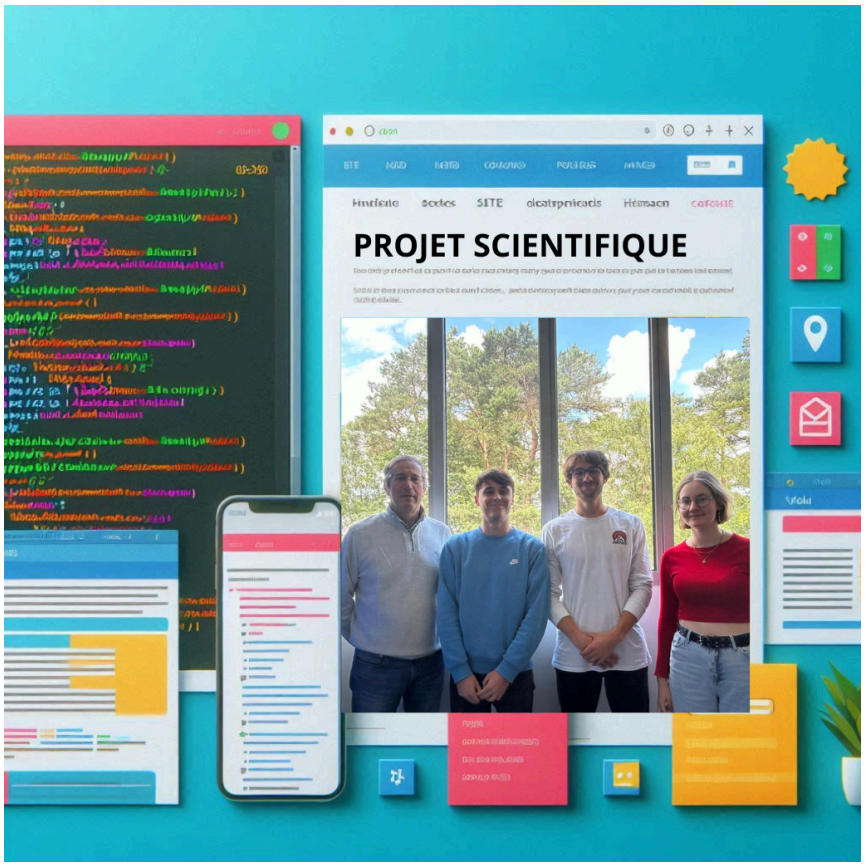
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet de Physique P6

STPI/P6/2024 – 32

GESTIONS D'INFORMATIONS MULTI-MÉDIA DANS UN DOCUMENT



Etudiants :

Iris DUSSUYER **Mathieu HARI**

Ugo LARTIGAU

Enseignant-responsable du projet :

Michel MAINGUENAUD

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Date de remise du rapport : ***15/06/2024***

Référence du projet : ***STPI/P6/2024 – 32***

Intitulé du projet : ***Gestions d'informations multi-média dans un document***

Type de projet : ***Bibliographie, prototype***

Objectifs du projet :

L'objectif premier de notre projet est de comprendre comment fonctionne l'algorithme d'un moteur de recherche, c'est-à-dire quels sont ses critères de pertinence et comment sont-ils évalués pour renvoyer une liste de pages internet à l'utilisateur. Le second objectif est ensuite de coder un moteur de recherche simple basé sur le XQuery. Enfin, le but de cette matière est d'apprendre à travailler en groupe de manière autonome et d'élargir notre culture scientifique.

Mots-clefs du projet: XML, XQuery, données, multi-média

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

TABLE DES MATIÈRES:

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

NOTATIONS, ACRONYMES

SEO : Search Engine Optimization

HTML : Hypertext Markup Language

SGML : Standard Generalized Markup Language

WWW : World Wide Web

HTTP : Hypertext Transfer Protocol

URL : Uniform Resource Locator

XHTML : Extensible Hypertext Markup Language

XML : Extensible Markup Language

WHATWG : Web Hypertext Application Technology Working Group

DTD : Document Type Documentation

CSS : Cascading Style Sheets

DOM : Document Object Model

XSD : XML Schema Definition

W3C : World Wide Web Consortium

SQL : Structured Query Language

DML : Data Manipulation Language

DDL : Data Definition Language

DCL : Data Control Language

ISO : International Organization for Standardization

IEC : International Electrotechnical Commission

FLOWR : For, Let, Where, Order by et Return

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

1. Introduction

Durant notre deuxième année de STPI, nous avons été amenés à mener un projet scientifique s'intitulant « Gestion d'informations multimédias dans un document ». Il s'agit d'un sujet de bibliographie et prototype. En effet, dans un contexte où l'information est présente et variée, il est crucial de gérer de manière efficace les données multimédias. L'objectif de notre projet est de faire face à ce défi en créant une solution informatique qui peut traiter et intégrer différents types de médias dans un seul document.

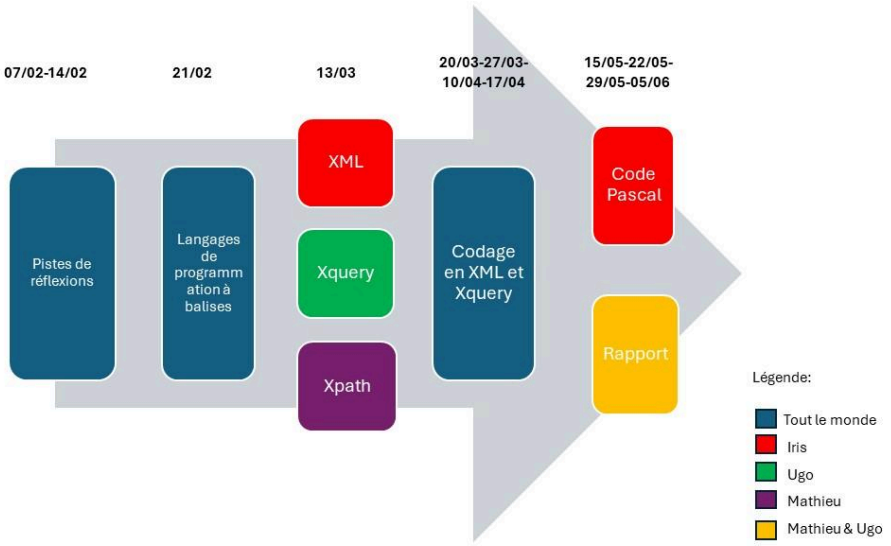
L'un des principaux buts de ce projet est de favoriser le développement de compétences en travail en équipe. Travailler en équipe nous aidera à développer notre autonomie et à acquérir des compétences pour répartir les tâches de manière efficace afin d'améliorer notre productivité. Chaque membre de l'équipe a une responsabilité particulière dans une partie du projet, ce qui nous permet une spécialisation des compétences et une gestion plus efficace du temps. L'objectif de cette approche collaborative est de reproduire un contexte professionnel où la coordination et la communication jouent un rôle crucial.

Deuxièmement, l'objectif final est de coder un programme en Pascal capable de gérer et questionner des informations multimédias dans un document.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

2. Méthodologie / Organisation du travail

Durant, les deux premières séances, Monsieur Mainguenaud nous a expliqué plus en détails le déroulement et les attentes qu'il avait concernant le projet. Nous avons ensuite exploré différentes pistes de réflexion liées au fonctionnement des moteurs de recherches et des données multimédia. Lors de la séance suivante, nous nous sommes intéressés davantage aux langages de programmation à balises HTML, XML et SQL. Pendant les vacances, nous avons séparé nos recherches, Iris devait se renseigner sur le XML, Mathieu sur le XPath et Ugo sur le XQuery. Durant les quatre séances ultérieures, nous avons retravaillé en groupe sur le XML et le XQuery, appris à coder dans ces deux langages et trouvé leurs limites. Enfin, les quatre dernières semaines nous ont permis de finaliser notre projet, Iris s'est occupée du code en Pascal permettant d'interroger un document XML à partir d'un code Xquery généré par Pascal et Mathieu et Ugo ont rédigé le rapport du projet.



D'un point de vue organisation, nous avons dès la première séance créé un document partagé pour synthétiser et partager nos recherches bibliographiques, ainsi qu'un Drive regroupant ce document et nos ébauches de codes en XML et Xquery.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

3. Travail réalisé et résultats

3.1. Information multimédia

Aujourd'hui, Internet a pris une place très importante dans notre quotidien et il est impensable de devoir s'en passer. Ses usages sont multiples, internet permet un accès total à l'information, simplifie la communication et est un outil formidable pour l'éducation et le commerce. Pour cela, l'utilisation des moteurs de recherches est devenue essentielle. En effet, 68% des expériences en ligne commencent par un moteur de recherche. Le fonctionnement des moteurs de recherches est un point stratégique aussi bien pour les créateurs de site internet que pour les utilisateurs.

3.1.1. SEO

La visibilité d'une page internet passe par son référencement qui est le moyen de classer les pages internet après une recherche. Le référencement se fait selon plusieurs critères qui peuvent être améliorés d'après le SEO: l'optimisation On-page, l'optimisation Off-page, et l'expérience utilisateur.

