H430-WS33

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Web Sémantique

Rapport

Chauvet Etienne, Delfosse Charlotte,Galy Adam Marah, Moureau Mathilde, Moutaouakil Oudghiri Youssef, Occelli William, Sahmoudi Zakaria

Présentation du Projet **1**

Technologies choisies **1**

Architecture générale **2**

Arborescence des fichiers 2

Description des fichiers 2

Fonctionnement **3**

Fonctionnement Général 3

Auto-complétion 4

Description des fonctionnalités **4**

Problèmes rencontrés **5**

Conclusion **6**

Annexes **7**

Chauvet E., Delfosse C.,Galy Adam M., Moureau M.,
Moutaouakil Oudghiri Y., Occelli W., Sahmoudi Z.

1. Présentation du Projet

L'objectif principal du projet est de se familiariser avec la notion et le fonctionnement du Web Sémantique. Pour cela, nous devons proposer un moteur de recherche intelligent permettant de lier différentes informations relatives à l'objet de la recherche effectuée.

Nous avons choisi de concentrer notre domaine de recherche aux divinités de la mythologie grecque, et de récupérer un maximum d'informations concernant le divinité choisie.

Les informations retournées par l'application sont le nom du dieu, sa fonction, sa demeure, son genre, ses symboles, ses parents proches (père, mère, frères, soeurs et compagnons) ainsi que les jeux vidéos, films et oeuvres d'art dans lesquels il est mentionné.

2. Technologies choisies

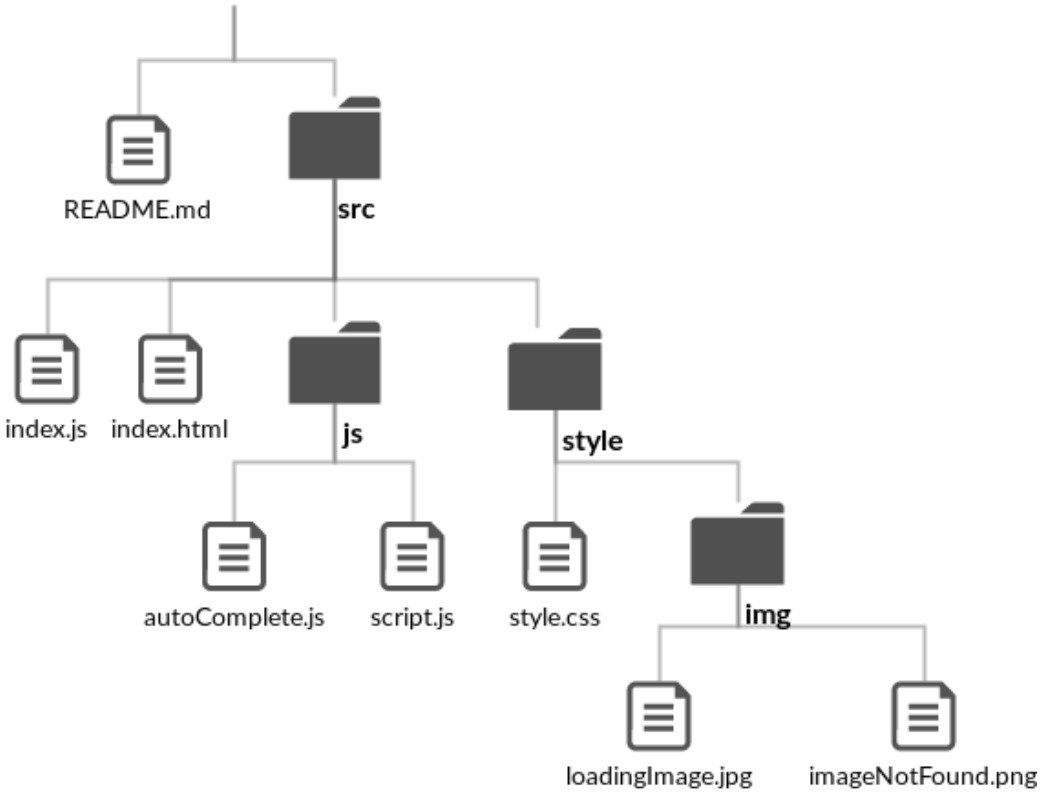
Nous avons utilisé le langage SPARQL pour requêter DBpedia afin d'obtenir des données RDF disponibles à travers l'Internet. Pour le site web, nous avons utilisé HTML, CSS, et Javascript (JS).

