Auteurs : LARRAZ MARTIN, MARTIN, ELGHISSASSI, JEANNE, LEVRARD

PhonarQL - H4301

I. Présentation du Projet

Objectif Principal

L'objectif principal de ce projet est de se familiariser avec la notion et le fonctionnement du **Web Sémantique**, en exploitant ses technologies pour concevoir un **moteur de recherche intelligent**. Ce moteur est capable de lier différentes informations relatives à un sujet spécifique, en utilisant les données ouvertes et les standards du Web Sémantique, comme **DBpedia** et le langage **SPARQL**.

Choix du Sujet

Nous avons choisi de développer un moteur de recherche dédié aux **smartphones**, un domaine populaire et riche en données. Notre application offre les fonctionnalités suivantes :

- **Recherche intelligente** : rechercher des téléphones et obtenir des informations détaillées.
- **Comparaison** : comparer les spécifications techniques de plusieurs smartphones.
- **Navigation contextuelle** : afficher des détails sur les marques, les composants et les caractéristiques techniques en fonction de leur contexte.

En plus des téléphones, notre moteur permet d'explorer d'autres entités liées, comme les marques et les composants, pour fournir une vision complète et interconnectée du domaine. Les différentes fonctionnalités seront explorées en détail dans la partie V.

II. Technologies choisies

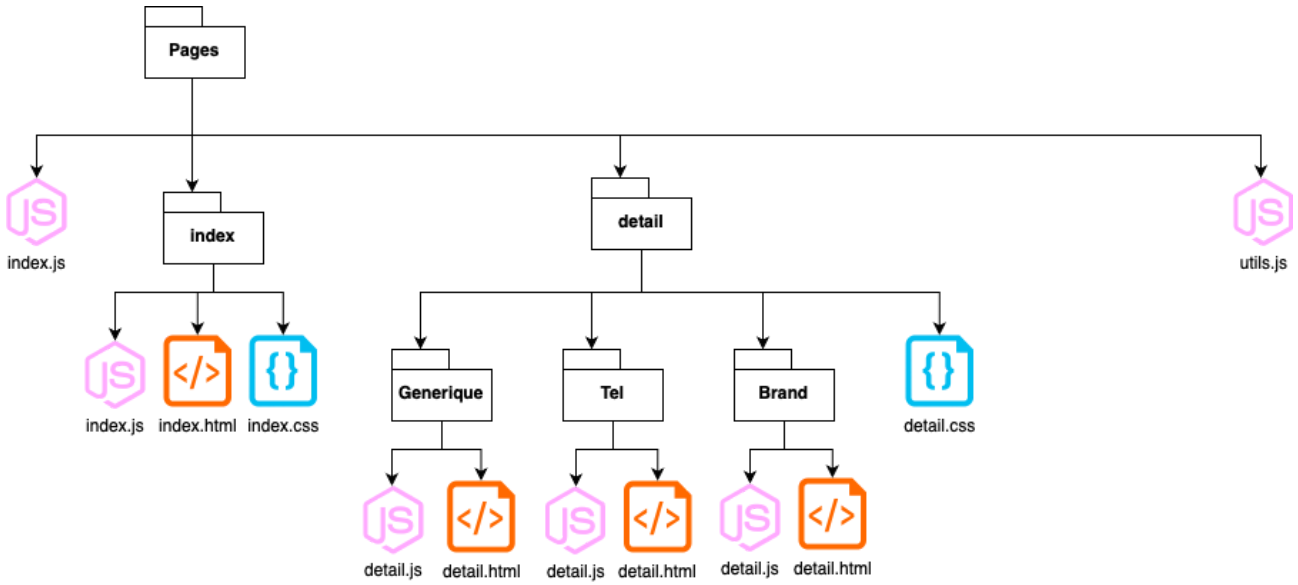
Nous avons décidé de concevoir notre application sous forme de site web (en local). Ainsi, nous avons utilisé de l'HTML, du CSS, et du Javascript (JS). Afin de récupérer les informations de DBpedia, nous avons effectué des requêtes (langage SPARQL) HTTP implémentées sous forme de requêtes AJAX.

III. Architecture générale

L'architecture de notre projet est simplement l'architecture d'une interface web classique.

III.1. Arborescence des fichiers

Le projet suit une architecture modulaire, avec des dossiers organisés par type de contenu et une séparation claire des responsabilités. Voici l'arborescence des fichiers :



III.2. Description des fichiers

III.2.a. Dossier Pages

Le dossier **Pages** contient des outils partagés entre les différents modules de l'application. Le fichier **utils.js**, par exemple, regroupe des fonctions utilitaires utilisées à travers le projet. Ces fonctions permettent d'échapper les caractères spéciaux dans les requêtes SPARQL, de générer dynamiquement des requêtes pour récupérer des propriétés spécifiques, et de créer des liens cliquables vers d'autres entités.

III.2.b. Dossier Index

Le dossier **Index** constitue le point d'entrée principal de l'application. C'est à partir de cette page que l'utilisateur peut effectuer une recherche de téléphones et accéder à leurs détails. Ce dossier comprend trois fichiers principaux : **index.html**, **index.css**, et **index.js**.

- **index.html :** Le fichier **index.html** définit la structure de la page d'accueil. Il comprend une barre de recherche où l'utilisateur peut entrer des mots-clés pour rechercher des téléphones. Les résultats sont affichés sous forme de liste, chaque élément comportant le nom du téléphone, une vignette, et un bouton permettant d'ajouter le téléphone au tableau comparatif. Ce tableau, également intégré dans la page, affiche côte à côte les spécifications des téléphones choisis par l'utilisateur.