L'optimisation On-page consiste à améliorer le contenu textuel grâce au balisage. En effet, chaque page et site internet sont codés à partir d'un langage de balisage qui donne une structure à la page dont se sert le moteur de recherche. Généralement le langage utilisé est le HTML. Le moteur de recherche repère d'abord la balise <title> de la page dont le contenu s'affiche en grand puis, lors d'une recherche d'un site particulier le contenu de la balise <description> s'affiche ensuite. Lors d'une recherche de mots clés, la balise <description> peut être remplacée par du texte de la page plus pertinent au regard de la recherche de l'utilisateur.

L'optimisation Off-page est la promotion de son site internet sur des sites ou pages externes car les moteurs de recherches classent les pages également par rapport aux liens renvoyant sur la page (backlinks). Ainsi, deux facteurs sont pris en compte, la quantité des liens, et plus important, la qualité de la provenance de ces backlinks.

L'expérience utilisateur est l'élément le plus récent à avoir été pris en compte par les moteurs de recherches mais a pris une grande place dans le référencement. Il regroupe plusieurs critères importants que sont l'ergonomie du site, le taux de rebond, le taux de conversion et le parcours utilisateur. Pour diminuer le taux de rebond ou améliorer le taux de conversion, nous pouvons notamment réduire le poids et la vitesse de chargement de la page, travailler l'apparence du site ou encore structurer l'architecture et l'arborescence du site.

Dans ce projet, nous allons nous concentrer sur la partie On-page et sur la façon dont les moteurs de recherches lisent les langages à balises pour ensuite classer les pages. Les langages de balisage spécialisé dans l'enrichissement d'info textuelle permettent de structurer un document grâce aux balises qui ne sont pas visibles par le lecteur et encadrent un texte de la manière suivante: une balise d'ouverture de cette forme, <balise> et une balise de fermeture de cette forme, </balise>.

3.1.2. HTML

Actuellement, 95% de tous les sites Web utilisent HTML. Nous nous sommes donc dans un premier temps intéressés à ce langage de programmation. HTML est conçu en 1989 par Tim Berners-Lee et est alors une application du SGML. Ce langage s'inscrit dans la création du WWW, du HTTP et des URL par monsieur Berners-Lee et permettait de nommer et de structurer le document, d'y inclure des liens hypertextes et de pouvoir y effectuer une recherche par index. Progressivement de nouvelles fonctionnalités vont s'ajouter comme les images, la saisie de données par l'utilisateur en 1993 ; les tables, les figures et les expressions mathématiques en 1996 ; les styles et les scripts, les cadres et les objets en 1997 avec la version 4.0 de HTML qui introduit également trois nouvelles variantes : strict, transitional et frameset qui rendent le langage plus accessible et favorise l'interopérabilité. A partir de 2000, le développement du HTML en tant qu'application du SGML est abandonné et laisse place au XHTML, application de XML. Parallèlement, le WHATWG développé en 2007 HTML5 compatible avec le XHTML et le XML, puis le HTML Living Standard en 2011 qui est une version du HTML5 en constante évolution qui permet l'ajout de nouvelles balises. Cette dernière version est celle actuellement utilisée par les développeurs.

Jusqu'à la version 4.0, HTML était formellement décrit comme une application du SGML. Cependant, les navigateurs Web n'ont jamais été capables de déchiffrer toutes les variations de syntaxe permises par SGML mais peuvent rattraper automatiquement beaucoup d'erreurs de syntaxe. De plus, la DTD de HTML permet de vérifier la validité d'un document HTML à l'aide d'un parseur SGML . Ainsi, les développeurs de pages Web ont pris des libertés avec les règles syntaxiques de SGML. En résumé, un document HTML valide respecte la syntaxe SGML, utilise des éléments et attributs standardisés, et suit l'imbrication des éléments décrite par le standard. Cependant, la conformité à la spécification HTML va au-delà de la validité et inclut d'autres contraintes spécifiées dans la documentation.

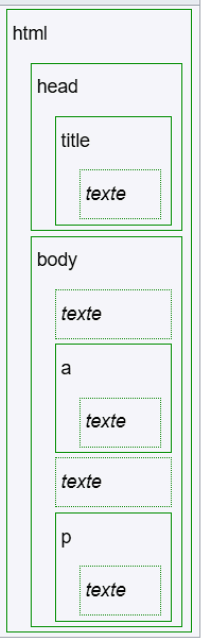
La structure des documents HTML était d'abord considérée comme plates et les balises étaient des éléments stylistiques comme la balise <p> qui servait au saut de ligne. La structure en arbre est apparue avec le CSS et le DOM. Un élément racine bien souvent <html> contient tous les autres éléments qui ont chacun un seul parent direct. La balise <html> a généralement deux enfants, <head> qui contient obligatoirement l'élément <title> affiché en titre d'onglet de navigateur,et en titre de résultat de moteur de recherche, et <body>.

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Structure d'un document HTML