Pour recueillir les informations de DBpedia dont nous avions besoin, nous avons effectué des requêtes HTTP sous forme de requêtes AJAX car elles ont l'avantage de faciliter l'appel et le suivi des requêtes HTTP.

3. Architecture générale

L'architecture de notre projet est celle d'une simple interface web. Nous ne nous sommes pas basés sur une architecture client-serveur pour notre projet car nous pouvions faire plus simplement sans.

a. Arborescence des fichiers



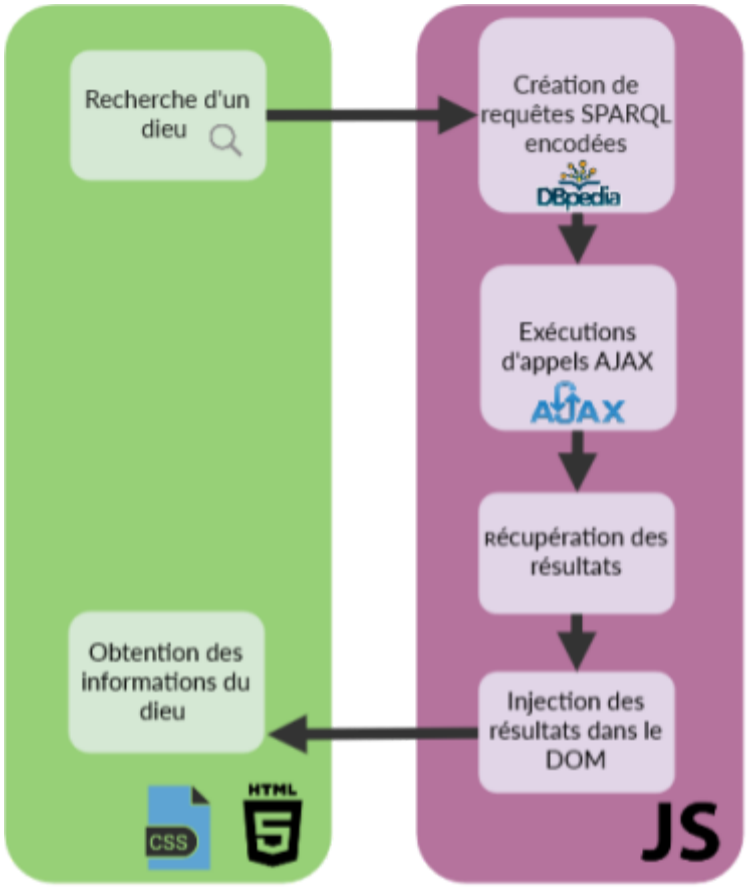
b. Description des fichiers

- Le fichier README contient la description du projet avec la commande à taper pour lancer le projet.
- Le fichier index.html contient le squelette de notre page web : principalement, une barre de recherche et un tableau pour contenir les informations des dieux recherchés.

- Le dossier "js" contient nos services métier : script.js contient le code nécessaire pour récupérer des ressources à partir de nos requêtes SPARQL, situées dans le même fichier. autoComplete.js contient le code permettant d'effectuer de l'auto-complétion dans la barre de recherche.
- Nous avons également implémenté un serveur HTTP local en NodeJS dans le fichier index.js.

4. Fonctionnement

a. Fonctionnement Général



Workflow de l'application

Ci-dessus figure un déroulé du fonctionnement de notre site. Pour fournir un résultat à l'utilisateur lors de sa recherche, nous avons construit notre propre API en utilisant le web service de requétage SPARQL. Celui-ci peut être couplé à une page HTML comme celle de Virtuoso, l'exécuteur en ligne de requêtes SPARQL que nous avons manipulé en cours. Nous avons alors effectué une requête AJAX dans script.js de type HTTP GET pour obtenir des informations pour chacune de nos requêtes SPARQL.

Les url que nous avons demandées se décomposent en plusieurs éléments :

- Un suffixe qui correspond au point d'accès SPARQL (url de la page d'accueil de Virtuoso) "http://dbpedia.org/sparql " .
- Un et un seul paramètre "query" égal à chaque requête SPARQL que nous avons encodée afin qu'elle corresponde au format d'écriture d'une url (remplacement des espaces et des caractères non ASCII avec un "%" suivi de deux digits).
- Un paramètre "default-graph-uri" facultatif et permettant d'explorer par défaut la base de connaissance de DBPedia.
- Un paramètre de format de sérialisation de la réponse en JSON. Il a pour valeur "**application/sparql-results+json**". Nous avons choisi le format de données JSON car il est facilement exploitable en Javascript (il a été conçu pour) et qu'il est plus léger que le XML.

