array1(2)

}

</total\_price\_info>,

for $j in

doc($mypath3)/ShoppingCart\_Info/ShoppingCart/productList/product

where $j/productID = $my\_array1(1)

return

<name\_and\_state>

{$j/name}

{if($j/isAvailable = true()) then "This product is available"}

</name\_and\_state>

A white background with black text

Description automatically generatedWe declare seven variables outside the two for clauses. And the first for clause to return the total price of our products. And the second one shows the state of the goods. We select to access the array directly without any embedded functions which results in the following output:

Strengths & Weaknesses:

* We strongly believe that the primary strength in our XML implementation lies in our precision. Each class we chose to adapt was translated perfectly into our XML & DTD files, which the allocation of ATTLISTS & Elements being astutely chosen in order to create the best implementation we could.
* We believe another key strength in our implementation to be the addition of attributes to our classes which helped to expand their functionality. Attributes such as ‘userState’ and ‘shippingSpeedCategory’ allowed us to create an additional layer of complexity upon our design which we believe elevated it to a higher standard.
* Both of the above strengths lead to our final one, being our various XQueries. By initially creating precise and well-designed XML/DTD files, we were enabled to create truly rich and complex queries which showcased the breadth of our design. The additional attributes allowed us to further facilitate our advanced data manipulation and create various queries that we’re truly proud of
* However, we believe that one of our weaknesses is that our chosen classes didn’t fully demonstrate the breadth of our design. Particularly, the “Authenticate Account” use case was a core tenet of our initial design, yet we failed to represent it when it came to designing our XML implementation. By including this and a few other cases, we believe our design as a whole would be stronger.
* A minor weakness that we believe we have is that our XQuery code itself isn’t commented on so at first glance, it may be convoluted and hard to understand. However, we believe that the descriptions we provided in this document help to alleviate this slightly by explaining them and making them understandable to outside observers.

Tasks:

Each week during our designated meetings, each member of the team pooled together our ideas and offered each other feedback on our designs when applicable.

When deciding upon who should be delegated which tasks, we wanted to ensure the workload was split evenly and fairly.

Flynn’s role was the creation of the Product & Shopping Cart XML/DTD files, commenting on its elements and attributes and ensuring its validity. He brought our numerous files together and created this report. He also wrote our Strengths and Weakness along with Shengxin.

Shengxin’s role was the creation of the User & Shipment XML/DTD files, commenting on its elements and attributes and ensuring its validity. He also wrote the Strengths and Weaknesses along with Flnn.

William’s role was the creation of the CreditCard & Payment XML/DTD files, commenting on its elements and attributes and ensuring its validity.

Tao was solely responsible for the creation of the XQueries, ensuring that each of the requirements for them were met.