Source HTML	Modèle du document

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
 <head>
  <title>
   Exemple de HTML
  </title>
 </head>
 <body>
  Ceci est une phrase avec un <a href="cible.html">hyperlien</a>.
  <p>
   Ceci est un paragraphe où il n'y a pas d'hyperlien.
  </p>
 </body>
</html>
```



3.1.3. Différence entre HTML et XML

HTML et XML sont deux langages de balisage utilisés dans le monde du développement Web et des technologies de l'information. Bien qu'ils partagent certaines similitudes, ils sont conçus pour des usages différents et peuvent répondre à des besoins différents.

Ces deux langages de balisage ont des objectifs et des utilisations différents. En effet, le HTML est principalement utilisé pour structurer et afficher le contenu sur le Web. Vous pouvez créer des pages Web en définissant des éléments tels que des titres, des paragraphes, des liens, des images, des formulaires, etc. Les navigateurs Web interprètent le HTML et affichent le contenu visuellement. Il est destiné à décrire l'apparence et la structure d'un site Web. Le XML, quant à lui, est utilisé pour transférer et stocker des données. Il se concentre sur la signification et la structure des données plutôt que sur la représentation visuelle des données. XML est utilisé pour échanger des informations entre les systèmes et pour stocker des données structurées. Cela vous permet de définir des balises personnalisées pour les besoins spécifiques de votre application.

HTML dispose également d'un ensemble prédéfini de balises, où les balises et leurs significations sont fixes. Les balises HTML doivent être utilisées correctement pour que le contenu s'affiche correctement dans un navigateur Web. Or, XML est plus flexible que HTML puisqu'il permet aux utilisateurs de créer leurs propres balises adaptées à leurs besoins. Les balises XML ne possèdent pas de fonctionnalités prédéfinies, et leur signification est déterminée par l'utilisateur.

Concernant la syntaxe, HTML est assez tolérant ; les navigateurs sont capables de gérer du code HTML mal formé dans une certaine mesure. Par exemple, les balises de fermeture et les éléments imbriqués incorrectement peuvent souvent être interprétés de manière adéquate par les navigateurs. Contrairement au HTML, le XML exige une syntaxe stricte et bien formée. Chaque balise ouvrante doit avoir une balise fermante correspondante, et les balises doivent être correctement imbriquées. Toute erreur de syntaxe rend le document XML invalide.

Concernant leur structure, celle de HTML est prédéfinie et sa validité peut être vérifiée contre les spécifications HTML. Les documents HTML suivent des standards qui évoluent avec les versions HTML (comme HTML5). XML permet l'utilisation de schémas pour définir la structure et les contraintes des documents XML. Des schémas comme DTD et XSD peuvent être utilisés pour garantir que le document suit une structure spécifique.

HTML et XML servent des objectifs fondamentalement différents. HTML est utilisé pour la présentation de contenu web, tandis que XML est utilisé pour le transport et le stockage structuré de données. Bien que ces deux langages partagent certaines similitudes syntaxiques, leur utilisation et leurs contraintes diffèrent significativement. Adapter le bon langage au bon contexte est essentiel pour résoudre les problèmes spécifiques de structure, de présentation et d'échange de données.

3.1.4. Pourquoi avoir choisi le XML

Le choix du XML pour notre projet scientifique repose sur plusieurs raisons clés qui font de ce format une solution privilégiée pour la gestion et l'interrogation de données structurées.

Le XML est un standard largement reconnu et adopté par le World Wide Web Consortium (W3C). Grâce à sa structure textuelle, les fichiers XML peuvent être facilement lus et interprétés par divers logiciels, sans dépendance à un environnement spécifique.

Le XML offre la possibilité de classer les données de façon hiérarchique. L'organisation des informations est essentielle pour des applications qui requièrent une organisation claire et logique, ce qui facilite l'extraction et la manipulation des données. Les différents éléments et leurs relations sont clairement définis par des balises, ce qui en fait le format XML parfait pour des requêtes complexes et précises via XQuery.

À la différence d'autres formats de données, le XML offre une grande flexibilité. Il permet de créer des balises sur mesure qui répondent aux exigences particulières de chaque projet. Il est crucial d'avoir cette souplesse pour des applications scientifiques où les types de données peuvent être très divers et demander des descriptions précises et sur mesure.

Le XML supporte la validation des données via des DTD ou des schémas XML. Cette fonctionnalité assure que les données adhèrent à une structure définie, garantissant ainsi leur intégrité et leur cohérence.

La communauté scientifique utilise fréquemment le format XML pour partager et publier des données. Le XML est à l'origine de nombreux standards de données scientifiques tels que les formats utilisés dans les domaines de la bio-informatique, de l'astronomie ou encore de la gestion des publications académiques.

Le choix du XML pour notre projet repose sur ses compétences en matière de structure, de validation et d'interrogation des données de manière souple et standardisée. Grâce à sa popularité au sein de la communauté scientifique et à sa compatibilité avec XQuery, cet outil est puissant et adapté aux besoins particuliers de notre application. Grâce à l'utilisation de XML, nous garantissons la solidité, la résistance et la compatibilité de notre système, assurant ainsi la réussite de notre projet scientifique.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

3.2. Langages d'interrogations

Après avoir choisi le XML, il nous fallait trouver un langage de programmation capable d'interroger un document XML. Nos recherches se sont d'abord portées sur le SQL, puis sur le Xpath et le Xquery.

3.2.1. SQL

Le SQL est un langage standard utilisé pour interagir avec des bases de données relationnelles. Il permet de créer, modifier, gérer et interroger des bases de données en utilisant des instructions spécifiques. Les bases de données relationnelles organisent les données en tables, chaque table étant composée de lignes (enregistrements) et de colonnes (attributs). Cela facilite la gestion structurée des données et assure l'intégrité des informations stockées.

SQL se divise en plusieurs sous-langages, chacun ayant un rôle spécifique. Le DML inclut des instructions pour manipuler les données. Par exemple, la commande SELECT est utilisée pour récupérer des données, tandis que INSERT ajoute de nouvelles lignes, UPDATE modifie des enregistrements existants, et DELETE supprime des lignes. Ces commandes permettent une gestion dynamique et précise des données au sein des tables.

Le DDL est utilisé pour définir et modifier la structure des bases de données. Par exemple, la commande CREATE permet de créer de nouvelles tables ou bases de données, tandis que ALTER modifie la structure des tables existantes, et DROP supprime des tables ou des bases de données. Ces instructions sont essentielles pour la conception et l'évolution de la structure des bases de données.

Le DCL gère les permissions et la sécurité des bases de données. Les commandes GRANT et REVOKE sont utilisées pour accorder et révoquer des droits d'accès aux utilisateurs respectivement. Cela permet de contrôler qui peut accéder et manipuler les données, assurant ainsi la sécurité et la confidentialité des informations.

SQL offre de nombreux avantages. Il est largement standardisé (ISO/IEC), ce qui facilite l'interopérabilité entre différents systèmes de gestion de bases de données comme MySQL, PostgreSQL, et Oracle. Il permet des requêtes complexes et des manipulations robustes des données grâce à sa syntaxe expressive et puissante. De plus, SQL supporte les transactions, assurant la cohérence et la fiabilité des opérations sur les données, ce qui est crucial pour les applications critiques.

Cependant, SQL présente aussi certaines limites. Les bases de données relationnelles peuvent rencontrer des difficultés de scalabilité horizontale pour des volumes de données extrêmement grands, en comparaison avec les bases de données NoSQL. De plus, les schémas des bases de données relationnelles sont rigides, nécessitant des modifications structurelles en cas de changements dans les besoins de données. Cela peut rendre les bases de données relationnelles moins flexibles face à des modifications fréquentes de la structure des données.

SQL se distingue de XQuery, qui est utilisé pour interroger des documents XML. Alors que SQL manipule des données tabulaires dans des bases de données relationnelles, XQuery est conçu pour naviguer et manipuler des données hiérarchiques dans des documents XML. Ces deux langages répondent à des besoins différents et sont optimisés pour leurs structures de données respectives.

En conclusion, SQL reste un outil puissant et largement utilisé pour la gestion des bases de données relationnelles. Il offre une grande flexibilité et une compatibilité avec divers systèmes de gestion de bases de données, permettant une gestion efficace et sécurisée des données structurées. Cependant, pour des projets nécessitant la manipulation de données XML, des langages comme XQuery peuvent être plus appropriés. SQL et XQuery, bien que différents, sont complémentaires dans leurs domaines d'application respectifs, permettant ainsi une gestion et une interrogation efficaces des données.

3.2.2. XPath

XPath, ou XML Path Language, est un langage utilisé pour naviguer et sélectionner des parties spécifiques d'un document XML. Ce langage est essentiel pour interroger et manipuler les données contenues dans des fichiers XML, en offrant une manière efficace et flexible d'extraire les informations requises. XPath permet de localiser des éléments, des attributs, des textes, et d'autres nœuds dans un document XML grâce à des expressions de chemin.

Les nœuds XML constituent les éléments de base dans la structure d'un document XML. Un nœud peut être un élément, comme <nom>Jean</nom>, un attribut, comme id dans <utilisateur id="1">, ou le texte contenu dans un élément, tel que "Jean" dans <nom>Jean</nom>. Il existe également des nœuds de commentaires, des nœuds de traitement d'instructions et des nœuds racines, qui sont les nœuds de plus haut niveau dans un document XML.

Les expressions XPath utilisent une syntaxe spécifique pour naviguer dans la structure hiérarchique des documents XML. Par exemple, le symbole / sélectionne le nœud racine, tandis que // sélectionne des nœuds à n'importe quel niveau du document. Par conséquent, //nom sélectionne tous les éléments <nom> présents dans le document. D'autres symboles comme . et .. permettent de sélectionner respectivement le nœud courant et le nœud parent. L'arobase @ est utilisée pour sélectionner des attributs, par exemple //@id sélectionne tous les attributs id.

XPath offre également la possibilité d'appliquer des filtres et des conditions pour affiner les sélections. Par exemple, l'expression //utilisateur[@id='1'] sélectionne l'élément <utilisateur> dont l'attribut id est égal à 1. XPath inclut de nombreuses fonctions intégrées, telles que text(), position(), et last(), qui permettent de manipuler et d'interroger les données de manière plus sophistiquée.

XPath présente plusieurs avantages significatifs. Il permet des sélections très précises et détaillées de nœuds dans les documents XML, offrant ainsi une grande flexibilité pour manipuler des données complexes. Sa compatibilité avec d'autres technologies XML, telles que XQuery et XSLT, en fait un outil polyvalent et puissant pour diverses applications XML.

Cependant, XPath a également des limites. Les expressions XPath peuvent devenir complexes et difficiles à maîtriser, en particulier pour les documents XML très imbriqués. De plus, l'exécution de requêtes XPath sur des documents XML de grande taille peut être coûteuse en termes de performance, nécessitant des optimisations spécifiques pour maintenir une efficacité acceptable.

XPath est souvent utilisé en combinaison avec XQuery, un langage plus complet pour interroger et transformer des données XML. Alors que XPath se concentre principalement sur la navigation et la sélection de nœuds dans un document XML, XQuery étend ces

capacités avec des fonctionnalités supplémentaires pour manipuler et transformer les données.

XQuery utilise fréquemment des expressions XPath pour localiser les données avant de les traiter de manière plus élaborée.

En conclusion, XPath est un outil indispensable pour interroger et manipuler des documents XML. Il permet une navigation précise et une sélection détaillée des nœuds, ce qui est essentiel pour toute application nécessitant une interaction avec des données XML. Maîtriser XPath est crucial pour tirer pleinement parti des capacités des technologies XML et effectuer des manipulations avancées des données XML.

3.2.3. XQuery

XQuery est un langage de requête XML qui date de janvier 2007. XQuery joue un rôle similaire à celui du langage SQL vis-à-vis des données relationnelles, mais spécifiquement pour les données au format XML. On peut trouver des analogies entre ces deux langages, bien que leurs domaines d'application diffèrent.

XQuery peut suivre deux syntaxes distinctes: le FLOWR qui est la syntaxe naturelle non adaptée au XML et la syntaxe XQueryX dont la requête est un document XML et qui renvoie un fragment de XML.

La syntaxe FLOWR en XQuery est utilisée pour effectuer des requêtes qui impliquent l'itération, le filtrage, le tri et la construction de valeurs. Voici comment chaque partie fonctionne :

- For : Sélectionne une séquence de nœuds ou de valeurs atomiques. C'est la base de l'itération où vous spécifiez une variable qui parcourt les éléments d'une séquence.
- Let : Attribue une valeur à une variable pour une itération spécifique. Cela peut être utilisé pour stocker des résultats intermédiaires.
- Where : Applique un prédicat de filtre sur les itérations. Cela permet de filtrer les résultats basés sur une condition spécifique.
- Order by : Trie les résultats basés sur une expression donnée. Cela peut être utilisé pour organiser les résultats dans un ordre spécifique.
- Return : Construit le résultat de l'instruction FLOWR. C'est ce que la requête retournera après avoir exécuté les clauses précédentes.

Pour ce projet nous allons utiliser la syntaxe XQueryX qui permet d'interroger un document XML ce qui est l'objectif final. Il y a deux cas d'utilisation du XQueryX, le cas simple que nous allons privilégier qui consiste à reprendre, dans la requête, le nom des balises déjà existantes dans le document XML interrogé et les réutiliser dans le résultat. Le deuxième cas plus complexe construit des nœuds XML en reprenant plusieurs noms de balises et de contenus associés pour créer une nouvelle balise avec son contenu unique dans le résultat.

XQueryX permet de nombreuses fonctionnalités très utiles dans l'interrogation de documents. Pour les expliquer nous allons utiliser le document XML suivant en exemple:

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

