XML Report

A Comprehensive Exploration of XML, XPath, and XQuery

Liam Power

November 29, 2024

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

This document is submitted as part of the coursework for CSU22041 Information Management I

Team Members

Finn Clancy 23375792

Robin Schulz 22337984

Liam Power 23374796

Rachel Ranjith 23363463

Nikolaos Mavrias 24372713

Roisin Smith 23373110

Noah Scolard 23374057

1 Approach

1.1 Adjustments for Assignment Requirements

Some classes needed adjusting to meet the assignment requirements of having six elements in each document. We decided to add others for depth and understanding of the whole design. The events document was missing two elements, so we added `manager` and `short description`. Similarly, the artwork document was missing two elements, so we added `category` and `description`. This allowed us to create detailed XML documents without losing the general idea of our UML design.

1.2 XQuery Ideas

When coming up with ideas for interesting XQueries, we thought it would be beneficial to have a query to find out how many pieces of art the gallery currently has that fit in a specific category, such as "Post-Impressionism." This query will make it easier for curators to gauge if there are enough pieces of art available for their desired exhibition. This update forced us to adjust our artwork class to include a category variable to match the category elements in the XML DTD.

Another idea for an interesting XQuery was to have different ways of structuring important events. An event ID and a manager were mentioned. The ID existed in the class diagram, but the manager did not. By adding this, we not only created the XML documents and their XQueries according to the assignment guidelines but also made our use cases more comprehensive. For instance, in this example, the relationship between external partners, events, and the administration of the Art Gallery was strengthened by adding a gallery manager overseeing the events.

1.3 Artist Class Adjustments

We added the following variables to the `Artist` class to ensure it matched the elements in the Artist XML DTD: `deathYear`, `specialty`, and `placeOfStudy`.

1.4 Curator Use Case

We also added a use case for the Curator called "Get Artwork For Each Artist," which will make the Curator's workflow more efficient. This allows them to gather information for potential exhibitions quickly or respond to inquiries from auctioneers, museums, and other galleries.

1.5 Group Processes

We held group meetings during the tutorial times to coordinate our efforts and ensure everyone was on the same page regarding project tasks. Meetings were scheduled during and after lecture times to maximize participation and accommodate different schedules. We brainstormed possible XML documents and XQueries to explore various approaches and solutions for our project.

The class diagram dimension was adjusted for the XML tasks to better align with the project requirements and improve design accuracy. Tasks were allocated among team members, and progress was discussed regularly to track our development and address any issues promptly. There was in-person collaboration on XML documents, which facilitated direct communication and quick problem-solving. Virtual collaboration occurred through a GitHub repository, allowing us to share updates efficiently and work asynchronously.

2 Strengths and Weaknesses Analysis

2.1 XML

2.1.1 Strengths

- We used a standard to create our documents, ensuring consistency across the project.

- The individual XML documents were evenly distributed among team members, promoting balanced workloads.
- The use of modular XML designs allows for easier future updates or modifications.

2.1.2 Weaknesses

- Time constraints during the design phase limited our ability to fully evaluate which classes would work best as XML documents.
- Although we gained significant knowledge, not all team members could immediately understand the concepts due to the lack of centralized documentation.
- The large number of Classes in the Class Diagram made it challenging to decide which ones to convert into XML files.

2.2 XQuery

2.2.1 Strengths

- Consistent use of variables, such as dates and prices, across different queries ensured clarity and standardization.
- XQueries were assigned based on prior involvement with specific XML documents, keeping team members focused on familiar Classes.
- The XQueries spanned multiple XML documents, providing a rich and interconnected dataset for queries.

2.2.2 Weaknesses

- The large number of XML files created made it difficult to decide which ones to include in the XQueries.
- While designing XQueries, parts of the Class Diagram had to be reconfigured to connect certain Classes, introducing additional complexity.
- Traversing different Classes with XQueries was challenging because different team members used inconsistent element names for attributes like dates and prices.

