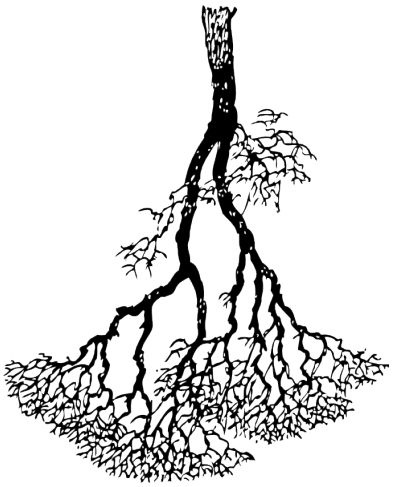
DEPARTEMENT INFORMATIQUE
DE LA FACULTE DES SCIENCES

Quentin Yeche (21520370), Yanis Allouch (21708237)

Rapport du TP Noté : XPath/XQuery



HMIN103 — Données du web

Référent: Federico Ulliana et Pierre Pompidor

2020

Table des matières

Introduction

L'analyse et le traitement des données consistent à les étudier afin d'en extraire des structures capables des les stocker ainsi que de les manipuler de façon la plus optimale.

Ce cinquième TP est une synthèse des précédents, de la conception de structures de données XML via DTD à l'utilisation de *XPATH* et *XQUERY*.

Ce TP est composé de quatre exercices. Le premier concerne la base de données *Tweets* et ses utilisateurs, le second travaille sur la base de données publique des disponibilités des *VeloMagg* de la TAM (Transports de l'Agglomération de Montpellier). Enfin les deux derniers exercices ont pour objectif de développer des concepts intrinsèques aux langages XPath et XQuery.

Le TP se déroule sur papier et machine.
Durant ce TP, nous appliquerons les règles qui seront abordées dans les cours référencés DdW2-XPath et DdW3-XQuery.

L'ensemble des réponses est reporté dans ce compte-rendu.

1 XQuery : Tweets

Pour cette question, vous aurez besoin de reprendre la DTD pour les Tweets réalisée lors du premier TP. La première étape consiste à faire évoluer votre DTD dans le sens suivant.

- La DTD doit permette d'enregistrer une collection de Tweets (et non pas un seul Tweet).
- La DTD doit également permettre l'enregistrement d'un ensemble d'utilisateurs.
- Tweets et utilisateurs seront reliés par des ID/IDREF.

La présente DTD est disponible dans son entièreté en annexe ici. Cela dis en voici quelques lignes pour se représenter en partie la structure du schéma utilisé pour produire les requêtes suivantes.

```
<!DOCTYPE bddTweet[
  <!ELEMENT bddTweet (tweet*, user*)>
    <!ELEMENT tweet (data,metadata?, body, reponse*)>
          <!ATTLIST tweet idT ID #REQUIRED>
          <!ATTLIST tweet idRefUser IDREF #REQUIRED>
        ........
    <!ELEMENT user (nom+, prenom+)>
      <!ATTLIST user idU ID #REQUIRED>
      <!ATTLIST user idRefTweet IDREF #REQUIRED>
      ...
```

Notre base de donnée contient des tweets et des utilisateurs.

- Un tweet est composé d'attributs qui référencent son id propre et son auteur.
- Un tweet est composé des éléments suivant : d'une date, de méta-données optionnelles, d'un corps qui compose le tweet dans son essence et enfin des réponses.
- Ces réponses sont elles-même des tweets. C'est donc la hiérarchie de l'aborescence qui représente la hiéarachi des réponses.
- Les méta-données enregistrent des informations sur la configuration de l'utilisateur, une localisation GPS et des liens externes si le tweet contient un média (partage d'une vidéo, d'une URL, etc.).
- Un body est composé d'informations sur le formatage du texte et la langue du tweet. La balise retweets contient le nombre de fois que le tweet a été retwitté.
- On y retrouve enfin l'auteur et son contenu. Un auteur consiste en un nom et un attribut qui référencent son ID dans la BDD.
- Enfin le contenu possède le texte, les références utilisateurs et les hashtag qui constituent le tweet.

Donner les requêtes XQuery correspondants aux expressions suivantes et évaluer ces expressions dans le document XML pour les Tweets.