Les résultats récupérés par la requête AJAX sont ensuite insérés dans des tableaux pour être injectés dans la grappe HTML. Le tout est mis en forme grâce à des feuilles de style.

L'utilisateur n'a donc qu'à recherché un nom de dieu, soit en cliquant sur un des dieux proposés par l'autocomplétion soit directement puis appuyer sur la touche entrée ou sur le bouton de recherche. Il obtient alors le résultat de sa requête sous forme de tableau d'informations sur le dieu.

b. Auto-complétion

Pour l'auto-complétion, nous avons effectué une unique requête AJAX avec dans l'url une requête permettant de retourner tous les noms de dieux grecs. Nous récupérons le résultat en JSON et nous trions les valeurs récupérées par ordre alphabétique.

5. Description des fonctionnalités

Le choix du sujet était une étape importante pour nous. Il nous paraissait primordial de trouver un sujet intéressant avec lequel nous pouvions lier les ressources entre elles. C'est pourquoi la mythologie Grecque nous est apparue comme un sujet de choix. La généalogie des différents Dieux, correctement renseignée sur Wikipédia, nous permet d'avoir une base solide pour créer des liens et former une arborescence.

Nos fonctionnalités sont donc les suivantes: pour chaque Dieu, nous affichons son nom, sa fonction ainsi que son image en en-tête. Puis, nous renseignons sous forme d'un tableau, son genre, sa demeure, ses symboles ainsi que ses conjoint(e)s, frères et soeurs, enfants et parents, les jeux vidéos, films et oeuvres d'art dans lesquels le Dieu apparaît. Pour certains dieux (les moins connus) ces informations ne sont pas forcément renseignées, nous affichons

alors un champ spécifiant l'absence de ces informations. Nous affichons également l'arbre généalogique du Dieu, puisque nous avons déjà récupéré les différentes informations nécessaires (ses parents, conjoints et enfants). L'arbre, présenté sous la forme d'un tableau, permet également de naviguer de la page du Dieu actuel vers celle d'un autre Dieu de sa famille simplement en cliquant sur son nom, sans effectuer de nouvelle recherche.

Une auto-complétion permet à l'utilisateur de saisir le nom du Dieu à rechercher sans faute et propose également une liste de dieux à rechercher pour guider l'utilisateur et l'inviter à découvrir des dieux qu'il ne connaît pas nécessairement. De plus, les URI DBPedia étant sensibles à la casse, cela posait des problèmes lorsque l'utilisateur ne rentrait pas le nom avec la première lettre en majuscule et le reste en minuscule. Nous avons donc fait en sorte de traiter la chaîne de caractères entrée par l'utilisateur afin de la rendre sous le format correct. L'utilisateur peut donc taper le nom sans se soucier de la casse.

6. Problèmes rencontrés

De la spécification à la conception, nous avons rencontré des situations complexes nous obligeant à adapter notre solution afin de répondre au mieux au cahier des charges initial.

Nous pouvons séparer la source des problèmes rencontrés en 2 grandes parties: les requêtes SPARQL et l'interface Web.

Nos connaissances limitées sur le langage SPARQL nous ont contraint à passer un temps non négligeable à tester de multiples requêtes dans le but d'affiner nos résultats (de manière manuelle car on ne peut pas s'assurer que les ressources d'Internet sont bonnes). Ainsi nous avons rédigé les requêtes partie par partie, chaque partie étant fonctionnelle individuellement. Cependant lors de la mise en commun de nos solutions, de nouvelles erreurs sont apparues (nous ne pouvions, par exemple, pas faire l'union des requêtes n°4 et 5 cf. Annexes). Nous avons donc séparé les requêtes. Par exemple, lors de la recherche des enfants d'un Dieu, nous avons une requête qui renvoie les objets du prédicat dbp:children du Dieu, et nous avons également une deuxième requête qui recherche tous les sujets qui ont notre Dieu comme objet du prédicat dbp:parent.