```
<bib>
  <book title="Comprendre XSLT">
    <author><la>Amann</la><fi>B.</fi></author>
    <author><la>Rigaux</la><fi>P.</fi></author>
    <publisher>O'Reilly</publisher>
    <price>28.95</price>
  </book>
  <book year="2001" title="Spatial Databases">
    <author><la>Rigaux</la><fi>P.</fi></author>
    <author><la>Scholl</la><fi>M.</fi></author>
    <author><la>Voisard</la><fi>A.</fi></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>35.00</price>
  </book>
  <book year="2000" title="Data on the Web">
    <author><la>Abiteboul</la><fi>S.</fi></author>
    <author><la>Buneman</la><fi>P.</fi></author>
    <author><la>Suciu</la><fi>D.</fi></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
</bib>
```

Except permet de renvoyer tous les sous éléments sauf ceux suivant la fonction except:

- Requête:

```
<livre>
  Tous les sous-elements sauf les auteurs:
  { doc("bib.xml")//book[1]/(* except author) }
</livre>
```

- Résultat:

```
<livre>
  Tous les sous-elements sauf les auteurs:
  <publisher>O'Reilly</publisher>
  <price>28.95</price>
</livre>
```

Pour pouvoir mettre en page le résultat et transformer le noeud en valeur, nous utilisons la fonction string:

- Requête:

```
"Les auteurs du premier livre sont",
doc("bib.xml")//book[1]/author/string(la)
```

- Résultat: la séquence

```
Les auteurs du premier livre sont, Amann, Rigaux
```

Nous pouvons également comparer des valeurs avec la fonction eq et des séquences avec des opérateurs tels que =,<, <=, !=, > et >= :

- Requête:

```
doc("bib.xml")//book/author[la eq "Scholl"]
```

- Résultat:

```
<author><la>Scholl</la><fi>M.</fi></author>
```

- Requête:

```
count(doc("bib.xml")//book[author/la =
                    ("Scholl", "Rigaux", "Abiteboul")])
```

- Résultat: 3

3.2.4. Pourquoi avoir choisi le XQuery

Le SQL n'étant pas adapté, nous devions choisir entre le XPath et le XQuery. Les deux langages sont utilisés pour travailler avec des documents XML, mais ils ont des applications et des capacités différentes. Nous avons finalement choisi le XQuery qui présente de nombreux avantages.

En effet, XQuery est conçu pour effectuer des requêtes et des opérations plus complexes. Il permet non seulement de naviguer et de sélectionner des éléments, mais aussi de construire des éléments, de trier les données, et d'effectuer des jointures entre différents documents XML.

De plus, il peut être utilisé pour transformer des données XML en un autre format, comme HTML ou texte, ce qui est particulièrement utile pour la publication de données sur le web.

XQuery est également un langage de requête complet qui inclut XPath comme sous-ensemble. Cela signifie que tout ce que nous pouvons faire avec XPath, nous pouvons également le faire avec XQuery, plus d'autres fonctionnalités telles que la création de nouvelles structures XML.

La syntaxe FLOWR est aussi un avantage. En effet, elle est similaire à SQL et permet une plus grande flexibilité dans la formulation des requêtes et la manipulation des résultats.

Enfin, XQuery peut être intégré avec d'autres standards XML tels que XSLT et XPointer, ce qui permet une plus grande interopérabilité entre les différentes technologies XML.

Ainsi, XQuery offre une plus grande puissance et flexibilité pour interroger et manipuler des données XML par rapport à XPath, qui est plus limité aux sélections de nœuds simples dans un document XML.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

3.3. Le code

Connaissant le but de notre projet, nous avons d'abord identifié 3 parties de code à gérer. La première partie serait une interface utilisateur permettant de comprendre ce qu'il voulait rechercher, ainsi que de lire et de comprendre le fichier xml. La deuxième partie serait de créer un fichier xquery à partir de ces informations, et de l'exécuter avec Saxon afin d'obtenir un fichier réponse. Enfin, le dernière serait d'interpréter ce fichier et d'afficher les résultats de la recherche de façon visuelle à l'utilisateur.

Nous avons ensuite fait une analyse descendante (voir Annexe n°1). Puis nous avons commencé à coder.

Pour cela, le programme principal serait en Pascal, utilisant la bibliothèque Crt pour un affichage plus visuel. Il aurait également une partie interprétant un fichier xml (comme un fichier texte), et une partie écrivant un fichier texte en Xquery. Enfin, comme le fichier réponse créé par Saxon est un html, il aurait une partie interprétant le html. Nous avons donc manipulé plusieurs langages de programmation.

3.3.1. Le fichier XML

La première étape a été de créer un fichier xml pour effectuer les tests. Nous avons d'abord créé un fichier très simple composé d'une liste de trois personnes avec leurs noms, prénoms, et autres informations. Pour ensuite coller au sujet, nous avons créé un fichier xml avec des recettes et donc du texte et des images. Vous pourrez en voir le début dans le 3.3.

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

<!DOCTYPE site [
        <!ELEMENT site (recette)*>
        <!ELEMENT recette (nom, type, ingredients, image, date_de_l_image?, presentation?)>
        <!ELEMENT nom (#PCDATA)>
        <!ELEMENT type (#PCDATA | entree | plat | dessert)*>
        <!ELEMENT ingredients (#PCDATA)>
        <!ELEMENT image (#PCDATA)>
        <!ELEMENT date_de_l_image (#PCDATA)>
        <!ELEMENT presentation (#PCDATA)>
]>

<site>
        <recette>
                <nom>Mousse au chocolat</nom>
                <type>dessert</type>
                <ingredients>chocolat, oeufs</ingredients>
                <image>moussechocjpg</image>
                <date_de_l_image>2022-11-27</date_de_l_image>
                <presentation>dans un bol</presentation>
        </recette>

        <recette>
                <nom>Salade Cesar</nom>
                <type>entree</type>
                <ingredients>salade, poulet, parmesan</ingredients>
                <image>salade_cesar.jpg</image>
```

Ln 8, Col 34 | 3 046 caractères | 100% | Windows (CRLF) | UTF-8

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

3.3.2. Code en Xquery

Nous avons d'abord effectué des tests simples sur le premier fichier xml, en codant directement en XQuery (sans Pascal), pour comprendre le fonctionnement de XQuery. Nous avons découvert que les navigateurs à notre disposition ne compilaient pas le XQuery.

3.3.3. Utilisation de Saxon et du terminal

Nous nous sommes donc renseignés sur comment utiliser le XQuery, et nous avons découvert l'outil Saxon. Saxon est une technologie éditée par la société Saxonica, qui permet, grâce à Java, de compiler des fichiers XQuery. Nous avons également dû apprendre à l'utiliser. Saxon n'est pas utilisable directement dans un programme Pascal, donc il faut ouvrir un Terminal et y taper une commande. Cette commande est donnée pendant l'exécution du programme, mais c'est dommage que l'on n'ait pas pu l'intégrer directement.

Ces deux étapes ont été assez difficiles car il existe très peu de documentation et d'informations sur le XQuery et Saxon, ou alors des informations très anciennes.

3.3.4. Code en Pascal

La partie la plus concrète du projet est donc le programme Pascal.

Le code est composé de 2 types, 6 fonctions et 11 procédures, ainsi que du programme principal. Vous trouverez le code complet en Annexe n°2 .

Les types utilisés sont les suivants : d'abord le type Categories qui est un tableau de Chaînes de caractères, puis le type TabRep qui est un tableau de Categories, et le type arbo qui est une structure utilisée pour stocker les niveaux d'arborescence.

Nous avons également développé des fonctions pour faciliter certaines tâches. La première est sliceStr, qui, à partir d'une chaîne de caractères et d'indices de début et de fin, renvoie la chaîne comprise entre ces deux indices. On a également AvancerChar qui est utilisé dans une boucle pour avancer d'un caractère dans une chaîne et d'augmenter un indice. Ensuite, il y a la fonction longueur qui donne la longueur non vide d'un tableau de type Categories. La fonction inArray renvoie True si une chaine de caractères en paramètre est dans une case d'un tableau de type Categories, False sinon. La fonction ConcatArrays prend en entrée deux tableaux de type Categories et renvoie un seul tableau qui est la concaténation des deux précédents. La fonction suivante, RecupEntreUL, est utilisée pour comprendre le fichier html produit par Saxon. Elle prend en entrée toute la partie d'une ligne qui est entre deux balises <ul> et , et renvoie un tableau avec chaque donnée séparée, dans sa catégorie. Enfin, la fonction centrer sert uniquement à l'affichage final des résultats : elle prend en entrée une chaîne de caractères et renvoie le même texte centré avec des espaces ajoutés pour que la chaîne fasse 20 caractères.

Ensuite, les procédures sont plutôt utilisées pour toutes les interactions avec l'utilisateur ou avec des fichiers extérieurs. La première procédure appelée est inputUser. Elle demande à l'utilisateur le nom du fichier xml qu'il souhaite interroger, ainsi que si le fichier continent une partie DTD. Cette partie DTD contient des informations sur les types de données fournies dans le xml. Au début, l'idée était donc de coder une procédure qui récupérait ces informations dans le DTD. Mais deux problèmes nous ont fait changer d'avis.