1. Indiquer le nombre de tweets et d'utilisateurs dans la base.
   - Proposition 1

     ```
     count(//tweet) + count(//user)
     ```

     Il est supposé qu'il est demandé de compter le total des tweets et des utilisateurs. Sinon il suffit de séparer leur calcul :

— Proposition 2

```
("Nombre de tweets",count(//tweet),
    "Nombre d'utilisateurs", count(//user))
```

2. Donner l'ensemble des hashtags contenus dans la base.
   — Proposition

   ```
   //hastag
   ```

   L'ensemble retourné peux contenir des doublons. Ce problème est aisement corrigé par l'utilisation de la fonction distinct-values() [1].

3. Créer une liste de paires tweet-auteur, avec chaque paire contenue dans un element result.
   — Proposition

   ```
   for $tweet in //tweet
   return
     <result>
     {($tweet,//user[@idU=$tweet/@idRefUser])}
     </result>
   ```

   Requête équivalente mais qu'on préférera lorsque le nombre d'utilisateur ou de tweet est important.

4. Pour chaque utilisateur, lister le nom de l'utilisateur et la date de tous ses tweets, le tout regroupé dans un élément result.
   — Proposition

   ```
   for $user in //user
   return
     <result>
       {
         ($user/nom),
         (for $tweet in //tweet
         return $tweet[@idRefUser = $user/@idU]/date)
       }
     </result>
   ```

   Pour chaqu'un des utilisateurs de la base de donnée, leur nom est d'abord affiché et éventuellement concaténé avec les dates de tous les tweets dans la base de donnée.

5. Lister les utilisateurs qui ont publié un tweet qui a été retwitté.
   — Proposition

   ```
   for $user in //user
   return $user[@idU=//tweet[@idT=//tweet/@idRetweet]/@idRefUser]
   ```

6. Pour chaque tweet, indiquer la date de ses deux premières réponses. Rajouter un element vide <nonRetwitted/> s'il n'a pas été retwitté.
   — Proposition

   ```
   for $tweet in //tweet[.//reponses]
   return
   <tweet>
   ```

1. Nous rappelons que distinct-values renverra alors la valeur de l'arborescence et non l'arborescence

```
  <id>{$tweet/@idT}</id>
  <dates>
      {($tweet//reponses/tweet/date)[position() lt 3]}
  </dates>
  {if (xs:int($tweet//retweets/text()=0)) then
  <nonRetwitted/>
  else ()
}
</tweet>
```

7. Lister les utilisateurs de la plateforme en ordre alphabétique.
   — Proposition

```
for $user in //user
order by $user/nom ascending, $user/prenom
return $user
```

8. Lister les tweets contenants l'hashtag "#I<3XML".
   — Proposition

```
for $tweet in //tweet
where $tweet/body//hashtag/text() = "#I<3XML"
return $tweet
```

9. Trouvez le tweet le plus ancien ainsi que le plus recent.
   — Proposition 1

```
let $order := for $tweet in //tweet
order by $tweet/xs:dateTime(date)
return $tweet

return ($order[1], $order[last()])
```

   — Proposition 2

```
for $tweet in //tweet
where $tweet/xs:dateTime(date) = (max(//tweet/xs:dateTime(date)), min
     (//tweet/xs:dateTime(date)))
return $tweet
```

10. Pour chaque utilisateur, indiquer l'ensemble des hashtags qu'il a utilisés dans ses Tweets.
    — Proposition

```
for  $user in //user
  return
    <result>
    {$user/@idU,
    for $tweet in //tweet
    return $tweet[$tweet/@idRefUser = $user/@idU]/body/content/
     hashtag }
    </result>
```

11. Pour chaque tweet ayant des références utilisateur, retournez le tweet avec la liste des références utilisateur.
    — Proposition

```
for $tweet in //tweet
return
```

```
  <result>
  {
    <listeReferences>
    {
       $tweet/body/content/userref
    }
    </listeReferences>, $tweet
  }
  </result>
```