Nous avons également observé lors de nos tests que les objets des prédicats dbp:siblings, dbp:children, dbp:parents n'étaient pas constants en type. Nous avions donc, selon les sujets, des chaînes de caractères avec tous les frères et soeurs séparés par des virgules, et d'autres fois des URI renvoyant vers la page du Dieu frère ou soeur. Nous avons donc dû séparer les cas, traiter les chaînes de caractères en séparant les objets grâce aux virgules et traiter les ressources lorsque l'objet n'était pas un littéral.

De plus, le fait que certains dieux soient plus connus que d'autres implique un manque de triplets RDF pour les dieux les moins connus. Nous avons dû rendre optionnels beaucoup de prédicats ainsi que traiter tous les cas où les triplets n'étaient pas renseignés de la même manière (dbp:godOf de Gaia par exemple qui était une ressource).

Concernant l'interface Web, nous nous sommes confrontés à une incertitude quant à la production finale à fournir. Tout d'abord les ressources censées nous aider à construire le projet sur moodle n'étaient plus d'actualité. Nous avons pensé qu'il fallait procéder à du web scrapping sur tout le Web à l'aide d'API alors qu'il s'agissait d'interroger uniquement DBpedia. Notre mauvaise compréhension de la consigne nous a fait perdre un temps non négligeable.

Nous étions donc d'abord parti sur une architecture client-serveur avec un back-end utilisant des API permettant de construire un vrai moteur de recherche Google et appelant des script shell exécutant des requêtes SPARQL alors que cela n'était pas demandé. De plus, nous avons eu une autre difficulté concernant la manière d'exécuter les requêtes SPARQL : nous avons dû tester 2 ou 3 API DBpedia pour exécuter nos requêtes mais elles ne marchaient pas comme nous voulions, avant de nous rendre compte qu'on pouvait construire une "API maison" avec comme endpoint la page de résultat de requêtes de Virtuoso.

Une autre difficulté rencontrée par la suite a été de recréer l'arbre généalogique avec un affichage adapté à chaque dieu. En effet, dans la mythologie grecque, les dieux peuvent avoir plusieurs épouses, des dizaines d'enfants, certains sont également mariés à leurs frères et soeurs, et d'autres problèmes similaires. Parvenir à un affichage satisfaisant a donc été particulièrement complexe, et nous avons finalement opté pour un tableau synthétisant ces informations.

7. Conclusion

En ce qui concerne le Web Sémantique de manière générale, nous trouvons que le champ des usages proposé par celui-ci est réellement intéressant, et son utilité n'est plus à démontrer. Cependant, durant le projet, nous avons pu nous rendre compte d'un bon nombre de problèmes qui sont liés à l'un des principes fondamentaux du web sémantique qui est que "Anyone can say Anything about Anything" (AAA). Ce principe permet d'une part de faciliter la contribution de n'importe quelle personne à l'enrichissement des bases de connaissances RDF du web sémantique, mais d'autre part ne donne aucune certitude quant à la fiabilité des données présentes sur ces bases de données. Le seul levier de l'utilisateur du web sémantique reste alors la robustesse et le fiabilité de ses requêtes pour produire les résultats les plus fiables possibles.

Dans l'optique de faciliter l'identification des données fiables, on pourrait imaginer un système collaboratif dans lequel l'ensemble des utilisateurs pourrait attester de la fiabilité ou non de chacun des triplets RDF. Ainsi, pour chaque triplet, il y aurait une sorte de note qualifiant sa fiabilité, et les utilisateurs pourront ainsi prendre en compte dans leurs requêtes les triplets ayant été jugés comme fiables. Ce système pourrait aussi inciter les utilisateurs ajoutant des données aux bases de connaissances RDF à mieux vérifier la correctitude des données avant de les insérer.