Déjà, la partie DTD n'est pas obligatoire. Même si elle est fortement conseillée, on n'en trouve pas dans tous les fichiers xml. Comme on voulait que notre code soit adapté au plus de situations possibles, ce n'était pas pratique. De plus, il était plus difficile de récupérer les niveaux d'arborescence avec le DTD. Donc la procédure ReadXML récupère les informations sur l'arborescence du xml en repérant les < (balises ouvrantes) et les > (balises fermantes).

ReadXML lit un fichier, passe la partie DTD si il y en a une et récupère toutes les catégories présentes dans le xml (entre les balises < et >).

Ensuite, on appelle les procédures QuelleRequete et QuelleCategorie, qui utilisent toutes les deux la procédure AfficherESelectionner, qui utilise la bibliothèque Crt pour permettre à l'utilisateur de sélectionner parmi une liste de manière visuelle.

Après l'obtention de l'entrée utilisateur, on a les deux procédures EcrireQueryRecherche et EcrireQueryAffciher. Ces deux procédures ont été codées sur la base des tests que nous avons réalisés pour savoir comment coder en XQuery : nous avons créé un code qui rédige un fichier texte avec les commandes en XQuery nécessaire pour les deux actions possibles. La première permet d'afficher toutes les données dont la donnée dans une certaine catégorie (choisie par l'utilisateur) correspond à une chaîne de caractères (entrée par l'utilisateur également). La deuxième permet d'afficher toutes les données d'une catégorie choisie par l'utilisateur.

Enfin, la dernière procédure appelée par le programme principal est OutputUser. Cette procédure va d'abord écrire dans le terminal les instructions pour utiliser Saxon, et attendre que l'utilisateur l'ait fait. Puis elle va appeler une autre procédure, comprendreHTML. Cette dernière procédure lit le fichier html dans le but d'en récupérer un Tableau de Categories. Ici, nous nous sommes heurtés à un problème. En effet, le fichier html créé par Saxon est en une seule ligne. Or cette ligne fait souvent plus de 255 caractères, c'est-à-dire plus que le nombre de caractères maximal du type String. Après recherches, nous avons trouvé le type AnsiString, qui est comme le type String mais sans limite de caractères.

Ensuite, OutputUser utilise la fonction centrer mentionnée précédemment pour afficher les résultats, soit sous forme de tableau si on a fait une recherche, soit sous forme de liste si on a fait un affichage.

Et le programme principal est dans une boucle pour pouvoir effectuer plusieurs recherches jusqu'à ce que l'action choisie soit 'fin'.

3.4. Exemple de fonctionnement - Résultats

Vous trouverez dans l'annexe n°3 des extraits de chaque type de fichier utilisé, ainsi qu'un exemple de l'exécution du programme.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

4. Conclusions et perspectives

Ce projet de gestion d'informations multimédias dans un document nous a permis d'explorer les mécanismes de fonctionnement des moteurs de recherche, notamment à travers l'étude du SEO, du HTML et du XML. Nous avons appris à utiliser les langages XML et XQuery pour structurer et interroger les données de manière efficace. En codant un moteur de recherche simple et en développant des compétences en travail d'équipe, nous avons acquis une expérience précieuse en gestion de projet et en programmation. Ce travail a montré l'importance de la collaboration et de l'autonomie dans un contexte scientifique et technique. Les résultats obtenus démontrent la pertinence du choix du XML pour la gestion de données structurées et soulignent les bénéfices d'une approche méthodologique rigoureuse. Les perspectives futures incluent l'optimisation du code et l'exploration de nouvelles applications des techniques apprises.

5. Bibliographie

Les balises pour référencer son site sur Google (e-monsite.com)(valide à la date du 12/06/2024)

SEO : Qu'est ce que le SEO (Search Engine Optimization) ?(valide à la date du 12/06/2024)

Langage de balisage — Wikipédia (wikipedia.org)(valide à la date du 12/06/2024)

Qu'est-ce que le XML ? – Le langage de balisage extensible (XML) expliqué – AWS (amazon.com)(valide à la date du 12/06/2024)

Balises HTML pour le référencement Google : les bonnes pratiques (efficaceweb.fr)(valide à la date du 12/06/2024)

valeurc-xquery-4x.pdf (cnam.fr)(valide à la date du 12/06/2024)

DOM (Document Object Model) XML - .NET | Microsoft Learn(valide à la date du 12/06/2024)

Bases de données XML : bases de XQuery (univ-nantes.fr)(valide à la date du 12/06/2024)

xquery.pdf (u-cergy.fr)(valide à la date du 12/06/2024)

Documentation d'eXist-db(valide à la date du 12/06/2024)

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

6. Annexes Documentation technique

6.1. Analyse descendante du programme

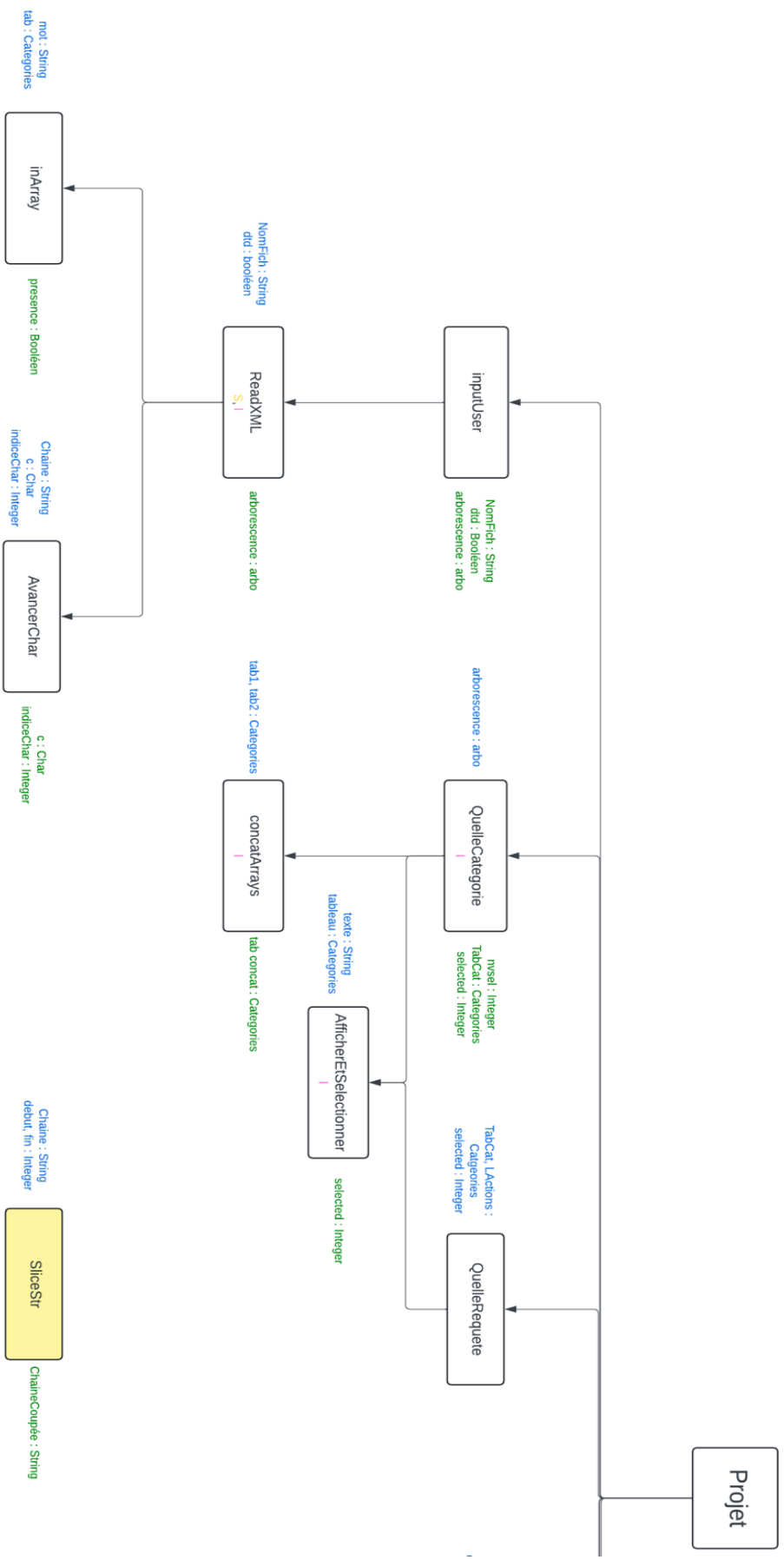
6.2. Types de fichiers utilisés et exemple de l'exécution du programme

6.3. Code du programme Pascal

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Annexe n°1 :