- **index.js :** index.js s'occupe de la logique. Il récupère les résultats de recherche en exécutant des requêtes SPARQL vers DBpedia et les affiche dynamiquement. Ce fichier contient également le code nécessaire pour gérer le tableau comparatif, en ajoutant ou en retirant des téléphones selon les choix de l'utilisateur.

- **index.css :** Pour le style, index.css gère l'apparence de la page. Il utilise des couleurs conviviales et une disposition claire pour offrir une navigation fluide. Les différents composants, comme la barre de recherche, la liste des résultats, et le tableau comparatif, sont stylisés pour s'adapter à des écrans de tailles variées, rendant le site responsive.

III.2.c. Dossier Detail

Le dossier **Detail** est conçu pour afficher les informations détaillées sur les entités recherchées, qu'il s'agisse de téléphones, de marques, ou d'entités génériques. L'approche adoptée repose sur une structure modulaire, chaque type d'entité ayant ses propres fichiers. Cela garantit une extensibilité, permettant d'ajouter facilement de nouveaux types d'entités.

Les entités sont d'abord classées grâce à une requête SPARQL qui détermine leur type.
Cette requête est lancée lorsque l'utilisateur tente de charger une page de détail en cliquant sur un lien. Si une entité correspond à un type pris en charge (comme un téléphone ou une marque), l'utilisateur est redirigé vers la page dédiée. Sinon, une page générique est utilisée pour afficher toutes les informations disponibles.

Dans le sous-dossier **Tel**, dédié aux smartphones, la page **detail.html** présente les caractéristiques techniques et les spécifications des téléphones dans un format structuré. Les informations sont organisées en catégories (par exemple, « Système d'exploitation », « Batterie », « Caméra ») pour une lecture facile. Le fichier **detail.js** s'occupe de récupérer ces informations via des requêtes SPARQL et de les afficher dynamiquement sur la page.

Le sous-dossier **Brand** est dédié aux marques de téléphones. Sa page **detail.html** affiche des informations générales sur la marque, telles que son fondateur, son pays d'origine, et une description. Les logos ou images de la marque sont également intégrés. Le fichier **detail.js** de ce sous-dossier gère les requêtes SPARQL nécessaires pour récupérer ces informations.

Le sous-dossier **Generique** traite les entités qui ne correspondent à aucun type spécifique. Sa page **detail.html** affiche un tableau listant toutes les propriétés disponibles de l'entité, sans catégorisation préalable. Le fichier **detail.js** utilise une approche générique pour récupérer et afficher ces propriétés.

Le fichier **detail.css**, commun à toutes les pages du dossier **Detail**, apporte une uniformité visuelle. Il gère l'apparence des tableaux, des images, et des catégories principales et secondaires, en s'assurant que chaque page reste claire et bien organisée.

IV. Fonctionnement

L'application repose sur des requêtes SPARQL dynamiques pour interagir avec les données disponibles sur **DBpedia**. Ces requêtes permettent de récupérer, organiser et afficher les informations relatives aux smartphones, marques et autres sujets dans des catégories bien définies. Voici une analyse détaillée des principaux types de requêtes utilisées dans le projet.

Ces requêtes sont réalisés sur les fichiers javascript grâce à Ajax en local avec une requête GET où:

- Nous l'envoyons au serveur sparql de dbpedia utilisé en TP et contenant la plupart des préfixes.
- Nous encodons par une fonction de base js la requête SPARQL sous format string pour tenir en compte les caractères spéciaux.
- Nous spécifions que le résultat à nous fournir doit être JSON pour faciliter la lecture.

```
"https://dbpedia.org/sparql" + "?query=" + encodeURIComponent(query) + "&format=json";
```

La requête Asynchrone est envoyée et une fois le résultat reçu nous traitons les données et complétons le html.

IV.1. Requêtes pour la recherche de smartphones

Lorsque l'utilisateur effectue une recherche via la barre dédiée, une requête SPARQL interroge DBpedia pour identifier les entités correspondantes. Cette requête combine des filtres textuels et structurels pour s'assurer que seuls des smartphones ou des appareils similaires sont retournés.

Exemple de requête :

```
SELECT DISTINCT ?tel ?label ?thumbnail WHERE {
  {
    ?tel a dbo:Device;
      dbo:abstract ?abstract;
      dbp:cpu ?cpu.
    FILTER (regex(?abstract, "smartphone", "i")).
  }
  UNION
  {
    ?tel dbp:type dbr:Smartphone;
      dbo:abstract ?abstract.
  }
  ?tel rdfs:label ?label.
  FILTER (lang(?abstract) = "en").
  FILTER (lang(?label) = "en").
  FILTER (regex(?label, ".*${searchTxt}.*", "i")).
  OPTIONAL { ?tel dbo:thumbnail ?thumbnail }
}
```

Points importants :

1. **Filtrage par langue :** La requête utilise FILTER (lang(?abstract) = "en") pour limiter les résultats aux descriptions en anglais.
2. **Recherche par mots-clés :** FILTER (regex(?label, ".*${searchTxt}.*", "i")) permet de faire correspondre les étiquettes avec le texte saisi par l'utilisateur.
3. **Union pour plus de flexibilité :** Deux blocs permettent d'inclure des entités catégorisées explicitement comme smartphones (dbp:type dbr:Smartphone) et celles ayant des descripteurs pertinents (dbo:abstract mentionnant "smartphone").

IV.2. Requêtes pour ajouter un smartphone au tableau comparatif

Lorsque l'utilisateur sélectionne un smartphone à comparer, une requête est exécutée pour extraire ses caractéristiques principales. Les informations sont regroupées grâce à la fonction GROUP_CONCAT, qui rassemble les valeurs multiples dans une seule chaîne, séparées par des virgules.