12. Déclarez la fonction local :aReponduAuTweet, qui, étant donné un tweet, retourne tous les utilisateurs qui ont répondu au Tweet.
    — Proposition

```
declare function local:aReponduAuTweet($tweet) {
    for $refUser in $tweet/descendant::tweet/@idRefUser
      return $refUser
};

(: Cas d'utilisation:)
(: pour le cas vide prendre t7226:)
for $tweet in //tweet
where $tweet/@idT="t6336"
return local:aReponduAuTweet($tweet)
```

    Le seul inconvénient qui vient du faite que cette fonction est local est qu'on ne peut pas lier l'id de l'utilisateur qui est référencé et donc obtenir son nom. Ceci étant seulement de notre point de vue parce qu'on aurait aimé la liste retourné associé au nom de l'utilisateur référencé. Sinon cette requête répond à la question posée.

2 Génération de Pages HTML via XQuery

Écrire un programme XQuery permettant de générer une page HTML contenant trois sections qui présentent la liste des stations Vélomag triées par :

- Ordre alphabétique.
- Capacité (indiquer le nombre total de places).
- Niveau de disponibilité. Pour ce dernier cas prévoir 3 catégories : faible (moins de 30% des vélos disponibles), moyenne (entre 30% et 60% des vélos disponibles), haute (plus de 60% des vélos disponibles).

Le code permettant la génération de cette page HTML étant un peu long, nous présenterons ici seulement des éléments de réponse synthétiques. Le code complet peut être trouvé en annexe

1. La première étape est de récupérer les stations dans les ordres demandés.

```
let $doc := doc("https://data.montpellier3m.fr/sites/default/files/
    ressources/TAM_MMM_VELOMAG.xml")

let $alpha:= for $si in $doc//si
order by $si/substring(@na,5)
return $si
```

   On utilise ici *substring* puisque les noms de station sont préfixés par leur id. Les autres ordre sont gérés ainsi :
   - Capacité totale :

```
  order by $si/xs:int(@to) descending
```

   - Disponibilité. Le plus simple consiste à créer trois séquences différentes pour les différents intervalles demandés. Par exemple pour des disponibilités entre 30 et 60% on utlisera :

```
  where ($si/xs:int(@av) div $si/xs:int(@to) >= 0.3 and
           $si/xs:int(@av) div $si/xs:int(@to) <= 0.6 )
```

2. La deuxième étape consiste à créer les tables. Pour l'ordre alphabétique :

```
let $tableAlpha:=
<div>
 <table>
  <tr>
    <th>Station</th>
    <th>Numero</th>
    <th>Position</th>
  </tr>
  {for $si in $alpha
  return
  <tr>
    <td>{$si/substring(string(@na),5)}</td>
    <td>{$si/xs:int(@id)}</td>
    <td>{$si/string(@la)}<br/>{$si/string(@lg)}</td>
  </tr>
}
</table>
</div>
```

   Les autres tableaux sont très similaires, seuls les en-têtes changent. Pour les classes de disponibilités, on place les trois blocs {for .. } des trois classes à la suite pour ordonner les séquences.

3. On peut enfin procéder à la génération de la page entière en assemblant simplement les tables créés à l'étape précédente :

```
return

<html lang="fr">
<head>
<meta charset="utf-8" />
<!-- <meta charset="iso-latin-1" / -->

<title>Velomagg</title>
<link rel="stylesheet" href="styleVelo.css"/>


</head>
<body>
<header>
<h1>Velomagg</h1>
</header>

<h2> Tries par ordre alphabetique</h2>
{$tableAlpha}
<h2> Tries par capacite </h2>
{$tableTotal}
<h2>Tries par niveau de disponibilite</h2>
{$tableDispo}

<footer>Velomagg : pedalez plus pour moins cher<br/>
</footer>

</body>
</html>
```

3 Propriétés des requêtes XPath

1. Reformuler les requêtes suivantes en utilisant exclusivement les axes child, descendant, descendant-or-self, following et following-sibling
   - Requête 1

```
//d/preceding-sibling::c
```

   - Proposition

```
//c[following-sibling::d]
```

     Ici la transformation est assez simple. Partir d'un noeud d puis chercher un frère précédent c est équivalent à directement chercher les noeuds c qui ont un frère suivant d.