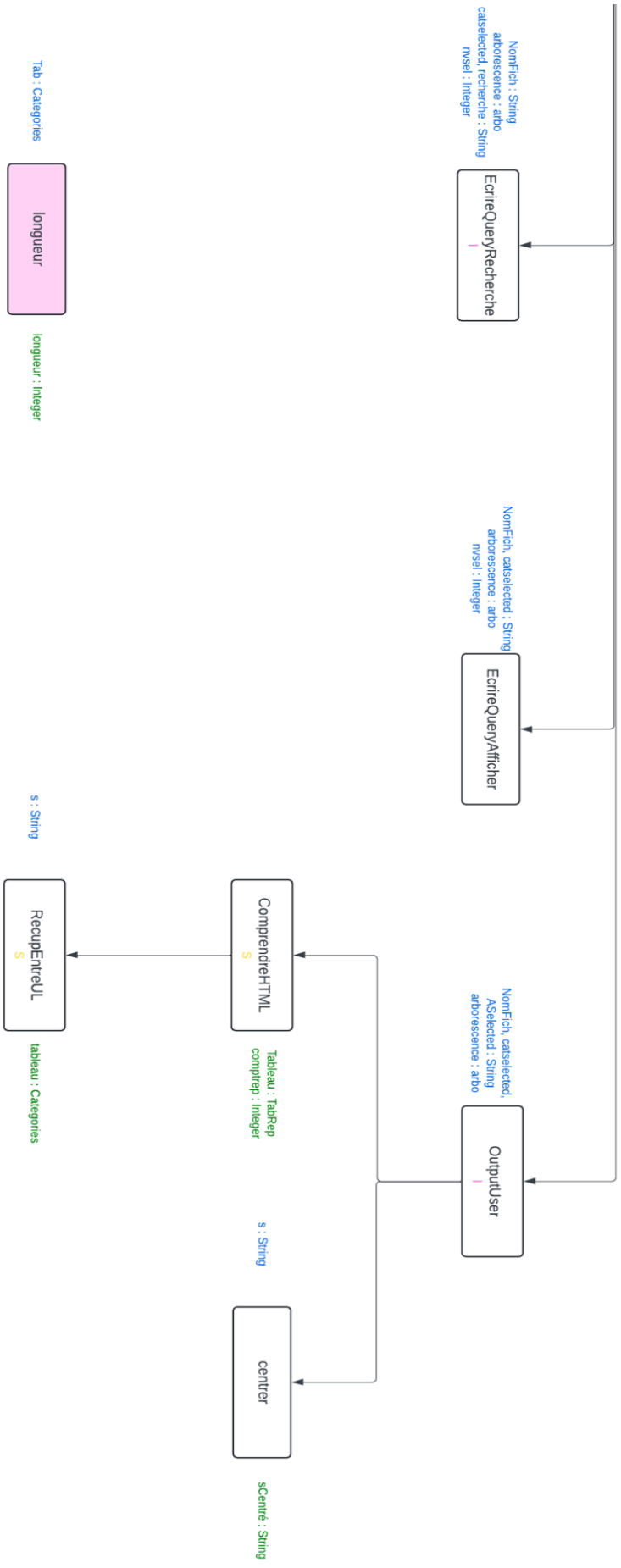
Analyse descendante (1/2)



INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Annexe n°1

Analyse descendante (2/2)



INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

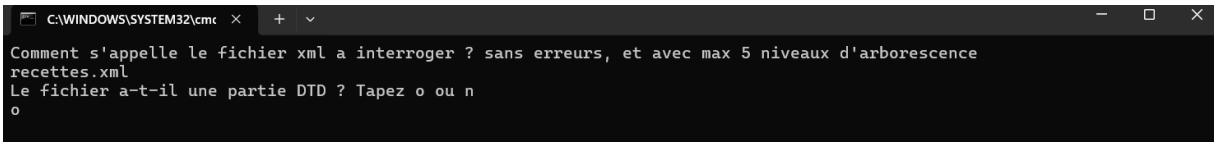
Annexe n°2 : types de fichiers utilisés, et exemple de l'exécution du programme

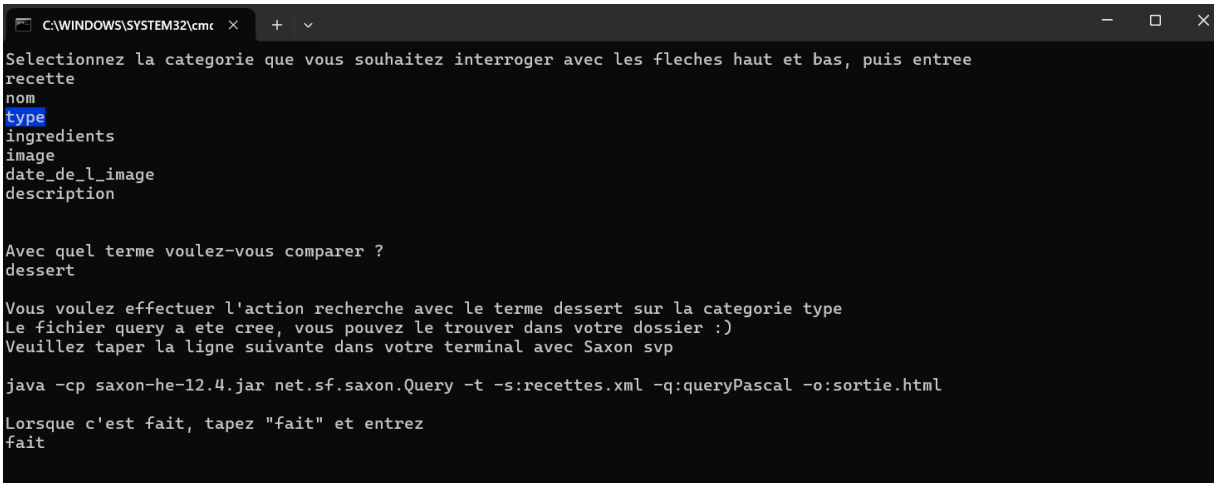
Voici le début du fichier xml utilisé pour l'exemple :

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

<!DOCTYPE site [
        <!ELEMENT site (recette)*>
        <!ELEMENT recette (nom, type, ingredients, image, date_de_l_image?, presentation?)>
        <!ELEMENT nom (#PCDATA)>
        <!ELEMENT type (#PCDATA | entree | plat | dessert)*>
        <!ELEMENT ingredients (#PCDATA)>
        <!ELEMENT image (#PCDATA)>
        <!ELEMENT date_de_l_image (#PCDATA)>
        <!ELEMENT presentation (#PCDATA)>
]>

<site>
        <recette>
                <nom>Mousse au chocolat</nom>
                <type>dessert</type>
                <ingredients>chocolat, oeufs</ingredients>
                <image>moussechocjpg</image>
                <date_de_l_image>2022-11-27</date_de_l_image>
                <presentation>dans un bol</presentation>
        </recette>

        <recette>
                <nom>Salade Cesar</nom>
                <type>entree</type>
                <ingredients>salade, poulet, parmesan</ingredients>
                <image>salade_cesar.jpg</image>
```

Ln 8, Col 34 | 3046 caractères | 100% | Windows (CRLF) | UTF-8

Voici les captures d'écran d'un exemple d'exécution du programme.





INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Voici la ligne copiée dans un terminal avec Saxon

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindow
s

PS C:\Users\CE PC\Documents\Iris\INSA\P6\SaxonHE12-4J> java -cp saxon-he-12.4.jar net.sf.saxon.Query -t -s:recettes.xml
-q:queryPascal -o:sortie.html
```

```
les recettes correspondant(e)s sont :
|        nom         |        type        |    ingredients     |       image        | date_de_l_image  |    presentation    |
-------------------------------------------------------------------------------------------------------------------------------
| Mousse au chocolat |      dessert       |  chocolat, oeufs   |   moussechocjpg    |     2022-11-27   |    dans un bol     |
|      Tiramisu      |      dessert       |  mascarpone, cafe  |    tiramisu.jpg    |     2023-05-18   |    en verrines     |
|    Panna Cotta     |      dessert       |       creme        |  panna_cotta.jpg   |     2023-07-22   |    en verrines     |
|    Creme brulee    |      dessert       |    creme, sucre    |  creme_brulee.jpg  |     2023-10-30   |    en ramequin     |
Lorsque vous avez fini, tapez fini pour effectuer une autre action
```

```
Vous voulez effectuer l'action afficher tout avec le terme  sur la categorie nom
Le fichier query a ete cree, vous pouvez le trouver dans votre dossier :)
Veuillez taper la ligne suivante dans votre terminal avec Saxon svp

java -cp saxon-he-12.4.jar net.sf.saxon.Query -t -s:recettes.xml -q:queryPascal -o:sortie.html

Lorsque c'est fait, tapez "fait" et entrez
fait
Voici la liste de tous les noms :
Mousse au chocolat
Salade Cesar
Quiche Lorraine
Tiramisu
Soupe a l'oignon
Poulet roti
Panna Cotta
Ratatouille
Bruschetta
Creme brulee
Riz Pilaf

Lorsque vous avez fini, tapez fini pour effectuer une autre action
```

Voici le fichier query créé par le programme Pascal pour la requête ‘afficher tout’ :

```
<html>
{for $a in doc ("recettes.xml")//site/recette
where $a/type = "dessert"
return
  <ul>{
    $a/nom,
    $a/type,
    $a/ingredients,
    $a/image,
    $a/date_de_l_image,
    $a/presentation
  }</ul>
}
</html>
```