3 XQueries

3.1 Query for Total Artworks Per Artist

- **Identification of the UML Use Case:** Supports the "Curators" use case.
- **Description of the Purpose:** This query allows curators to determine how many pieces of art the gallery holds for each artist. It helps in planning exhibitions for specific anniversaries and assessing whether additional pieces need to be sourced from other institutions.

Example Output:

```
<artistTotal>
    <name>Vincent van Gogh</name>
    <totalArtworks>3</totalArtworks>
</artistTotal>
```

3.2 Query for Events Managed by a Specific Manager

- **Identification of the UML Use Case:** Supports the "Creates Event" use case.
- **Description of the Purpose:** This query identifies all events managed by a specific manager based on their first name. It provides a comprehensive list of events for planning and oversight purposes.

Example Output: Input: `Robin`

```
<is_manager_for>
    <eventName>Historical Easter Getaway</eventName>
</is_manager_for>
<is_manager_for>
    <eventName>Boxing Day Charity</eventName>
</is_manager_for>
<is_manager_for>
    <eventName>Van Gogh Week</eventName>
</is_manager_for>
<is_manager_for>
    <eventName>Photography Festival</eventName>
</is_manager_for>
<is_manager_for>
    <eventName>Sponsor Gratitude Evening</eventName>
</is_manager_for>
```

Input: `Liam`

```
<is_manager_for>
    <eventName>Children's Art Workshop</eventName>
</is_manager_for>
```

3.3 Query for Event Duration

- **Identification of the UML Use Case:** Supports the "Creates Event" use case.
- **Description of the Purpose:** This query calculates and outputs the duration of a specified event in days, including the specific dates it spans.

Example Output: Input: `Van Gogh Week`

```
Van Gogh Week lasts 7 days from the 24th to the 30th.
```

Input: `Sponsor Gratitude Evening`

```
Sponsor Gratitude Evening lasts 1 day and is on the 12th.
```

3.4 Query for Membership Revenue

- **Identification of the UML Use Case:** Supports the "Purchase Membership" use case.
- **Description of the Purpose:** This query calculates the total revenue generated from each type of membership. It helps assess revenue performance and informs membership-related decision-making.

Example Output:

```
<TotalMembershipRevenue type="Patron_Director">5000.0</TotalMembershipRevenue>
<TotalMembershipRevenue type="Patron_Dargan">10000.0</TotalMembershipRevenue>
<TotalMembershipRevenue type="Patron_Curator">4500.0</TotalMembershipRevenue>
```

3.5 Query for Artworks by Category

- **Identification of the UML Use Case:** Supports the "Creates Event" use case.
- **Description of the Purpose:** This query allows curators to see the number of artworks available in each category, helping them organize exhibitions based on specific themes or styles.

Example Output:

```
<category>
    <name>Post-Impressionism</name>
    <artworkCount>3</artworkCount>
</category>
<category>
    <name>Surrealism</name>
    <artworkCount>4</artworkCount>
</category>
<category>
    <name>Expressionism</name>
    <artworkCount>1</artworkCount>
</category>
<category>
    <name>Impressionism</name>
    <artworkCount>1</artworkCount>
</category>
```

3.6 Query for Feedback from Non-Paying Visitors

- **Identification of the UML Use Case:** Supports the "Feedback" use case.
- **Description of the Purpose:** This query retrieves feedback submitted by non-paying visitors, potentially identifying spam or malicious reviews.

Example Output:

```
<FeedbackFromNonPayingUser>
    <visitorID>V123</visitorID>
    <rating>1</rating>
    <comments>The reception staff is rude, paintings boring</comments>
    <dateSubmitted>2023-11-14</dateSubmitted>
    <ratingValue>1</ratingValue>
</FeedbackFromNonPayingUser>
```

3.7 Query for Donor Contributions

- **Identification of the UML Use Case:** Supports the "Donate" use case.
- **Description of the Purpose:** This query retrieves and displays donations made by a specific donor, including details such as amount, currency, and donation type.