Annexes

Requête n°1 (cf var generalQuery dans script.js) : Récupère toutes les informations du Dieu qui ne sont pas des listes (son nom, l'URL de son image, son lieu de résidence et son genre). Les objets image, abode et gender sont optionnels. Si le prédicat dbp:gender n'est pas renseigné dans la ressource, nous regardons si le dieu est "God of" ou "Goddess of" pour déterminer son sexe.

```
select ?uri as ?resource, STR(?n) as ?nameOfGod, ?image, STR(?go) as
?GodOf,STR(?abode) as ?Abode, if(EXISTS{?uri foaf:gender ?ge},
STR(?ge),(if(regex(?GodOf,".*God .*","i"),
"Male",(if(regex(?GodOf,".*Goddess .*","i"), "Female","Not
specified"))))) as ?Gender where {

   #Get only the Greek gods
   ?uri dbp:name ?n;
   dbp:type ?t.
   #We filter first in order to reduce the number of matches
    Filter(regex(?t,".*Greek.*") and datatype(?n)=rdf:langString and
regex(?n, ".*Cronus(_| |$)"))
   #God of ?go :
   {
    ?uri dbp:godOf ?go.
    FILTER(isLiteral(?go))
   }
   UNION
   {
    ?uri dbp:godOf ?goresource.
    ?goresource rdfs:label ?go.
    FILTER(isLiteral(?go) and lang(?go)="en")
   }
   #Get the image of the god
   optional{?uri dbo:thumbnail ?image}.

   #Abode of the god
   optional{
      ?uri dbp:abode ?a.
      ?a rdfs:label ?abode.
      FILTER(lang(?abode)="en")
   }.
  #Gender of the god
   optional{?uri foaf:gender ?ge}.
```

```
}
```

Requête n°2 (cf var siblingsQuery dans script.js) : Cette requête récupère le nom des frères et soeurs du Dieu. Nous avons identifié 2 cas possibles, celui où l'objet ?siblings est un string composé des frères et soeurs séparés par des virgules, et le cas où ?siblings est un groupement de ressources. Nous traitons les 2 cas et nous renvoyons uniquement les noms des frères et soeurs.

```
SELECT DISTINCT STR(?sibling) as ?Sibling WHERE
{
  ?uri dbp:name ?n;
  dbp:godOf ?go;
  dbp:type ?t.
  #We filter first in order to reduce the number of matches
  FILTER(regex(?t,".*Greek.*") and regex(?n,".*Ares( |$)","i"))

  {
   #Get the siblings if the dbp:sibling ?object is a string composed
of the siblings (we split the string at each comma)
  VALUES ?N { 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30} #can split the siblings string into 20
different siblings
  ?uri dbp:siblings ?siblings.
  FILTER(!isBlank(?siblings) and isLiteral(?siblings))
  BIND(replace(?siblings, " and ", " ") as ?sibStr)
  BIND (concat("^([^,]*,){", str(?N) ,"} *") AS ?skipN)
    BIND  (replace(replace(?sibStr,  ?skipN,  ""),  ",.*$",  "")  AS
?sibling)
  }
  UNION
  {
     #Get the siblings if the dbp:siblings ?object is a resource
     {?uri dbp:siblings ?siblings.}
     UNION
     {?siblings dbp:siblings ?uri.}

     ?siblings dbp:godOf ?go2;
     dbp:type ?t;
     dbp:name ?sibling.
```

```
                                    Filter(isLiteral(?sibling)      and
datatype(?sibling)=rdf:langString)
   }
}
```

Requête n°3 (cf var symbolsQuery dans script.js) : Cette requête retourne le nom des symboles du Dieu. Nous séparons les cas où l'objet ?symbols est un littéral et le cas où l'objet est une ressource.

```
select DISTINCT STR(?symbol) as ?Symbol where {
   ?uri dbp:name ?n;
   dbp:godOf ?go;
   dbp:type ?t.
   Filter(regex(?t,".*Greek.*") and regex(?n,".*Apollonis( |$)","i"))
     {
        #?symbols is a resource
        ?uri dbp:symbol ?symbols.
        ?symbols rdfs:label ?symbol.
        FILTER(!isLiteral(?symbols) and lang(?symbol)="en")
     }
     UNION
     {
        #?symbol is literal
        ?uri dbp:symbol ?symbol.
        FILTER(isLiteral(?symbol))
     }
}
```

Requête n°4 (cf var childrenQuery dans script.js) : Cette requête retourne le nom des enfants du Dieu (qu'ils soient des dieux ou non). A l'instar de la requête n°2, nous séparons les cas où l'objet est un littéral et les cas où l'objet est un groupement de ressources.

```
SELECT DISTINCT STR(?child) as ?Children
WHERE {
  ?uri dbp:name ?n;
  dbp:godOf ?go;
  dbp:type ?t.
  FILTER(regex(?t,".*Greek.*") and regex(?n,".*Zeus( |$)","i"))
```

```
  {
     #the object of the dbp:children predicate is a literal
      VALUES ?N { 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30} #can split the children string into 30
different children
     ?uri dbp:children ?childrenStr.
     FILTER(!isBlank(?childrenStr)  and isLiteral(?childrenStr))
     BIND(replace(?childrenStr, " and ", " ") as ?childStr)
     BIND (concat("^([^,]*,){", str(?N) ,"} *") AS ?skipN)
       BIND (replace(replace(?childStr, ?skipN, ""), ",.*$", "") AS
?child)
  }
  UNION
  {
     #the object of the dbp:children predicate is a resource
     {?uri dbp:children ?children.}
     UNION
     {?children dbp:parents ?uri.}

     ?children rdfs:label ?child.

     Filter(isLiteral(?child) and lang(?child)="en")
  }
}
```