Et voici le fichier html créé par Saxon :

```
<?xml version="1.0" encoding="UTF-8"?><html><ul><nom>Mousse au chocolat</nom></ul><ul><nom>Salade Cesar</nom></ul><ul>
<nom>Quiche Lorraine</nom></ul><ul><nom>Tiramisu</nom></ul><ul><nom>Soupe a l'oignon</nom></ul><ul><nom>Poulet
roti</nom></ul><ul><nom>Panna Cotta</nom></ul><ul><nom>Ratatouille</nom></ul><ul><nom>Bruschetta</nom></ul><ul><nom>
Creme brulee</nom></ul><ul><nom>Riz Pilaf</nom></ul></html>
```

```
 1  program P6 ;
 2  uses Crt;
 3
 4  const MAXLignes=40; //doit être supérieur au nombre de lignes du xml
 5
 6  Type Categories = array[1..MAXLignes] of Ansistring;
 7
 8  Type arbo = record   //structure qui récupère l'arborescence
 9          nom : String;
10          nv2 : Categories;
11          nv3 : Categories;
12          nv4 : Categories;
13          nv5 : Categories;
14      end;
15
16  Type TabRep = array[1..MAXLignes] of Categories;
17
18
19  //fonction "slice", qui récupère une chaine de caractères entre les indices debut
    et fin de la chaine de caractères d'entrée
20  function sliceStr (chaine : Ansistring; debut, fin : Integer): String;
21      var i : Integer;
22      chainecoupee :String;
23      begin
24          chainecoupee := '';
25          for i:=debut to fin do
26              chainecoupee := chainecoupee + chaine[i];
27
28          sliceStr := chainecoupee;
29      end;
30
31  procedure AvancerChar (lect : String ; var c : Char ; var ic : Integer);
32      begin
33          c:= lect[ic];
34          ic:=ic+1;
35      end;
36
37  //donne la longueur non vide d'un tableau
38  function longueur(Tab : Categories): Integer;
39  begin
40      longueur :=0;
41      repeat
42          longueur :=longueur+1;
43      until Tab[longueur] = '';
44      longueur :=longueur-1;
45  end;
46
47  //focntion qui renvoie un booleéen indiquant la présence d'une chaine de
    caractères dans un tableau
48  function inArray (mot : String; tab :Categories): Boolean;
49  var rep : boolean;
50      i: Integer;
51  begin
52      rep:=False;
53      for i:=1 to longueur(tab) do
54      begin
55          if tab[i] = mot then rep := True;
56      end;
57      inArray := rep;
58  end;
59
```

```
60  //procedure qui va lire le fichier xml et récuperer toutes les informations utiles.
61  procedure ReadXML (NomFich : String; dtd : boolean; var arborescence : arbo);
62      var lect, nomArbo: String;
63          fichier: Text;
64          c : Char;
65          ic, idebut, ifin : Integer;
66
67      begin
68          //ouvrir le fichier
69          assign(fichier, NomFich);
70          reset(fichier);
71          readln(fichier, lect); //passe la ligne type <?xml version="1.0"?>
72
73          //si il y a une partie DTD, il faut la passer
74          if dtd then
75          begin
76              repeat
77                  readln(fichier,lect);
78                  until (eof(fichier)) or (lect = ']>');
79          end;
80
81          //code qui recupere et comprend l'arborescence
82          repeat
83              readln(fichier, lect);
84              if not (lect = '') then //pour éviter le cas de lignes sautées
85              begin
86                  c:='a';
87                  ic := 0;
88                  while (c <> '<') and (not (ic=length(lect))) do
89                      AvancerChar(lect, c, ic);
90                  idebut := ic;
91                  while (c <> '>') and (not (ic=length(lect)+1)) do
92                      AvancerChar(lect, c, ic);
93                  ifin := ic-2;
94                  if lect[idebut]='/' then nomArbo := sliceStr(lect, idebut+1, ifin)
                    else
95                  nomArbo := sliceStr(lect, idebut, ifin); //recupere le nom de la
                    catégorie entre < et >
96
97                  //en fonction de l'indentation, on recupere a quel niveau de
                    l'arborescence la catégorie se situe
98                  if  (idebut = 2)  then arborescence.nom := nomArbo
99                  else if idebut = 3 then
100                     begin
101                     if  not (inArray(nomArbo, arborescence.nv2)) then
                        arborescence.nv2[longueur(arborescence.nv2)+1]:=nomArbo;
102                     end
103                 else if idebut = 4 then
104                     begin
105                     if not inArray(nomArbo, arborescence.nv3) then
                        arborescence.nv3[longueur(arborescence.nv3)+1]:=nomArbo
106                     end
107                 else if idebut = 5 then
108                     if not inArray(nomArbo, arborescence.nv4) then
                        arborescence.nv4[longueur(arborescence.nv4)+1]:=nomArbo
109                 else if idebut = 6 then
110                     if not inArray(nomArbo, arborescence.nv5) then
                        arborescence.nv5[longueur(arborescence.nv5)+1]:=nomArbo
111                 else  writeln('trop de niveaux d''arborescence');
112             end;
113         until (eof (fichier)) ;
```

```
114     end;
115
116 //récupérer une arborescence à interroger à partir d'un fichier xml
117 procedure inputUser(var NomFich : String ; var dtd : Boolean; var arborescence :
    arbo);
118     var c : Char;
119     begin
120         writeln('Comment s''appelle le fichier xml a interroger ? sans erreurs, et
            avec max 5 niveaux d''arborescence');
121         readln(NomFich);
122         writeln('Le fichier a-t-il une partie DTD ? Tapez o ou n ');
123         readln(c);
124
125         if c = 'o' then dtd := True
126         else dtd := False;
127         ReadXML(NomFich, dtd, arborescence);
128     end;
129
130 //  affichage avec crt et selection
131 procedure AfficherEtSelectionner(texte : String; tableau : Categories ; var
    selected : Integer);
132 var i, taille : Integer;
133         touche : Char;
134
135     begin
136         taille := longueur (tableau)+1;
137         selected :=1;
138
139         //affiche toutes les catégories, celle qui est "selectionnée" d'une autre
            couleur
140         repeat
141             ClrScr();
142             writeln (texte);
143             for i:=1 to taille + 1 do
144                 if i = selected then
145                     begin
146                         TextBackground ( Blue );
147                         writeln(tableau[i]);
148                     end
149                 else
150                     begin
151                         TextBackground ( Black );
152                         writeln(tableau[i]);
153                     end;
154
155             //permet la selection grace aux fleches du clavier
156             touche := ReadKey ();
157             case touche of
158                 #0 : begin
159                 touche := ReadKey ();
160                     case touche of
161                         #72 : if selected > 1 then selected := selected-1;
162                         #80 : if selected < taille-1 then selected := selected + 1;
163                     end;
164                 end;
165             end ;
166         until touche =#13 //entrée
167     end;
168
169 function concatArrays (tab1, tab2 : Categories): Categories;
170     var tab : Categories;
```

```
171             i, l : Integer;
172         begin
173             tab := tab1;
174             l := longueur(tab1);
175             for i:= 1 to longueur(tab2) do
176                 tab[l+i] := tab2[i];
177             concatArrays := tab;
178         end;
179
180     procedure QuelleCategorie (arborescence : arbo; var selected : Integer ; var nvsel : Integer; var TabCat : Categories);
181         var texte : String;
182         begin
183             //on va transformer la structure "arborescence" en tableau avec toutes les catégories pour pouvoir afficher et selectionner
184             TabCat := concatArrays(arborescence.nv2, arborescence.nv3);
185             TabCat := concatArrays(TabCat, arborescence.nv4);
186             TabCat := concatArrays(TabCat, arborescence.nv5);
187
188             texte :='Selectionnez la categorie que vous souhaitez interroger avec les fleches haut et bas, puis entree';
189             AfficherEtSelectionner(texte, TabCat, selected );
190
191             if selected <= longueur(arborescence.nv2) then nvsel := 2
192             else if selected <= longueur(arborescence.nv3) then nvsel := 3
193             else if selected <= longueur(arborescence.nv4) then nvsel := 4
194             else nvsel :=5
195
196         end;
197
198     procedure QuelleRequete(LAction : Categories; var selected :Integer);
199     var texte : String;
200         begin
201             texte :='Selectionnez l''action que vous souhaitez effectuer';
202             AfficherEtSelectionner(texte, LAction, selected );
203         end;
204
205     //procedure qui crée un fichier texte avec du code query qui recherche un terme dans une catégorie, et qui sera ensuite traité par saxon
206     //affichera toutes les données dont l'attribut dans une catégorie donnée correspnd à la recherche
207     procedure ecrireQueryRecherche (NomFich : String ; arborescence : arbo ; catselected, recherche : String ; nvsel :Integer );
208     var fichier : Text; //un fichier texte avec du code
209         nomArbo : String;
210         i :Integer;
211         begin
212             assign (fichier, 'queryPascal');
213             rewrite (fichier);
214
215             nomArbo := arborescence.nom;
216             writeln(fichier, '<html>');
217
218             if nvsel >= 2 then
219                 nomArbo := nomArbo + '/'+arborescence.nv2[1];
220
221             writeln(fichier, '{for $a in doc ("' + NomFich + '")//' + nomArbo);