Example Output:

```
<DonorDonation>
    <DonationID>D001</DonationID>
    <Amount>100.00</Amount>
    <Currency>EUR</Currency>
    <DonationType>Individual</DonationType>
</DonorDonation>
```

3.8 Query for Total Revenue on a Specific Date

- **Identification of the UML Use Case:** Supports the "Donations," "Memberships," and "Tickets" use cases.
- **Description of the Purpose:** This query calculates total revenue generated from donations, memberships, and tickets on a specific date.

Example Output: Input: `2023-11-15`

```
<TotalRevenue>The total revenue for 2023-11-15 is $8,390.00!</TotalRevenue>
```

4 XML Documents

4.1 Artist

- Finn

```
<?xml version="1.0" ?>

<!--
This XML document stores information about artists, including their biographical details,
artistic specialty, and place of study. It can be used in applications or queries to
retrieve artist-specific data.
-->

<!DOCTYPE artists [

  <!--
  The "artists" element contains one or more "artist" elements,
  hence the cardinality is "+" (at least one artist is required).
  -->
  <!ELEMENT artists (artist+)>

  <!--
  Each "artist" includes details about their name, nationality, birth and death years,
  artistic specialty, and place of study. These elements are all required,
  so no optional or zero cardinality is specified.
  -->
  <!ELEMENT artist (name, nationality, birthYear, deathYear, specialty, placeOfStudy)>

  <!ELEMENT name (#PCDATA)> <!-- The artist's name -->
  <!ELEMENT nationality (#PCDATA)> <!-- The artist's nationality -->
  <!ELEMENT birthYear (#PCDATA)> <!-- The year the artist was born -->
  <!ELEMENT deathYear (#PCDATA)> <!-- The year the artist passed away -->
  <!ELEMENT specialty (#PCDATA)> <!-- The artist's area of specialization -->
  <!ELEMENT placeOfStudy (#PCDATA)> <!-- The institution where the artist studied -->
]>

<artists>
  <!-- An example of an artist's information -->
  <artist>
    <name>Vincent van Gogh</name>
    <nationality>Dutch</nationality>
    <birthYear>1853</birthYear>
    <deathYear>1890</deathYear>
    <specialty>Post-Impressionist</specialty>
    <placeOfStudy>Académie Royale des Beaux-Arts</placeOfStudy>
  </artist>
  <!-- Additional artist records follow the same structure -->
</artists>
```

4.2 Events

- Robin

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

<!--This is the DTD and the XML document for the events class and is used by two event
related XQueries-->

<!DOCTYPE events [

<!--The following is a description of the main elements, the cardinality and the general
outline of the DTD seen below:

Event is set to "*: zero-or-more" as no events are technically needed. There will not
always be an event planned to document. All of the event elements are set to one as we
at least need all the information in the document for each single event once. By using
an extra outside bracket for dates, we are not limited in the amount of dates for each
event, hence why it is "+: one-or-more". There is at least one location needed. Some
events might take multiple places/rooms but one is the minimum.

The 6 #PCData elements are all elements in the document helping to describe the events.

eventID ATTLIST is #REQUIRED for every new event for easier allocation and logistic
purposes-->

<!ELEMENT events (event*)>
<!ELEMENT event (eventName, dates, locations, host, manager, eventDescription)>
<!ELEMENT dates (date+)>
<!ELEMENT locations (location+)>

<!ELEMENT eventName (#PCDATA)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT manager (#PCDATA)>
<!ELEMENT eventDescription (#PCDATA)>

<!ATTLIST event eventID CDATA #REQUIRED>

]>

<!--The following XML document is used to give functionality to the events part of our
UML diagram.

The diagrams showed an external partner working with the admins to create and manage
events, so in this document the functionality of a system is created,that allows the
management to oversee certain events with all the necessary details. This could allow
them to properly prepare, to have a general overview, to structure the year, to
calculate funds and to make the necessary changes needed.