   - Requête 2

```
//c/a/preceding-sibling::a/preceding::e
```

   - Proposition

```
//c/e[following-sibling::a[following-sibling::a]]
    | //e[following::c/a/following-sibling::a]
```

     On a besoin ici de l'union de deux requêtes. La première capture le cas où e et les deux a de la requête sont tous fils d'un c. La deuxième capture le cas où le c qui est parent des deux a est un noeud qui suit e.

   - Requête 3

```
//d[parent::b/c]
```

   - Proposition

```
//b[./c]/d
```

     Question assez similaire à la première requête. La requête renvoie les d qui ont pour père un b qui a pour fils un c. Une reformulation serait, les d qui ont pour père un b et pour frère un c. Pour éviter d'utiliser l'axe *parent* on filtre d'abord les d qui ont pour enfant un c, puis on sélectionne leurs enfants d.

   - Requête 4

```
/r/b/..//*/./../preceding::d
```

   - Proposition

```
/r[./b]/d[following::*[child::*]]
```

     On notera tout d'abord la présence dans la requête de /./ qui n'a ici aucun effet. La partie /r/b/../ est équivalente à /r[./b] : au lieu de chercher /r/b et remonter au parent r on sélectionne directement les r qui ont pour enfant un b. La partie //*/./../ sélectionne tous les noeuds puis remonte à leurs parents. Cela revient donc à sélectionner tous les noeuds qui ont un enfant. Le cas du parent de r que r//*/./../ sélectionnerait n'est pas gênant puisque r est ici la racine de l'arborescence.

— Requête 5

```
//a/ancestor::c/child::d/parent::e
```

— Proposition

```
()
```

Cette requête est équivalente à la requête vide puisque toutes deux ne renvoient jamais aucun résultat. En effet d doit être un enfant d'un c et avoir pour parent un e, ce qui n'est pas possible. Par définition, dans un arbre le parent est unique.

— Requête 6

```
//c[preceding::d]
```

— Proposition

```
//d/following::c
```

Cette requête et sa solution sont le cas inverse de la requête 1. On transforme un filtrage en une arborescence et inversement.

2. Reformuler les requêtes en utilisant les axes descendant-or-self, ancestor, following-sibling et preceding-sibling.

— Requête 1

```
//a/following::b
```

— Proposition

```
//a/following-sibling::b |
    //a/ancestor::*/following-sibling::*/descendant-or-self::b
```

La première partie de la requête sélectionne tout simplement les b frères suivants d'un a. La seconde partie sélectionne le reste des b suivants de a. On tire parti du fait que les suivants d'un noeud a (qui ne sont pas ses frères) sont l'union des descendant-or-self des following-sibling de tous les ancêtres de a.

— Requête 2

```
//a/preceding::b
```

— Proposition

```
//a/preceding-sibling::b |
    //a//ancestor::*/preceding-sibling::*/descendant-or-self::b
```

Cette requête est identique à la première, il s'agit juste de remplacer following par preceding pour passer de l'une à l'autre. C'est également le cas pour la solution.

3. Pour chaque requête définie aux points 1 et 2, proposer un document XML pour lequel la réponse à la requête n'est pas vide, sinon expliquer pourquoi un tel document n'existe pas.

Les documents XML suivants ne sont pas aussi courts qu'ils pourraient l'être. Ce sont les arbres sur lesquels nous avons testé nos requêtes. Nous avons donc essayé d'être aussi exhaustifs que possible, en incluant des balises qui ne devraient pas être sélectionnées par la requête.

Question 1

1)

```xml
<a>
  <c>1</c>      <!--   matching   -->
  <d>2</d>
  <c>4</c>      <!-- non-matching -->
  <c>3          <!-- non-matching -->
    <d></d>
  </c>
</a>
```

2)

```xml
<ro>
   <c>
    <a>1</a>
    <a>2</a>
  </c>
  <e>1</e>      <!--   matching   -->
  <c>
    <e>2</e>    <!--   matching   -->
    <a>1</a>
    <a>2</a>
  </c>
  <e>3</e>      <!-- non-matching -->
  <a>3</a>
  <c>
      <e>4</e>  <!-- non-matching -->
      <a>4</a>
  </c>
</ro>
```