222             writeln(fichier, 'where $a/', catselected, ' = "', recherche, '"');
223             writeln(fichier, 'return');
224             writeln(fichier, '  <ul>{');
225             for i := 1 to longueur (arborescence.nv3)-1 do
```

```
226             writeln(fichier, '    $a/',arborescence.nv3[i],',');
227
228         writeln(fichier, '    $a/',arborescence.nv3[i+1]);
229         writeln(fichier, '  }</ul>');
230         writeln(fichier, '}');
231         writeln(fichier, '</html>');
232
233         close(fichier);
234         writeln('Le fichier query a ete cree, vous pouvez le trouver dans votre
            dossier :)');
235
236     end;
237
238 //récupère toutes les données d'une catégorie
239 procedure ecrireQueryAfficher (NomFich : String ; arborescence : arbo ;
    catselected : String ; nvsel :Integer );
240 var fichier : Text; //un fichier texte avec du code
241     nomArbo : String;
242     begin
243         assign (fichier, 'queryPascal');
244         rewrite (fichier);
245
246         nomArbo := arborescence.nom;
247         writeln(fichier, '<html>');
248
249         if nvsel >= 3 then nomArbo := nomArbo + '/'+arborescence.nv2[1];
250         if nvsel >= 4 then nomArbo := nomArbo + '/'+arborescence.nv3[1];
251
252         writeln(fichier, '{for $a in doc ("' + NomFich + '")//' + nomArbo);
253         writeln(fichier, 'return');
254         writeln(fichier, '  <ul>{');
255
256         writeln(fichier, '    $a/' ,catselected);
257         writeln(fichier, '  }</ul>');
258         writeln(fichier, '}');
259         writeln(fichier, '</html>');
260
261         close(fichier);
262         writeln('Le fichier query a ete cree, vous pouvez le trouver dans votre
            dossier :)');
263
264     end;
265
266 function RecupEntreUL (s :String): Categories;
267 var i, id, ifin, nbRep: Integer;
268     c: Ansistring;
269     tab : array[1..MAXLignes] of Ansistring;
270
271     //en entrée : ce qu'il y a entre 2 balises <ul>
272     //ex :
    <nom>Bloggs</nom><prenom>Fred</prenom><date_de_naissance>2008-11-27</date_de_nai
    ssance><genre>masculin</genre>. on doit le comprendre
273     begin
274     i := 0;
275     nbRep:=1;
276     repeat
277         repeat
278             c:= s[i];
279             i:=i+1;
280         until c = '>';
281         id := i-1;
```

```
282
283             repeat
284                 c:= sliceStr(s, i, i+1);
285                 i:=i+1;
286             until c = '</';
287             ifin := i-1;
288             tab [nbRep]:=sliceStr(s,id+1, ifin-1);
289             nbRep := nbRep + 1;
290             repeat i:=i+1
291             until s[i] = '>';
292             i:=i+1;
293         until i = length(s)-1;
294
295         RecupEntreUL := tab;
296         end;
297
298     procedure comprendreHTML (var Tableau : TabRep; var comptrep : Integer );
299         var fichier :Text;
300         lect: Ansistring ; //type comme String mais non limité a 255 caractères
301         temp, temphtml : String;
302         i, idebut, ifin  : Integer;
303
304     begin
305         assign(fichier, 'sortie.html'); //pr utilisation
306
307         reset(fichier);
308         readln(fichier, lect);
309         if lect <> '<?xml version="1.0" encoding="UTF-8"?><html/>' then //si fichier
        vide, affiche erreur
310         begin
311             comptrep := 1;
312             idebut := 0;
313             ifin := 0;
314             i:= length ('<?xml version="1.0" encoding="UTF-8"?>')-1; //passe le début
315             repeat
316                 //il y a autant de <ul> que de réponses donc je dois récupérer ce quil
                y a entre les ul
317                  //je vais donc recupérer indices de fin du ul ouvrant et de debut du
                  ul fermant
318
319                 temp := sliceStr(lect, i, i+length('<ul>')-1);
320                 if temp = '<ul>' then
321                     begin
322                         idebut := i + length('<ul>') ;
323                         repeat
324                             temp := sliceStr(lect, i, i+length('</ul>')-1);
325                             i:=i+1;
326                         until temp = '</ul>';
327
328                         begin
329                         ifin := i;
330                         Tableau[comptrep] := RecupEntreUL (sliceStr(lect, idebut, ifin));
331                         comptrep := comptrep +1;
332                         end;
333                     end;
334
335                 i:=i+1;
336                 //tester si fin :
337                 temphtml := sliceStr(lect, i, i+length ('</html>')-1);
338             until (temphtml = '</html>');
339             comptrep:=comptrep-1;
```

```
340             close(fichier);
341         end
342         else writeln('ERREUR : le html est vide');
343     end;
344
345     //pour l'affichage : renvoie une string de taille 20 avec un mot au centre et des espaces autour
346     function centrer (s : string): String;
347         var res : String;
348         i, nbespavant, nbespapres : Integer;
349     begin
350         res := '';
351         nbespavant := (20 - length(s)) div 2;
352         if length(s) mod 2 = 0 then nbespapres := nbespavant
353         else nbespapres := nbespavant +1;
354         for i:=1 to nbespavant do
355             res := res+' ';
356         res:=res+s;
357         for i:=1 to nbespapres do
358             res := res+' ';
359         centrer := res;
360     end;
361
362     //après avoir crée le query, récupérer un html avec saxon, l'interpréter et l'afficher
363     procedure OutputUser (nomFich : String; arborescence : arbo ; catselected, ASelected : String);
364     var lect : String;
365         Tab : TabRep;
366         i, j, nbRep : Integer ;
367     begin
368         writeln('Veuillez taper la ligne suivante dans votre terminal avec Saxon svp');
369         writeln(' ');
370         writeln('java -cp saxon-he-12.4.jar net.sf.saxon.Query -t -s:', nomFich, ' -q:queryPascal -o:sortie.html');
371         writeln();
372         writeln('Lorsque c''est fait, tapez "fait" et entrez');
373
374         repeat
375             readln(lect)
376         until(lect = 'fait');
377
378         comprendreHTML(Tab, nbRep);
379         repeat
380             if ASelected = 'recherche' then
381             begin
382                 writeln('les ', arborescence.nv2[1] , 's correspondant(e)s sont :');
383
384                 //affichage d'un 'tableau' avec les résultats
385                 write('|');
386                 for j:= 1 to longueur (arborescence.nv3) do
387                     write( centrer(arborescence.nv3[j]), '|'); //écrit les catégories
388
389                 writeln('');
390                 for i:=1 to 21 * longueur(arborescence.nv3) + 1 do write('-');
391
392                 for i :=1 to nbRep do
393                     begin
394                         writeln('');
395                         write('|');
396                         for j:= 1 to longueur (arborescence.nv3) do
```

```
397                         write(centrer(Tab[i][j]), '|');
398                 end;
399         end
400         else
401         begin
402             j:=1;
403             writeln('Voici la liste de tous les ', catSelected , 's :');
404             repeat
405                 writeln(Tab[j][1]);
406                 j:=j+1;
407             until Tab[j][1]='';
408         end;
409
410         writeln();
411         writeln('Lorsque vous avez fini, tapez fini pour effectuer une autre
            action');
412         readln(lect);
413     until lect = 'fini';
414 end;
415
416
417 var  selected, nvsel :Integer;
418 catSelected, ASelected, recherche, NomFich : String;
419 var  LActions, TabCat : Categories;
420     arborescence : arbo;
421     dtd : Boolean;
422 //Main
423 begin
424     inputUser(NomFich, dtd, arborescence); //on a récupéré l'arborescence avec les
    catégories à partir du xml
425     ASelected :='';
426     while ASelected <> 'fin' do
427     begin
428         LActions[1]:='recherche';
429         LActions[2]:='afficher tout';
430         LActions[3]:='fin';
431         QuelleRequete(LActions, selected); //récupère l'action que l'utilisateur
    veut effectuer
432         ASelected := LActions[selected];
433
434         if ASelected <>'fin' then
435         begin
436             QuelleCategorie(arborescence, selected, nvsel, TabCat); //récupère la
    catégorie que l'utilisateur veut interroger
437             catSelected := TabCat[selected];
438
439             if (ASelected = 'recherche')  then
440             begin
441                 writeln('Avec quel terme voulez-vous comparer ?');
442                 readln(recherche);
443             end
444             else recherche :='';
445
446             writeln();
447             writeln('Vous voulez effectuer l''action ' + ASelected +' avec le
            terme '+ recherche +' sur la categorie ' + catSelected );
448
449             if ASelected = 'recherche' then
450             begin
451                 ecrireQueryRecherche(NomFich, arborescence, catselected,
                recherche, nvsel);
```

```
452                     OutputUser(NomFich, arborescence, catselected, ASelected);
453                 end
454                 else if ASelected = 'afficher tout' then
455                 begin
456                     ecrireQueryAfficher (NomFich, arborescence, catselected, nvsel);
457                     OutputUser(NomFich, arborescence, catselected, ASelected);
458                 end;
459             end;
460         end;
461
462
463     end.
464
```