We have the events document, with the individual events all sorted via eventID. Within
these events, there are multiple other elements that had mostly been included in the
```

```
class diagram already. Some additional events were created. Every event has its eventName,
its dates, the location it will be occupying/using, the host/external partner, the manager
responsible for the event and a short event description. All of this information allows
everyone that has access, to get a quick overview of the most important facts and to
understand the dimensions of each event.

    There are 8 events with different data to be found below. -->

<events>
    <event eventID="230401001">
        <eventName>Historical Easter Getaway</eventName>
        <dates>
            <date>2023-04-01</date>
        </dates>
        <locations>
            <location>Front Court</location>
        </locations>
        <host>Trinity Food Society</host>
        <manager>Robin Schulz</manager>
        <eventDescription>
The Historical Easter Getaway is an event focused on exploring and celebrating the
cultural and historical traditions of Easter, featuring themed activities and
educational presentations.
        </eventDescription>
    </event>
    <!-- Additional events records follow the same structure -->
</events>
```

4.3 Membership

- Liam

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

<!--
    This XML document represents a list of memberships for an organisation or institution.
    Each membership contains details e.g. purchase price, type, purchase data, expiration,
    date, visitor ID and membership type
-->

<!DOCTYPE Members [

<!ELEMENT Members (Membership+)> <!-- The '+' cardinality is used to indicate that there
must be at least one 'Membership' element in the document-->
<!ELEMENT Membership (price, expirationDate, visitorID, type)> <!-- Each 'Membership'
element contains child elements -->
<!ELEMENT purchasePrice (#PCDATA)>
<!ELEMENT purchaseType (#PCDATA)>
<!ELEMENT purchaseDate (#PCDATA)>
<!ELEMENT expirationDate (#PCDATA)>
<!ELEMENT visitorID (#PCDATA) >
```

```
<!ELEMENT type (#PCDATA)>

<!ATTLIST Membership membershipID CDATA #REQUIRED> <!-- The 'membershipID' attribute is
required for each 'Membership' element to uniquely identify it-->

]>


<Members>
    <!-- Membership record for Visitor ID 1234 with a Patron membership-->
    <Membership membershipID = "0001">
        <purchasePrice>500</purchasePrice>
        <purchaseType>Final</purchaseType>
        <purchaseDate>2023-11-14</purchaseDate>
        <expirationDate>2024-11-14</expirationDate>
        <visitorID>1234</visitorID>
        <type>Patron</type>
    </Membership>
    <!-- Additional membership records follow the same structure -->
</Members>
```

4.4 Artwork

- Rachel

```
<?xml version="1.0" ?>

<!-- This XML document represents a museum's collection of artworks.
        Each artwork entry contains detailed information about the piece,
        including its title, artist, submission date, status, category, and
        description. -->

<!DOCTYPE museum [

    <!-- The root element 'museum' contains multiple 'artwork' elements.
            Cardinality '*' is used to allow zero or more artworks. -->
    <!ELEMENT museum (artwork*)>

    <!-- The 'artwork' element represents an individual art piece.
            Each artwork must have exactly one of each of the following child
            elements: title, artistName, submissionDate, status, category, and
            description. -->
    <!ELEMENT artwork (title, artistName, submissionDate, status, category,
    description)>

    <!-- Simple text content (#PCDATA) is expected for each child element. -->
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT artistName (#PCDATA)>
    <!ELEMENT submissionDate (#PCDATA)>
    <!ELEMENT status (#PCDATA)>
    <!ELEMENT category (#PCDATA)>
```

```
    <!ELEMENT description (#PCDATA)>

    <!-- The attribute 'artworkID' is optional (IMPLIED) and is used as a
    unique identifier for each artwork. -->
    <!ATTLIST artwork artworkID CDATA #IMPLIED>
]>

<museum>
    <artwork artworkID="2001">
    <title>Starry Night</title>
    <artistName>Vincent van Gogh</artistName>
    <submissionDate>12/06/2024</submissionDate>
    <status>Displayed</status>
    <category>Post-Impressionism</category>
    <description>
        A depiction of a swirling night sky filled with stars and expressive
        brushwork that conveys intense emotion.
    </description>
    <!-- Additional artwork entries follow the same structure -->
</museum>
```