Exemple de requête :

```
SELECT DISTINCT
   (GROUP_CONCAT(DISTINCT ?os; SEPARATOR=", ") AS ?os),
   ?name,
   ?thumbnail,
   (GROUP_CONCAT(DISTINCT ?cpu; SEPARATOR=", ") AS ?cpu),
   (GROUP_CONCAT(DISTINCT ?gpu; SEPARATOR=", ") AS ?gpu),
   (GROUP_CONCAT(DISTINCT ?memory; SEPARATOR=", ") AS ?memory),
   (GROUP_CONCAT(DISTINCT ?storage; SEPARATOR=", ") AS ?storage),
   (GROUP_CONCAT(DISTINCT ?display; SEPARATOR=", ") AS ?display),
   (GROUP_CONCAT(DISTINCT ?rearCamera; SEPARATOR=", ") AS
?rearCamera),
   (GROUP_CONCAT(DISTINCT ?battery; SEPARATOR=", ") AS ?battery)
WHERE {
   dbr:${ressource} rdfs:label ?name.
   FILTER(lang(?name) = "en") .
   OPTIONAL { dbr:${ressource} dbp:os ?os. }
   OPTIONAL { dbr:${ressource} dbp:cpu ?cpu. }
   OPTIONAL { dbr:${ressource} dbp:gpu ?gpu. }
   OPTIONAL { dbr:${ressource} dbp:memory ?memory. }
   OPTIONAL { dbr:${ressource} dbp:storage ?storage. }
   OPTIONAL { dbr:${ressource} dbp:display ?display. }
   OPTIONAL { dbr:${ressource} dbp:rearCamera ?rearCamera. }
   OPTIONAL { dbr:${ressource} dbp:battery ?battery. }
   OPTIONAL { dbr:${ressource} dbo:thumbnail ?thumbnail. }
}
```

Points importants :

1. **Agrégation des valeurs multiples :** GROUP_CONCAT permet d'afficher plusieurs options pour des attributs comme memory (4 GB, 6 GB).
2. **Récupération des images :** Les vignettes des téléphones sont récupérées via dbo:thumbnail.
3. **Structure conditionnelle :** L'utilisation d'OPTIONAL garantit que l'absence de certaines informations n'empêche pas la récupération d'autres.

IV.3. Requêtes pour obtenir les détails d'un sujet

Lorsque l'utilisateur clique sur un lien, une requête est utilisée pour déterminer dynamiquement le type du sujet et rediriger vers la page appropriée.

Exemple de requête pour identifier le type :

```
SELECT ?isWhat WHERE {
   dbr:${ressource} dbp:type ?type.
   BIND(IF(
      regex(?type, ".*PHONE.*", "i") || regex(?type, ".*PHABLET.*", "i"),
      "Tel",
      IF(
         regex(?type, ".*COMPANY.*", "i"),
         "Brand",
         "Generique"
      )
   ) AS ?isWhat)
}
```

Points importants :

1. **Catégorisation dynamique :**
   - regex(?type, ".*PHONE.*", "i") identifie les smartphones.
   - regex(?type, ".*COMPANY.*", "i") identifie les marques.
   - Les autres sujets sont classés comme "génériques".
2. **Redirection basée sur le type :** Les résultats sont utilisés pour naviguer vers des pages spécifiques (detail/Tel, detail/Brand) ou génériques.

IV.4. Requêtes pour afficher les détails d'un smartphone

Une fois le type de l'attribut déterminé (smartphone, marque ou générique), nous affichons des informations pertinentes pour cet attribut. Nous détaillerons dans ce rapport uniquement la logique de construction des détails d'un attribut de type smartphone, puisqu'il s'agissait du but principal de notre moteur de recherche.

a) Fonctionnement

Pour afficher les caractéristiques d'un téléphone, une requête regroupe les informations par catégorie.

Exemple de requête pour les caractéristiques générales :

```
SELECT DISTINCT
   (GROUP_CONCAT(DISTINCT ?name; SEPARATOR=" | ") AS ?name),
   (GROUP_CONCAT(DISTINCT ?abstract; SEPARATOR=" | ") AS ?abstract),
   (GROUP_CONCAT(DISTINCT ?brand; SEPARATOR=" | ") AS ?brand),
   (GROUP_CONCAT(DISTINCT ?manufacturer; SEPARATOR=" | ") AS
?manufacturer)
WHERE {
   OPTIONAL { dbr:${ressource} foaf:name ?name. }
   OPTIONAL { dbr:${ressource} dbo:abstract ?abstract. FILTER(LANG(?abstract)
= "en") }
   OPTIONAL { dbr:${ressource} dbp:brand ?brand. }
   OPTIONAL { dbr:${ressource} dbp:manufacturer ?manufacturer. }
}
```

Points importants :

- **Catégories et sous-catégories :** Les informations sont structurées en catégories principales (Caractéristiques Techniques, Informations Complémentaires) et sous-catégories (CPU, Mémoire).
- **Gestion des images :** Si des propriétés comme depiction ou thumbnail sont disponibles, elles sont affichées sous forme d'images.

b) Méthodologie de développement

Dans le cadre de ce projet, nous avons développé un script Python pour explorer de manière systématique les données disponibles sur DBpedia pour une liste pré-définie de smartphones. Ce script a joué un rôle crucial dans la conception de notre application en permettant de déterminer les attributs pertinents à extraire via des requêtes SPARQL dans l'application. Vous pourrez retrouver ce script dans le dossier **Demarche_Attributs.**

Objectifs du script

1. **Exploration exhaustive des données DBpedia :**

Le script interroge DBpedia pour récupérer l'ensemble des propriétés et valeurs associées à chaque smartphone dans une liste donnée. Cela permet d'avoir une vision complète de ce qui est disponible dans la base de données.

2. **Analyse des données récupérées :**

Les informations collectées sont structurées sous forme de tableaux comparatifs. Ces tableaux offrent une vue d'ensemble des propriétés disponibles pour chaque smartphone, facilitant l'identification des attributs couramment renseignés ou pertinents.