Requête n°5 (cf var childrenQuery2 dans script.js) : Cette requête vient en complément de la requête n°4. Elle retourne le nom des enfants d'un Dieu mais en s'intéressant cette fois-ci au prédicat dbp:parent. Nous récupérons donc les enfants en regardant si le Dieu en question fait partie de leurs parents. Toujours en séparant les cas où l'objet est un littéral ou une ressource.

```
SELECT DISTINCT STR(?child) as ?Children
WHERE {
   {
     ?uri dbp:name ?n;
     dbp:godOf ?go;
     dbp:type ?t3.
     FILTER(regex(?t3,".*Greek.*") and regex(?n,".*Zeus( |$)","i"))
     ?childRes dbp:parents ?parents;
     dbp:type ?t3.
     FILTER(!isBlank(?parents) and isLiteral(?parents))
```

```
       VALUES ?N1 { 1 2 3 4} #can split the parent string into 4
different parents
     BIND(replace(?parents, " and ", " ") as ?parentStr)
     BIND (concat("^([^,]*,){", str(?N1) ,"} *") AS ?skipN1)
      BIND (replace(replace(?parentStr, ?skipN1, ""), ",.*$", "") AS
?parent)
     #Get the parent's name that match our God
     FILTER(regex(?parent, ?n))
     ?childRes dbp:name ?child.
     FILTER( datatype(?child)=rdf:langString)
  }
  UNION
  {
     ?childRes dbp:parents ?parents;
     dbp:type ?t3.
     ?parents dbp:type ?t3;
     dbp:name ?parentName.
     FILTER(regex(?parentName, ".*Zeus( |$)", "i"))
     ?childRes dbp:name ?child.
     FILTER( datatype(?child)=rdf:langString)
  }
}
```

Requête n°6 (cf var parentsQuery dans script.js) : Cette requête récupère le nom des parents du Dieu (qu'ils soient eux-mêmes des Dieux ou pas). Nous séparons les cas où l'objet du prédicat dbp:parent est un string ou une ressource. Cet objet étant globalement bien renseigné, nous n'avons pas trouvé nécessaire de rajouter une requête complémentaire pour trouver les sujets ayant notre Dieu comme enfant (cela faussait les résultats au lieu de les compléter).

```
select DISTINCT ?parent where
{
   ?uri dbp:name ?child;
   dbp:godOf ?go;
   dbp:type ?t.
   Filter(regex(?t,".*Greek.*") and regex(?child,".*Zeus( |$)","i"))
   {
     ?uri dbp:parents ?parents.
     FILTER(!isBlank(?parents) and isLiteral(?parents))
       VALUES ?N1 { 1 2 3 4} #can split the parent string into 4
different parents
```

```
     BIND(replace(?parents, " and ", ",") as ?parentStr)
     BIND (concat("^([^,]*,){", str(?N1) ,"} *") AS ?skipN1)
      BIND (replace(replace(?parentStr, ?skipN1, ""), ",.*$", "") AS
?parent)
   }
   UNION
   {
      ?uri dbp:parents ?parents.
      ?parents dbp:name ?parent.
      FILTER(isLiteral(?parent))
   }
}
```

Requête n°7 (cf var consortsQuery dans script.js) : Cette requête récupère le nom des conjoint(e)s du dieu. Nous séparons toujours les cas où l'objet du prédicat dbp:consort est un littéral ou une ressource.

```
select DISTINCT STR(?consort) as ?Consort where {
   ?uri dbp:name ?n;
   dbp:godOf ?go;
   dbp:type ?t.
   Filter(regex(?t,".*Greek.*") and regex(?n,".*Zeus( |$)"))

  {
    VALUES ?N { 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20}
#can split the consort string into 20 different consorts
    ?uri dbp:consort ?consorts.
    FILTER(!isBlank(?consorts) and isLiteral(?consorts))
    BIND(replace(?consorts, " and ", ",") as ?consortStr)
    BIND (concat("^([^,]*,){", str(?N) ,"} *") AS ?skipN)
     BIND (replace(replace(?consortStr, ?skipN, ""), ",.*$", "") AS
?consort)
  }
  UNION
  {
    ?uri dbp:consort ?consorts.
    ?consorts dbp:type ?t;
    dbp:name ?consort.
    FILTER(datatype(?consort)=rdf:langString)
  }
}
```