4.5 Tickets

- Noah

```
<?xml version="1.0" ?>

<!--
This is the XML and DTD document for the Tickets Use Case. It stores IDs for
both tickets and visitors, the type of tickets along with their price, purchase
date, status and expiration date. -->

<!DOCTYPE Tickets [

    <!-- \Tickets" element has an undefined number of elements, but at least more
    than one, hence the \+". -->
    <!ELEMENT Tickets (Ticket+)>

    <!-- All tickets include information about their type of ticket, along with
    prices and validity dates. -->
    <!ELEMENT Ticket (visitorID, ticketType, price, purchaseDate, ticketStatus,
    expirationDate)>

    <!-- every ticket has a mandatory attribute that must be included, being
    the ID of the ticket -->
    <!ATTLIST Ticket ticketID ID #REQUIRED>

    <!ELEMENT visitorID (#PCDATA)> <!-- ID of the visitor -->
    <!ELEMENT ticketType (#PCDATA)> <!-- type of ticket -->
    <!ELEMENT price (#PCDATA)> <!-- price of ticket -->
    <!ELEMENT purchaseDate (#PCDATA)> <!-- date of purchase -->
    <!ELEMENT ticketStatus (#PCDATA)> <!-- validity of ticket -->
```

```
    <!ELEMENT expirationDate (#PCDATA)> <!-- expiration date of ticket -->
    ]>

<Tickets>
    <!-- one example of a ticket object -->
    <Ticket ticketID="001">
    <visitorID>v150</visitorID>
    <ticketType>Adult</ticketType>
    <price>75.00</price>
    <purchaseDate>2023-11-14</purchaseDate>
    <ticketStatus>Expired</ticketStatus>
    <expirationDate>2023-11-16</expirationDate>
    </Ticket>
    <!-- following ticket objects have the exact same structure -->
</Tickets>
```

4.6 Donations

- Roisin

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

<!--  This XML document stores donation information for various events. Each
donation record includes details about the donor, the amount donated, payment
date, type of donation, and the purpose of the donation.
→

<!DOCTYPE Donations [
    <!-- The 'Donations' element contains one or more 'Donation' elements -->
    <!ELEMENT Donations (Donation+)>

    <!-- The 'Donation' element must contain six child elements: donationID,
    donorID, amount, paymentDate, donationType, donationPurpose -->
    <!ELEMENT Donation (donationID, donorID, amount, paymentDate, donationType,
    donationPurpose)>

    <!-- 'donationID' is a simple text-based element that uniquely identifies the
    donation -->
    <!ELEMENT donationID (#PCDATA)>

    <!-- 'donorID' is a simple text-based element that identifies the donor -->
    <!ELEMENT donorID (#PCDATA)>

    <!-- 'amount' is a text-based element that contains the donation amount, with
    a required attribute 'currency' specifying the currency -->
    <!ELEMENT amount (#PCDATA)>
    <!ATTLIST amount currency CDATA #REQUIRED>

    <!-- 'paymentDate' is a text-based element that specifies the date of the
    donation -->
    <!ELEMENT paymentDate (#PCDATA)>
```

```
    <!-- 'donationType' specifies the type of the donor (e.g., Individual,
    Corporate, Organization) -->
    <!ELEMENT donationType (#PCDATA)>

    <!-- 'donationPurpose' specifies the purpose of the donation (e.g., Exhibition
    Support, General Fund) -->
    <!ELEMENT donationPurpose (#PCDATA)>
]