3. **Définition des attributs cibles pour l'application :**

À partir des tableaux générés, nous avons analysé les données pour sélectionner les attributs les plus utiles à extraire dans l'application web (par exemple, les caractéristiques techniques, les images, les fabricants).

Résultats obtenus

1. **Vision complète des données disponibles** :

Le script a permis d'identifier toutes les propriétés disponibles sur DBpedia pour une liste de smartphones. Cela inclut des informations générales (nom, description, fabricant) et des caractéristiques techniques (CPU, RAM, stockage, batterie).

2. **Identification des attributs récurrents et utiles** :

En analysant les tableaux générés, nous avons identifié les attributs les plus communs et les plus informatifs. Par exemple :

- **Informations générales** : dbo:abstract, dbp:brand, dbo:manufacturer
- **Caractéristiques techniques** : dbp:cpu, dbp:memory, dbp:storage, dbp:battery
- **Images** : dbo:thumbnail, dbp:logo

Cela nous a permis de concevoir des requêtes SPARQL ciblées pour extraire uniquement les informations pertinentes dans l'application.

3. **Identification des lacunes dans les données** :

Le script a également révélé des lacunes dans les données DBpedia, comme des valeurs manquantes pour certains attributs (par exemple, dbp:gpu ou dbp:charging). Ces observations ont orienté notre choix de présentation et de gestion des données dans l'application. Un exemple de tableau nous ayant permis d'établir la liste des attributs

pertinents se trouve dans **Demarche_attributs/smartphone_comparison_full.csv.**

V. Description des fonctionnalités

Le choix du sujet a été une étape importante car il était primordial de trouver un sujet qui nous plaisait et avec lequel il était possible de lier des ressources entre elles. C'est pourquoi nous avons choisi le thème des téléphones. En effet, il y avait beaucoup d'informations sur DBpedia même si nous avons tout de même rencontré divers problèmes que nous détaillerons davantage dans la partie suivante (VI).

V.1. Recherche et comparaison de téléphones

La première fonctionnalité que notre moteur de recherche propose est la recherche de téléphones par nom. Nous avons fait en sorte que la casse ne soit pas prise en compte, c'est à dire que par exemple, si l'utilisateur saisit "aTiV" dans la barre de recherche, notre moteur de recherche proposera, entre autres, : "Samsung Ativ S" et "Samsung ATIV SE". L'utilisateur peut donc faire ses recherches sans se soucier de la casse.

Une deuxième fonctionnalité proposée par notre application est la comparaison de téléphones. Après que l'utilisateur a fait une recherche, il peut évidemment consulter les résultats de celle-ci mais il peut aussi sélectionner un ou des téléphones apparaissant dans les résultats afin de les ajouter à un tableau comparatif en bas de page (cf. image ci-dessus). Ce tableau permet de

comparer des composants du téléphone tels que le GPU ou le CPU. Il est possible d'ajouter au même tableau comparatif, des résultats de différentes recherches. Il est également possible de retirer du tableau comparatif un téléphone qui avait été sélectionné auparavant.

V.2. Consulter les détails d'un téléphone

IPhone 13

Informations Générales	
name	iPhone 13, iPhone 13 Mini
abstract	The iPhone 13 and iPhone 13 Mini (stylized as iPhone 13 mini) are smartphones designed, developed, marketed, and sold by Apple Inc. They are the fifteenth generation of iPhones (succeeding the iPhone 12 and iPhone 12 Mini respectively). They were unveiled at an Apple Event in Apple Park in Cupertino, California on September 14, 2021, alongside the higher-priced iPhone 13 Pro and iPhone 13 Pro Max flagships. Pre-orders for the iPhone 13 and iPhone 13 Mini began on September 17, 2021. They were officially released on September 24, 2021.
brand	
manufacturer	Pegatron, Foxconn
Caractéristiques Techniques	
Système d'exploitation et SoC	
os	Original: iOS 15.0
soc	Apple A15
CPU et GPU	
cpu	Hexa-core – 2.02 GHz
gpu	Apple-designed 4 core
Mémoire et Stockage	
memory	4
storage	128256512
Écran et Caméras	
display	13
frontCamera	3.6, 4, HDR
rearCamera	12, 4, Dual-LED dual-tone flash, HDR

De plus, notre application permet de consulter les détails d'un smartphone. Pour cela, après avoir effectué une recherche, l'utilisateur peut cliquer sur l'un des smartphones proposés. Il a également la possibilité d'afficher les détails d'un smartphone figurant dans le tableau comparatif.

Les différentes informations sur le téléphones sont réparties dans 4 sections différentes : "Informations Générales", "Caractéristiques Techniques", "Informations Complémentaires", "Historique et Relations". Il y a également une image du téléphone qui est affichée.

Voici le contenu possible (si une information n'est pas présente alors le cadre est vide) de chacune de ces sections :

Informations Générales

- Name
- Abstract
- Brand
- Manufacturer

Caractéristiques Techniques

Système d'exploitation et SoC :

- OS
- SOC

CPU et GPU :

- CPU
- GPU

Mémoire et Stockage :

- Memory
- Storage

Écran et Caméras :

- Display
- FrontCamera
- RearCamera

Batterie et Chargement :

- Battery
- Charging

Connectivité et Réseaux :

- Connectivity).
- Networks).

Étanchéité :

- WaterResist

Informations Complémentaires

- MemoryCard
- Input
- Form
- Colors
- Weight

Historique et Relations

- Released
- Available
- Predecessor

- Successor

Notons également qu'à partir de cette page de détail d'un téléphone, il est possible de naviguer sur la page de détail de la marque associée (que nous détaillerons dans la partie suivante). Mais, il est également possible de naviguer vers la page de détail, via un lien, de n'importe quelle information qui a été obtenue et qui est une URI. Nous détaillerons cela dans la suite.