3)

```xml
<ro>
  <b>
    <d>1</d>          <!--   matching   -->
    <c></c>
  </b>
   <b>
    <c></c>
    <d>2</d>          <!--   matching   -->
  </b>
   <b>
    <c> <d>3</d></c>  <!-- non-matching -->
  </b>
</ro>
```

4)

```xml
<r>
  <b>1</b>
  <c>1</c>
  <d>1            <!--   matching   -->
    <b>3</b>
  </d>
  <e>
    <f></f>
```

```
  </e>
  <g></g>
  <d>2</d>this <d/> should not be selected because the <e>2</e> has
     no child
  <e>2</e>
</r>
```

5) Voir notre réponse à la question 1 concernant cette requête. Il n'y a pas d'arbre qui peut satisfaire cette contrainte.

6)

```
<ro>
  <c>4</c>            <!-- non-matching -->
  <b>
    <d>
      <e>
        <c>6</c>      <!-- non-matching -->
      </e>
    </d>
  </b>
  <d>matcher</d>
  <c>7</c>            <!--   matching   -->
  <b>
    <c>8</c>          <!--   matching   -->
  </b>
</ro>
```

Question 2

1)

```
<ro>
  <b>1</b>          <!-- non-matching -->
  <a>4
    <b>3</b>        <!-- non-matching -->
  </a>
  <b>6</b>          <!--   matching   -->
  <b>2
    <a>
      <b>3</b>      <!--   matching   -->
    </a>
  </b>
</ro>
```

2)

```
<ro>
  <b>2</b>        <!--   matching   -->
  <d>
    <b>3</b>      <!--   matching   -->
  </d>
  <d></d>
  <b>1            <!-- non-matching -->
    <a></a>
  </b>
</ro>
```

4 L'égalité dans XQuery

1. Soient X,Y, Z des séquence d'éléments XML. Est il vrai que, dans le cadre du langage XPath, si X = Y et Y = Z alors X = Z ? Est-ce le cas pour XQuery ?
   — Au regard de la citation [1] nous pouvons affirmer que, dans XPath 1.0 et XQuery basé sur XPath 1.0, l'égalité n'est pas la transitive. En effet on a :

```
let $a := 1
let $b := (1,2)
let $c := 2
a = b   (: renvoie true() :)
b = c   (: renvoie true()  :)
a = c   (: renvoie false() :)
```

2. Donner une fonction XQuery qui renvoie vrai si et seulement si deux séquences sont identiques. Pour simplifier, nous ne considérons pas les attributs (mais nous considérerons bien l'ordre des éléments).

```
1  declare function local:eq($n1,$n2) {
2      if (count($n1)!= count($n2))
3        then false()
4      else
5      if (count($n1)=0) then true()
6      else
7      if (count($n1) > 1)
8          then (local:eq($n1[1],$n2[1]) and
9                local:eq($n1[position() >1], $n2[position() >1])
10              )
11     else
12     if ($n1 instance of element())
13         then if (not($n2 instance of element())) then false()
14              else ( local:eq(name($n1), name($n2))
15                         and local:eq($n1/text(), $n2/text())
16                         and local:eq($n1/child::* , $n2/child::*)
17                   )
18     else
19         ($n1 = $n2)
20 };
```

   Cette fonction, bien que longue, n'est qu'une simple succession de tests sur la nature possible des arguments $\$n_1$ et $\$n_2$:
   — ligne 2 : Si $count(\$n_1) \neq count(\$n_2)$) alors l'égalité est impossible ;
   — ligne 5 : Si $count(\$n_1) = 0$ alors $\$n_1 = \$n_2 = ()$;
   — ligne 7 : Si $count(\$n_1) > 1$ alors $\$n_1$ et $\$n_2$ sont des séquences. On teste donc l'égalité de leurs têtes et l'égalité de leurs queues grâce à un appel récursif ;
   — ligne 11 : A ce niveau $\$n_1$ et $\$n_2$ sont tous les deux de taille 1 ;
   — ligne 12-13 : Si $\$n_1$ et $\$n_2$ sont des arborescences on vérifie l'égalité de leurs noms de balise $name()$, de leur contenus $text()$, puis récursivement l'égalité de leurs séquences d'enfants respectives ;
   — ligne 18 : A ce niveau $\$n_1$ et $\$n_2$ sont de taille 1 et ne sont pas des arborescences. Ce sont donc des types primitifs (eg. xs :int, xs :string, xs :time ...) , et le comportement par défaut de l'opérateur d'égalité = est alors approprié à notre cahier des charges.