<Donations>
    <!-- Each Donation represents a single donation made to the gallery -->
    <Donation>
        <donationID>D001</donationID>
        <!-- Unique identifier for the donation -->
        <donorID>USER001</donorID>
        <!-- Unique identifier for the donor -->
        <amount currency="EUR">100.00</amount>
        <!-- The amount donated, with the associated currency (EUR) -->
        <paymentDate>2023-11-14</paymentDate>
        <!-- The date on which the donation was made -->
        <donationType>Individual</donationType>
        <!-- Type of donor: Individual, Corporate, Organization -->
        <donationPurpose>Exhibition Support</donationPurpose>
        <!-- The purpose for which the donation was made -->
    </Donation>
    <!-- Additional Donation records follow the same structure -->
</Donations>
```

4.7 Payment Systems

- Nikolaos

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PaymentSystems [
    <!ELEMENT PaymentSystems (PaymentSystem+)>
    <!ELEMENT PaymentSystem (amount, paymentDate, paymentMethod, userID, status)>
    <!ATTLIST PaymentSystem paymentID ID #REQUIRED>
    <!ELEMENT amount (#PCDATA)>
    <!ELEMENT paymentDate (#PCDATA)>
    <!ELEMENT paymentMethod (#PCDATA)>
    <!ELEMENT userID (#PCDATA)>
    <!ELEMENT status (#PCDATA)>
]>
<PaymentSystems>
    <PaymentSystem paymentID="p002">
        <amount>75.00</amount>
        <paymentDate>2023-11-15</paymentDate>
        <paymentMethod>PayPal</paymentMethod>
        <userID>v124</userID>
        <status>completed</status>
```

```
    </PaymentSystem>
    <PaymentSystem paymentID="p003">
        <amount>30.00</amount>
        <paymentDate>2023-11-16</paymentDate>
        <paymentMethod>Debit Card</paymentMethod>
        <userID>v125</userID>
        <status>completed</status>
    </PaymentSystem>
    <PaymentSystem paymentID="p004">
        <amount>35.00</amount>
        <paymentDate>2022-11-16</paymentDate>
        <paymentMethod>Debit Card</paymentMethod>
        <userID>v126</userID>
        <status>completed</status>
    </PaymentSystem>
</PaymentSystems>
```

4.8 Visitor Users

- Nikolaos

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE VisitorUsers [
    <!ELEMENT VisitorUsers (VisitorUser+)>
    <!ELEMENT VisitorUser (name, email, membershipStatus, purchases, feedbackID)>
    <!ATTLIST VisitorUser visitorID ID #REQUIRED>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
    <!ELEMENT membershipStatus (#PCDATA)>
    <!ELEMENT purchases (ticketID+)>
    <!ELEMENT ticketID (#PCDATA)>
    <!ELEMENT feedbackID (#PCDATA)>
]>
<VisitorUsers>
    <VisitorUser visitorID="v123">
        <name>Alex Turner</name>
        <email>alex505@boardwalk.com</email>
        <membershipStatus>false</membershipStatus>
        <purchases>
            <ticketID></ticketID>
        </purchases>
        <feedbackID>f100</feedbackID>
    </VisitorUser>
    <VisitorUser visitorID="v124">
        <name>Jane Smith</name>
        <email>jane.smith@example.com</email>
        <membershipStatus>false</membershipStatus>
        <purchases>
            <ticketID>t003</ticketID>
        </purchases>
        <feedbackID>f101</feedbackID>
    </VisitorUser>
```