V.3. Consulter les détails d'une marque

De surcroît, notre application permet de consulter les détails d'une marque de téléphone en particulier. En effet, à partir d'une page de détail d'un téléphone, l'utilisateur peut naviguer vers la page de détail de la marque du téléphone en cliquant sur la valeur de 'Brand' (section "Informations Générales"). Notons que cela n'est possible que si une URI avait été fournie pour la marque sur DBpedia.

Voici les différentes informations que nous affichons sur la marque (en plus d'une image) :

Informations Générales

- Label

- Abstract
- Founder
- Country
- Comment

De la même manière que pour la page de détail d'un téléphone, il est possible de naviguer vers la page de détail, via un lien, de de n'importe quelle information qui a été obtenue et qui est une URI.

Nous détaillerons cela dans la suite.

V.4. Consulter les détails de toute ressource obtenue sous forme d'URI

Enfin, notre application offre la possibilité de consulter les détails de toute ressource obtenue sous forme d'URI, et ce, sur l'ensemble de nos pages. Par exemple, à partir de la page de détail d'un smartphone, il est possible d'accéder aux détails sur l'OS, le SOC, la série de smartphones associée, la 4G, un de ses capteurs, etc.

Pour toutes ces ressources, nous présentons l'ensemble des informations disponibles sur DBpedia. En effet, nous ne pouvons pas filtrer les prédicats pertinents car ils varient fortement entre chaque ressource. Même si, des filtrages sont réalisés sur des éléments communs non-nécessaires et non pertinents pour éviter de les afficher (cf. VI.4)

Ces informations sont affichées sur une page générique. Comme mentionné précédemment, il serait envisageable, avec davantage de temps, de personnaliser la page de n'importe quel type de ressource, de la même manière que pour les téléphones ou les marques. Cela permettrait de spécifier les informations à présenter.

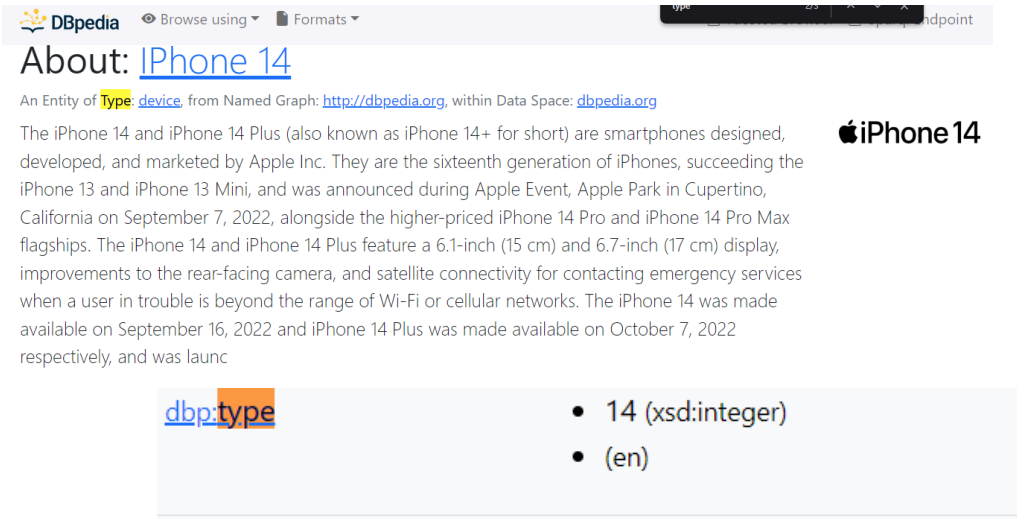
VI. Problèmes rencontrés

VI.1. (AAA) Données non typées.

Un problème majeur que nous avons rencontré est le manque de typage des smarphones. Cela se présente par:

- un typage trop générique (souvent “device”) qui prenait donc trop de ressources qui n’étaient pas des smartphones (imprimantes, ordinateurs, …)

- Un typage manquant ou incohérent ne permettant pas de déterminer si le sujet est un smartphone ou pas ce qui rend beaucoup de smartphones invisibles à la recherche.
  exemple: Iphone 14 typé comme “device”, “14” et “(vide)”.