5 Annexe

5.1 XQuery : Tweets

La Document Type Definition de l'exercice 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bddTweet [
    <!ELEMENT bddTweet (tweet*, user*)>
        <!ELEMENT tweet (date,metadata?,body, reponses*)>
            <!ATTLIST tweet idT ID #REQUIRED>
            <!ATTLIST tweet idRefUser IDREF #REQUIRED>
            <!ATTLIST tweet idRetweet IDREF #IMPLIED>
            <!ELEMENT date (#PCDATA)>
            <!ELEMENT metadata (media?,location?,userConfig?)>
                <!ELEMENT media (url+)>
                <!ELEMENT url (#PCDATA)>

            <!ELEMENT location (gps?,city?,country?)>
                <!ELEMENT gps (lat,long)>
                <!ELEMENT lat (#PCDATA)>
                <!ELEMENT long (#PCDATA)>
                <!ELEMENT city (#PCDATA)>
                <!ELEMENT country (#PCDATA)>

            <!ELEMENT userConfig (version)>
                <!ATTLIST userConfig osType (Android|iOS|Windows|Linux|unknown
    |other) "unknown">
                <!ELEMENT version (#PCDATA)>

            <!ELEMENT body (formatting,language,retweets,author,content)>
                <!ATTLIST body idOriginalTweet IDREF #IMPLIED>

                <!ELEMENT formatting (fontsize,fontcolor,font)>
                    <!ELEMENT fontsize (#PCDATA)>
                    <!ELEMENT fontcolor (#PCDATA)>
                    <!ELEMENT font (#PCDATA)>
                <!ELEMENT language (#PCDATA)>
                <!ELEMENT retweets (#PCDATA)>
                <!ELEMENT author (name, userref)>
                <!ATTLIST author idAuthor IDREF #REQUIRED>
                    <!ELEMENT name (#PCDATA)>

                <!ELEMENT content (#PCDATA|url|hashtag|userref)*>

                    <!ELEMENT hashtag (#PCDATA)>
                    <!ELEMENT userref (#PCDATA)>

            <!ELEMENT reponses (tweet)*>
    <!ELEMENT user (nom+, prenom+)>
        <!ATTLIST user idU ID #REQUIRED>
        <!ATTLIST user idRefTweet IDREF #IMPLIED>
        <!ELEMENT nom (#PCDATA)>
        <!ELEMENT prenom (#PCDATA)>
]>
```

XML servant au test des query de l'exo 1.

```
<bddTweet>
```

```
<tweet idT="t6666" idRefUser="u1">
  <date>2020-10-15T18:53:25</date>
  <body>
    <formatting>
      <fontsize></fontsize>
      <fontcolor></fontcolor>
      <font></font>
    </formatting>
    <language></language>
    <retweets>1</retweets>
    <author idAuthor="u1">
      <name></name>
      <userref></userref>
    </author>
    <content>
      .
      <userref>@exemple</userref>
      absolutely smashed it at
      <hashtag>#mtvlivelockdown</hashtag>
      ! Catch hhim at the official
      <userref>@clubmtvuk</userref>
      after party tonight @ 10pm
    </content>
  </body>
    <reponses>
      <tweet idT="t5665" idRefUser="u2">
      <date>2020-10-16T13:48:32</date>
      <body>
        <formatting>
          <fontsize></fontsize>
          <fontcolor></fontcolor>
          <font></font>
        </formatting>
        <language></language>
        <retweets>0</retweets>
        <author idAuthor="u2">
          <name></name>
          <userref></userref>
        </author>
        <content>
          Nooope!
        </content>
      </body>
      </tweet>
</reponses>
</tweet>
  <tweet idT="t6226" idRefUser="u1">
  <date>2020-10-10T07:06:17</date>
  <body>
    <formatting>
      <fontsize></fontsize>
      <fontcolor></fontcolor>
      <font></font>
    </formatting>
    <language></language>
    <retweets>0</retweets>
    <author idAuthor="u1">
      <name></name>
      <userref></userref>
    </author>
    <content>
      I can assure
```