```
    <VisitorUser visitorID="v125">
        <name>Alex Johnson</name>
        <email>alex.johnson@example.com</email>
        <membershipStatus>true</membershipStatus>
        <purchases>
            <ticketID>t004</ticketID>
        </purchases>
        <feedbackID>f102</feedbackID>
    </VisitorUser>
    <VisitorUser visitorID="v126">
        <name>Lorem Ipsum</name>
        <email>lorem.ipsum@example.com</email>
        <membershipStatus>true</membershipStatus>
        <purchases>
            <ticketID>t005</ticketID>
        </purchases>
        <feedbackID>f105</feedbackID>
    </VisitorUser>
</VisitorUsers>
```

4.9 Feedbacks

- Nikolaos

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Feedbacks [
    <!ELEMENT Feedbacks (Feedback+)>
    <!ELEMENT Feedback (visitorID, rating, comments, dateSubmitted, ratingValue)>
    <!ATTLIST Feedback feedbackID ID #REQUIRED>
    <!ELEMENT visitorID (#PCDATA)>
    <!ELEMENT rating (#PCDATA)>
    <!ELEMENT comments (#PCDATA)>
    <!ELEMENT dateSubmitted (#PCDATA)>
    <!ELEMENT ratingValue (#PCDATA)>
]>
<Feedbacks>
    <Feedback feedbackID="f100">
        <visitorID>v123</visitorID>
        <rating>1</rating>
        <comments>The reception staff is rude, paintings boring</comments>
        <dateSubmitted>2023-11-14</dateSubmitted>
        <ratingValue>1</ratingValue>
    </Feedback>
    <Feedback feedbackID="f101">
        <visitorID>v124</visitorID>
        <rating>5</rating>
        <comments>Loved it! Highly recommend.</comments>
        <dateSubmitted>2023-11-15</dateSubmitted>
        <ratingValue>5</ratingValue>
    </Feedback>
    <Feedback feedbackID="f102">
        <visitorID>v125</visitorID>
```

```
        <rating>3</rating>
        <comments>It was good, but could be better.</comments>
        <dateSubmitted>2023-11-16</dateSubmitted>
        <ratingValue>3</ratingValue>
    </Feedback>
    <Feedback feedbackID="f103">
        <visitorID>v126</visitorID>
        <rating>5+</rating>
        <comments>Good ol craic</comments>
        <dateSubmitted>2023-11-16</dateSubmitted>
        <ratingValue>5+</ratingValue>
    </Feedback>
</Feedbacks>
```

5 Credits

5.1 Finn

- Created 1 XML document: Artist
- Added six elements
- Created 1 XQuery:

  artworkPerArtist: Outputs the amount of artwork the gallery has per artist.

- Established a GitHub to enable a collaborative workflow to store XML documents.

5.2 Robin

- Created 1 XML document: Events
- Added six elements
- Created 2 XQueries:

  eventsLengthInDays: Takes in an event title and prints out the number of days and which days of the month the event occurs.

  eventsByManager: Takes in a manager's first name and prints out all the events this manager is responsible for.

- Created a poll for task allocation
- Created the XML report template
- Contributed to:

  UML Design to XML Implementation

  XQueries

  XML Documents

5.3 Liam

- Created 1 XML document: Membership
- Added six elements
- Created 1 XQuery:

  getMemberRevenue: Calculates and displays total revenue generated from types of memberships.

- Compiled and formatted the final report

5.4 Rachel

- Created 1 XML document: Artwork
- Added two elements
- Created 1 XQuery:

  getArtworkForCategory: Implements user-defined function

5.5 Nick

- Created 3 XML documents: Feedback, PaymentSystem and UserVisitor
- Added multiple elements
- Created 2 XQueries:

  getFeedbackFromNonPayingUsers: Retrieves feedback from non-paying visitors.
  getFeedbackFromPayingUsers: Retrieves feedback from paying visitors.

5.6 Roisin

- Created 1 XML document: Donation
- Added six elements
- Created 1 XQuery:

  getSpecificDonor: Retrieves and displays donations by a specific donor.

- Identified strengths and weaknesses of the XML and XQuery design.

5.7 Noah

- Created 1 XML Document: Tickets
- Added six elements
- Created 1 XQuery:

  getRevenueFromDate: Implements user-defined function that retrieves all revenue across Tickets, Membership, and Donations on a certain date.

- Identified strengths and weaknesses of the XML and XQuery design.