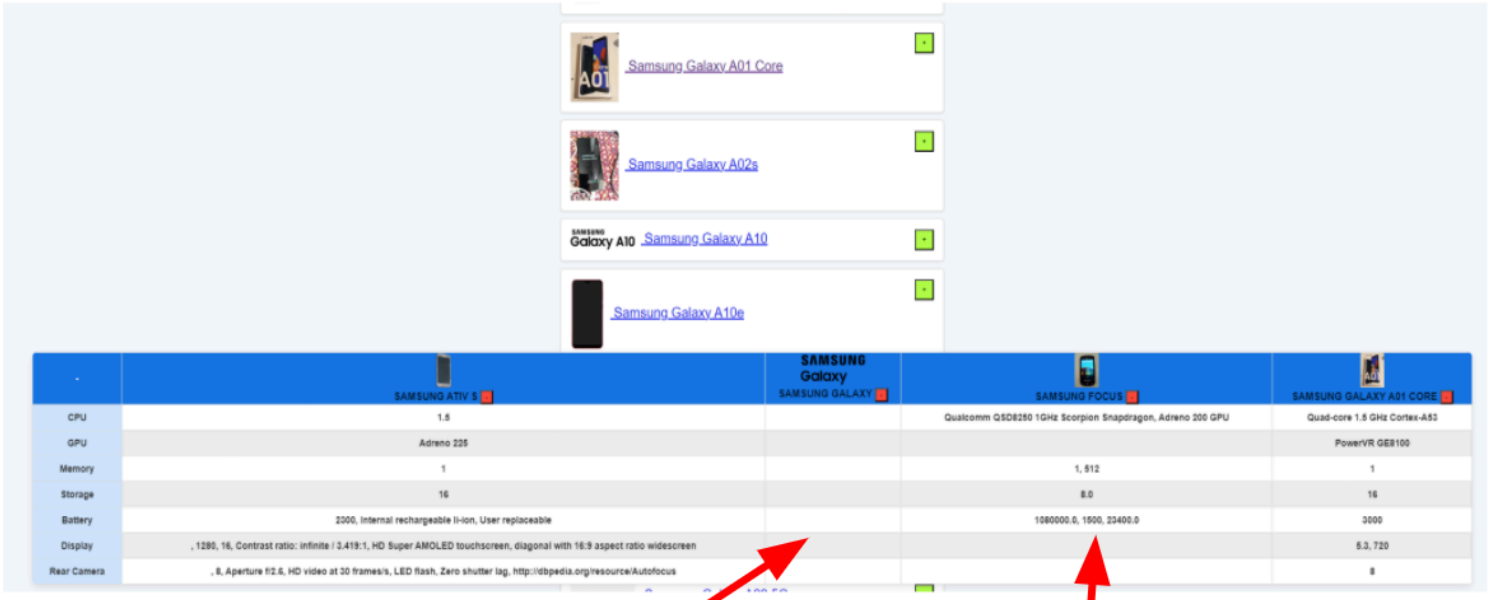
La solution est a été présentée en détail dans le Fonctionnement (cf. IV.2) et se base sur la lecture du regex “.*phone.*”, des abstraits et de leur nom, et de la présence de prédicats propres aux smartphones.

Cette lecture permet d'obtenir la grande majorité des smartphones se trouvant sur dbpedia.

Nous voulons préciser aussi que nous avons envisagé l’utilisation de wikidata pour une obtention de données plus structurée mais il n’y avait presque pas de modèles de smartphones dans leur base, surtout des images. Nous restons

De même, lors de la redirection, ces smartphones ne présentant pas de typage correct sont redirigés vers générique. Cela ne pose pas de problème véritable car leur informations sont souvent manquantes et donc la vue générique reste pertinente.

VI.2. (AAA) Données incohérentes, incomplètes et hétérogènes.



-	SAMSUNG ATIV S	SAMSUNG GALAXY	SAMSUNG FOCUS	SAMSUNG GALAXY A01 CORE
CPU	1.5		Qualcomm QSD8250 1GHz Scorpion Snapdragon, Adreno 200 GPU	Quad-core 1.5 GHz Cortex-A53
GPU	Adreno 225			PowerVR GE8100
Memory	1		1, 512	1
Storage	16		8.0	16
Battery	2300, Internal rechargeable li-ion, User replaceable		1080000.0, 1500, 23400.0	3000
Display	, 1280, 16, Contrast ratio: infinite / 3.419:1, HD Super AMOLED touchscreen, diagonal with 16:9 aspect ratio widescreen			5.3, 720
Rear Camera	, 8, Aperture f/2.6, HD video at 30 frames/s, LED flash, Zero shutter lag, http://dbpedia.org/resource/Autofocus			8

Beaucoup de valeurs vides!
=> OPTIONALS ! et donc requêtes inefficaces.

Données incohérentes et hétérogènes (!= unités)

Pour ce qui concerne les smartphones sur dbpedia, leur objets étaient souvent vides ou avec des valeurs incohérentes (une batterie sans unité ou de un million !).
Ce n'est pas que leur valeur, certains prédicats n'étaient directement pas présents ce qui complexifie et rend inefficaces les requêtes par l'ajout d'OPTIONALs.

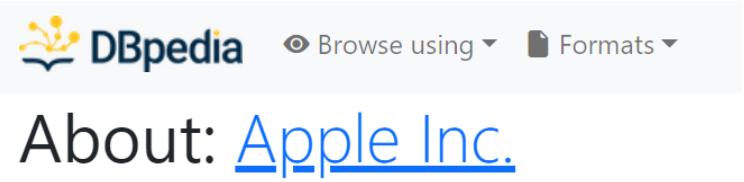
Heureusement cela était surtout chez les vieux modèles mais les nouveaux n'en échappent pas.

VI.3. (AAA) Caractères spéciaux sur le nom des sujets référencés

Certains noms de sujets contiennent des caractères spéciaux qui donnaient erreur lors de la requête SPARQL si mis comme sujet directement dans le triplet de recherche.

Exemples :

- Un point à la fin:



- Des parenthèses:

Notre solution a été simple, l'entourer de "< … >" et avec le préfixe complet.

```
'<http://dbpedia.org/resource/" + ressource + ">";
```

VI.4. Surcharge du serveur face à des grandes requêtes

Un problème récurrent rencontré lors de l'élaboration du projet est la surcharge du serveur DBpedia lorsque des requêtes SPARQL impliquent un grand volume de données ou de résultats. Ce problème est exacerbé par plusieurs facteurs liés à la structure même des données sur DBpedia et à la nature des requêtes nécessaires pour interroger ces données.

Contexte du Problème

Les requêtes impliquant des agrégations (par exemple, GROUP_CONCAT) ou de nombreux filtres (comme regex pour rechercher des mots-clés ou des correspondances dans les chaînes) peuvent générer une charge importante sur le serveur SPARQL. Lorsqu'un grand nombre de résultats ou des ensembles de données non restreints sont traités, les temps de réponse augmentent considérablement, voire aboutissent à des échecs de requêtes.
Par exemple :

- Une requête non optimisée pour rechercher tous les smartphones peut inclure des entités non pertinentes (imprimantes, ordinateurs, etc.), entraînant des résultats volumineux et inutiles.
- Des entités mal typées ou avec des informations incomplètes peuvent multiplier les résultats, augmentant ainsi le nombre d'éléments que le serveur doit traiter.