```
      <userref>@Alxxwi</userref>
      Looking around me,
      <hashtag>#I&lt;3XML</hashtag>
      We use a collection of XML
      <userref>@Cristophe</userref>
      . Its mission is to provide superior technology and expertise
    </content>
  </body>
</tweet>
  <tweet idT="t6336" idRefUser="u3">
  <date>2020-10-01T12:14:53</date>
  <body>
    <formatting>
      <fontsize></fontsize>
      <fontcolor></fontcolor>
      <font></font>
    </formatting>
    <language></language>
    <retweets>0</retweets>
    <author idAuthor="u3">
      <name></name>
      <userref></userref>
    </author>
    <content>
      .
      <userref>@Cristophe</userref>
      Then I summarize the reasons for which it is an absolutely abominable
   film?
      <hashtag>#mtvlivelockdown</hashtag>
      Everything has to be absolutely above-board
      <userref>@Cristophe</userref>
      I needed to talk with someone who was very smart after party tonight @
    10pm
    </content>
    </body>

    <reponses>
      <tweet idT="t7226" idRefUser="u2">
        <date>2020-10-10T08:06:17</date>
        <body>
          <formatting>
            <fontsize></fontsize>
            <fontcolor></fontcolor>
            <font></font>
          </formatting>
          <language></language>
          <retweets>0</retweets>
          <author idAuthor="u1">
            <name></name>
            <userref></userref>
          </author>
          <content>
            This is just to say
          </content>
        </body>
      </tweet>
      <tweet idT="t7227" idRefUser="u3">
        <date>2020-10-10T09:06:17</date>
        <body>
          <formatting>
            <fontsize></fontsize>
            <fontcolor></fontcolor>
```

```
            <font></font>
          </formatting>
          <language></language>
          <retweets>0</retweets>
          <author idAuthor="u1">
            <name></name>
            <userref></userref>
          </author>
          <content>
            I have eaten
            the <hashtag>#plums</hashtag>
            that were in
            the icebox
          </content>
        </body>
      </tweet>
      <tweet idT="t7228" idRefUser="u1">
        <date>2020-10-10T10:06:17</date>
        <body>
          <formatting>
            <fontsize></fontsize>
            <fontcolor></fontcolor>
            <font></font>
          </formatting>
          <language></language>
          <retweets>0</retweets>
          <author idAuthor="u1">
            <name></name>
            <userref></userref>
          </author>
          <content>
             and which
             you were probably
             saving
             for breakfast
          </content>
        </body>
      </tweet>
    </reponses>
</tweet>
<tweet idT="t6446" idRefUser="u4" idRetweet="t6666">
  <date>2020-10-17T00:42:35</date>
  <body>
    <formatting>
      <fontsize></fontsize>
      <fontcolor></fontcolor>
      <font></font>
    </formatting>
    <language></language>
    <retweets>0</retweets>
    <author idAuthor="u4">
      <name></name>
      <userref></userref>
    </author>
    <content>
      She was quite aware of her own limitations
      <userref>@Jean</userref>
      Scotland coach Matt Williams is absolutely right
      <hashtag>#howTOuseAbsolutely</hashtag>
      All I know is what I read in the paper,
      <userref>@Alxxwi</userref>
      because of the authority he brings to it
```

```
      </content>
    </body>
  </tweet>
  <tweet idT="t6556" idRefUser="u4">
    <date>2020-10-13T11:23:46</date>
    <body>
      <formatting>
        <fontsize></fontsize>
        <fontcolor></fontcolor>
        <font></font>
      </formatting>
      <language></language>
      <retweets>0</retweets>
      <author idAuthor="u4">
        <name></name>
        <userref></userref>
      </author>
      <content>
        .
        <userref>@Jean</userref>
        I know where to go when
        <hashtag>#COVID19</hashtag>
        need new news. What about all those words and expressions
        <userref>@Cristophe</userref>
        ?
      </content>
    </body>
  </tweet>
  <user idU="u1">
    <nom>Dupont</nom>
    <prenom>Jean</prenom>
  </user>
  <user idU="u2">
    <nom>Dupont</nom>
    <prenom>Christophe</prenom>
  </user>
  <user idU="u3">
    <nom>Mazrie</nom>
    <prenom>Emilie</prenom>
  </user>
  <user idU="u4">
    <nom>Alxxwi</nom>
    <prenom>Dupont</prenom>
  </user>
</bddTweet>
```

5.2 Génération HTML des stations Vélomagg : code entier