Approches et Solutions

Pour réduire la surcharge du serveur et optimiser les performances de l'application, plusieurs stratégies ont été mises en place :

1. **Limitation des résultats via des filtres pertinents**

En ajoutant des filtres précis, nous avons restreint les résultats des requêtes à ceux qui sont vraiment pertinents. Par exemple, nous avons utilisé des combinaisons de FILTER pour cibler uniquement les entités avec des descriptions contenant le mot "smartphone" ou avec des propriétés spécifiques comme dbp:type dbr:Smartphone. Cela permet d'exclure les entités génériques ou non pertinentes.

Exemple :

```
FILTER (regex(?abstract, "smartphone", "i"))
FILTER (lang(?label) = "en")
```

2. **Gestion de la langue des résultats**

L'un des principaux facteurs contribuant à la surcharge est la présence d'objets multilingues dans DBpedia. Pour éviter que chaque entité ne retourne des résultats dans plusieurs langues, nous avons appliqué un filtre pour ne récupérer que les informations en anglais (lang(?abstract) = "en"). Cela réduit considérablement la taille des résultats.

3. **Utilisation d'agrégations conditionnelles**

L'agrégation avec GROUP_CONCAT a été utilisée pour rassembler les valeurs multiples d'une même

propriété en une seule chaîne de caractères. Cela diminue le nombre de résultats individuels renvoyés par le serveur tout en conservant l'intégrité des données.

Exemple :
(GROUP_CONCAT(DISTINCT ?cpu; SEPARATOR=", ") AS ?cpu)

4. **Récupération par lots et limitation des résultats**

Pour éviter de surcharger le serveur en récupérant toutes les données en une seule fois, nous avons découpé certaines opérations en requêtes distinctes, chaque requête se concentrant sur une catégorie spécifique de données (par exemple, "Caractéristiques Techniques" ou "Historique"). Cette approche réduit la complexité des requêtes individuelles.

5. **Gestion asynchrone des requêtes**

En intégrant AJAX asynchrone dans notre application, nous avons permis une exécution non bloquante des requêtes SPARQL. Cela garantit que même si le serveur prend plus de temps à répondre, l'expérience utilisateur reste fluide, sans blocage de l'interface.

6. **Timeout et gestion des erreurs**

Des mécanismes de gestion des erreurs ont été mis en place pour gérer les cas où le serveur ne peut pas répondre à temps ou retourne un trop grand volume de données. Dans de tels cas, des messages informatifs sont affichés à l'utilisateur.

Résultats

Ces optimisations ont permis de maintenir les performances de l'application à un niveau acceptable, même lorsque des données volumineuses sont interrogées. Les requêtes sont exécutées dans un délai raisonnable, et les utilisateurs bénéficient d'une expérience fluide malgré les limites inhérentes à DBpedia.

Cependant, il est important de noter que certaines limites techniques du serveur SPARQL de DBpedia restent inévitables, notamment lorsque le serveur est sous forte charge ou que des données spécifiques manquent ou sont mal structurées. Dans ces cas, des ajustements supplémentaires pourraient être envisagés, comme l'utilisation de caches ou l'intégration d'une base de données intermédiaire pour alléger la charge sur DBpedia.

VI.5. (AAA) Filtrer les informations pertinentes sans connaître les prédicats (Détail Générique)

Lors de l'affichage du détail générique, bien que la plupart des informations présentées soient intéressantes, certaines caractéristiques qui sont présentes dans la plupart des sujets de dbpedia ne le sont pas.

Exemple:

SameAs	http://rdf.freebase.com/ns/m.04cnky, http://yago-knowledge.org/resource/Digital_Living_Network_Alliance, http://www.wikidata.org/entity/Q53222, http://ar.dbpedia.org/resource/تحالف_الشبكة_الرقمية_الحية, http://bg.dbpedia.org/resource/Digital_Living_Network_Alliance, http://ca.dbpedia.org/resource/Digital_Living_Network_Alliance, http://cs.dbpedia.org/resource/DLNA, http://da.dbpedia.org/resource/Digital_Living_Network_Alliance, http://de.dbpedia.org/resource/Digital_Living_Network_Alliance, http://es.dbpedia.org/resource/Digital_Living_Network_Alliance, http://fa.dbpedia.org/resource/دی‌الانای, http://fi.dbpedia.org/resource/Digital_Living_Network_Alliance, http://fr.dbpedia.org/resource/Digital_Living_Network_Alliance, http://it.dbpedia.org/resource/Digital_Living_Network_Alliance, http://ja.dbpedia.org/resource/Digital_Living_Network_Alliance, http://ko.dbpedia.org/resource/DLNA, http://ml.dbpedia.org/resource/ഡിജിറ്റൽ_ലിവിങ്ങ്_നെറ്റ്വർക്ക്_അലയൻസ്, http://nl.dbpedia.org/resource/Digital_Living_Network_Alliance, http://no.dbpedia.org/resource/Digital_Living_Network_Alliance, http://pl.dbpedia.org/resource/Digital_Living_Network_Alliance, http://pms.dbpedia.org/resource/DLNA, http://pt.dbpedia.org/resource/DLNA, http://ro.dbpedia.org/resource/DLNA, http://ru.dbpedia.org/resource/DLNA, http://simple.dbpedia.org/resource/Digital_Living_Network_Alliance, http://sv.dbpedia.org/resource/Digital_Living_Network_Alliance, http://tr.dbpedia.org/resource/Digital_Living_Network_Alliance, http://uk.dbpedia.org/resource/DLNA, http://zh.dbpedia.org/resource/数字生活网络联盟, https://global.dbpedia.org/id/4idi1

Nous avons pu le résoudre facilement en ajoutant de nombreux filtres regex sur les prédicats ce qui donne un résultat correct.

De même, la présence de plusieures langues sur plusieurs objets donne lieux à la surcharge du serveur par la multiplication du résultat (trop important). De ce fait nous ne prenons que l'anglais.

Or, tous les objets n'ont pas de langage, notre filtre teste donc 4 cas pour l'objet : si son type admet une langue, si c'est vide, s'il n'a pas de langue ou si sa langue est l'anglais:

```
SELECT DISTINCT ?Property (GROUP_CONCAT(?Value; SEPARATOR = ",  ") AS ?Value)
WHERE {
    ${ressource} ?Property ?Value .
    FILTER (isIRI(?Value) || isBlank(?Value) || LANG(?Value) = "en" || LANG(?Value) = "").
    FILTER (!regex(?Property, ".*type.*","i")) .
    FILTER (!regex(?Property, ".*differentFrom.*","i")) .
    FILTER (!regex(?Property, ".*depiction.*","i")) .
    FILTER (!regex(?Property, ".*thumbnail.*","i")) .
    FILTER (!regex(?Property, ".*logo.*","i")) .
    FILTER (!regex(?Property, ".*subject.*","i")) .
    FILTER (!regex(?Property, ".*SameAs.*","i")) .
    FILTER (!regex(?Property, ".*sid.*","i")) .
    FILTER (!regex(?Property, ".*soc.*","i")) .
    FILTER (!regex(?Property, ".*input.*","i")) .
    FILTER (!regex(?Property, ".*topic.*","i")) .
    FILTER (!regex(?Property, ".*UsesTemplate.*","i")) .
    FILTER (!regex(?Property, ".*Wiki.*Id","i")) .
    OPTIONAL{${ressource} dbo:thumbnail ?Value .}
}
GROUP BY ?Property
LIMIT 60
```

VI.6. Temps d'attente pour la redirection

Du fait que nous utilisons Ajax asynchrone et que pour déterminer le type il nous faut réaliser une requête SPARQL, lorsque nous cliquons sur un lien nous pouvons rencontrer un petit temps d'attente de maximum 1-2 secondes où rien ne se passe avant d'être redirigés vers la page cliquée.

VII. Conclusion et réflexion sur le Web Sémantique

Le projet PhonarQL a permis d'explorer en profondeur les concepts et outils liés au Web Sémantique. À travers la conception d'un moteur de recherche intelligent dédié aux smartphones, nous avons pu exploiter la puissance de DBpedia et du langage SPARQL pour interroger, structurer et présenter des données liées. Le résultat est une application capable non seulement de fournir des informations riches et interconnectées, mais aussi de les organiser de manière compréhensible pour l'utilisateur.
Les objectifs initiaux ont été atteints, notamment :

- La mise en place d'un système de recherche performant capable d'interroger des bases de données ouvertes.
- La comparaison efficace de smartphones en extrayant des attributs spécifiques à partir de données structurées.
- L'intégration de fonctionnalités permettant de naviguer entre des entités liées, élargissant ainsi les perspectives offertes à l'utilisateur.

Malgré les défis rencontrés, tels que les données incohérentes, les lacunes dans le typage, ou les limites techniques des serveurs SPARQL, des solutions innovantes ont été développées. Ces obstacles ont renforcé notre compréhension des avantages et des contraintes du Web Sémantique.

Réflexion sur le Web Sémantique

Le Web Sémantique représente une avancée majeure dans la manière dont les données sont connectées et accessibles. En permettant aux machines de comprendre et de traiter les relations entre des entités, il ouvre la voie à des applications intelligentes capables de naviguer et d'exploiter un réseau d'informations interconnectées. Ce projet nous a permis de tirer plusieurs enseignements importants :

1. **Avantages du Web Sémantique**
    - **Interopérabilité des données** : L'utilisation de standards comme RDF et SPARQL facilite l'intégration de données provenant de différentes sources.
    - **Richesse des relations** : Les bases comme DBpedia offrent une vision holistique d'un sujet grâce aux liens entre entités.
    - **Flexibilité** : Le langage SPARQL permet de concevoir des requêtes complexes pour répondre à des besoins très spécifiques.
2. **Limites actuelles**
    - **Qualité des données** : Les données disponibles sur DBpedia sont parfois incohérentes, incomplètes ou mal structurées, ce qui peut limiter leur utilité.
    - **Performances des serveurs** : La surcharge des serveurs SPARQL pose des défis en

termes de temps de réponse et de fiabilité.

- **Accessibilité** : Bien que les technologies du Web Sémantique soient puissantes, leur mise en œuvre demande une expertise technique significative.

3. **Perspectives et améliorations**

   - **Enrichissement des données** : L'amélioration de la qualité des données, par exemple via Wikidata, pourrait renforcer la fiabilité des applications.

   - **Cache et indexation** : L'intégration de caches locaux ou d'index intermédiaires peut pallier les limites des serveurs SPARQL.

   - **Personnalisation** : Avec des ressources supplémentaires, il serait possible de développer davantage de pages spécifiques aux entités, au-delà des smartphones et marques.

Enjeux futurs

Le Web Sémantique joue un rôle clé dans des domaines variés tels que la recherche, l'intelligence artificielle et l'analyse de données. Des projets comme **PhonarQL** démontrent comment ces technologies peuvent être utilisées pour rendre l'information non seulement accessible, mais aussi intelligible et contextuellement enrichie. Toutefois, pour que le Web Sémantique atteigne tout son potentiel, des efforts concertés devront être faits pour améliorer les données, les outils et l'infrastructure technique qui le sous-te