```
let $doc := doc("https://data.montpellier3m.fr/sites/default/files/ressources/
    TAM_MMM_VELOMAG.xml")

let $alpha:= for $si in $doc//si
order by $si/substring(@na,5)
return $si

let $total :=  for $si in $doc//si
order by $si/xs:int(@to) descending
return $si

let $faible :=  for $si in $doc//si
```

```
where ($si/xs:int(@av) div $si/xs:int(@to) < 0.3)
return $si

let $moyen :=  for $si in $doc//si
where ($si/xs:int(@av) div $si/xs:int(@to) >= 0.3 and $si/xs:int(@av) div $si/
     xs:int(@to) <= 0.6 )
return $si

let $haut :=  for $si in $doc//si
where ($si/xs:int(@av) div $si/xs:int(@to) >0.6 )
return $si

let $tableAlpha:=
<div>
 <table>
  <tr>
    <th>Station</th>
    <th>Numero</th>
    <th>Position</th>
  </tr>
  {for $si in $alpha
  return
  <tr>
    <td>{$si/substring(string(@na),5)}</td>
    <td>{$si/xs:int(@id)}</td>
    <td>{$si/string(@la)}<br/>{$si/string(@lg)}</td>
  </tr>
}
</table>
</div>

let $tableTotal :=
<div>
 <table>
  <tr>
    <th>Station</th>
    <th>Nombre total de places</th>
    <th>Position</th>
  </tr>
  {for $si in $total
  return
  <tr>
    <td>{$si/substring(string(@na),5)}</td>
    <td>{$si/xs:int(@to)}</td>
    <td>{$si/string(@la)}<br/>{$si/string(@lg)}</td>
  </tr>
}
</table>
</div>

let $tableDispo :=
<div>
 <table>
  <tr>
    <th>Station</th>
    <th>Disponibilite</th>
    <th>Position</th>
  </tr>
  {for $si in $haut
  return
  <tr style="background-color:#2dc937;">
    <td>{$si/substring(string(@na),5)}</td>
```

```
    <td>Haute ({format-number($si/xs:int(@av) div $si/xs:int(@to), '0%')})</td>
    <td>{$si/string(@la)}<br/>{$si/string(@lg)}</td>
  </tr>
}
  {for $si in $moyen
  return
  <tr style="background-color:#e7b416;">
    <td>{$si/substring(string(@na),5)}</td>
    <td>Moyenne ({format-number($si/xs:int(@av) div $si/xs:int(@to), '0%')})</td>
    <td>{$si/string(@la)}<br/>{$si/string(@lg)}</td>
  </tr>
}
  {for $si in $faible
  return
  <tr style="background-color:#db7b2b;">
    <td>{$si/substring(string(@na),5)}</td>
    <td>Faible ({format-number($si/xs:int(@av) div $si/xs:int(@to), '0%')})</td>
    <td>{$si/string(@la)}<br/>{$si/string(@lg)}</td>
  </tr>
}

</table>
</div>

return

<html lang="fr">
<head>
<meta charset="utf-8" />
<!-- <meta charset="iso-latin-1" / -->

<title>Velomagg</title>
<link rel="stylesheet" href="styleVelo.css"/>


</head>
<body>
<header>
<h1>Velomagg</h1>
</header>

<h2> Tries par ordre alphabetique</h2>
{$tableAlpha}
<h2> Tries par capacite </h2>
{$tableTotal}
<h2>Tries par niveau de disponibilite</h2>
{$tableDispo}

<footer>Velomagg : pedalez plus pour moins cher<br/>
</footer>

</body>
</html>
```

Références

[1] Don CHAMBERLIN et al. *Influences on the Design of XQuery [On XQuery from the Experts : A Guide to the W3C XML Query Language]*. URL : https://www.informit.com/articles/article.aspx?p=100667&seqNum=5. (accessed : 15.10.2020).