Requête n°8 (cf var consortsQuery2 dans script.js) : A l'image de la requête n°5, cette requête vient en complément de la requête n°7 en retournant le nom des sujets ayant pour objet du prédicat dbp:consort le Dieu recherché.

```
SELECT DISTINCT STR(?consort) as ?Consorts
WHERE {
  {
     ?uri dbp:name ?n;
     dbp:godOf ?go;
     dbp:type ?t.
     FILTER(regex(?t,".*Greek.*") and regex(?n,".*Hera( |$)","i"))

     ?consortRes dbp:consort ?ourGod;
     dbp:type ?t.
     FILTER(!isBlank(?ourGod) and isLiteral(?ourGod))

      VALUES ?N1 { 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20}
#can split the consort string into 20 different parents
     BIND(replace(?ourGod, " and ", " ") as ?ourGodStr)
     BIND (concat("^([^,]*,){", str(?N1) ,"} *") AS ?skipN1)
      BIND (replace(replace(?ourGodStr, ?skipN1, ""), ",.*$", "") AS
?ourGodName)
     FILTER(regex(?ourGodName, CONCAT(?n,"( |$)"), "i"))
     ?consortRes dbp:name ?consort.
  }
  UNION
  {
     ?uri dbp:name ?n;
     dbp:godOf ?go;
     dbp:type ?t.
     FILTER(regex(?t,".*Greek.*") and regex(?n,".*Hera( |$)","i"))
     ?consortRes dbp:consort ?ourGod;
     dbp:type ?t.
     ?ourGod dbp:type ?t;
     dbp:name ?ourGodName.
     FILTER(regex(?ourGodName, CONCAT(?n,"( |$)"), "i"))
     ?consortRes dbp:name ?consort.
  }
}
```

Requête n°9 (cf var gamesQuery dans script.js) : Cette requête permet de récupérer les jeux vidéos dans lesquels il y a une référence au Dieu recherché. On regarde donc les jeux qui possèdent une ou plusieurs occurrences du nom du Dieu recherché dans leur abstract et/ou commentaire. La difficulté de cette requête est de pouvoir faire la différence entre le cas où le nom du Dieu retrouvé fait bien référence au Dieu grec, et le cas où il fait référence à un autre personnage n'ayant pas de lien avec la mythologie grecque. Pour cela nous regardons donc si le jeu a des liens avec la mythologie grecque ou pas.

```
select Distinct(STR(?label)) as ?game where{
      ?uri rdf:type dbo:VideoGame;
           dbo:abstract ?abstract;
           rdfs:label ?label;
           dct:subject ?subject;
           rdf:type ?type.
        Filter(( lang(?label)="en" and lang(?abstract)="en" ) and (
regex(?abstract,"`+godNamewithGoodCaps+`(      |,|;|\\\\.)")      and
!regex(?abstract,"Zeus Software") and   (regex(?type,"mytholog","i")
||                      regex(?abstract,"mytholog","i")           ||
regex(?subject,"mytholog","i")      ||      regex(?abstract,"   god(
|,|;|.//)","i")  )  ))
}
```

Requête n°10 (cf var moviesQuery dans script.js) : Cette requête permet de récupérer les films dans lesquels il y a une référence au Dieu recherché. On regarde donc les films qui possèdent une ou plusieurs occurrences du nom du Dieu recherché dans leur "abstract". Comme pour la requête précédente, la difficulté est de pouvoir faire la différence entre le cas où le nom du Dieu retrouvé fait bien référence au Dieu grec, et le cas où il fait référence à un autre personnage n'ayant pas de lien avec la mythologie grecque. Pour cela nous regardons donc si le film a des liens avec la mythologie grecque ou pas.

```
select Distinct(STR(?label)) as ?movie , ?uri where{
      ?uri rdf:type dbo:Film;
           dbo:abstract ?abstract;
           rdfs:label ?label;
           rdf:type ?type.
        Filter(( lang(?label)="en" and lang(?abstract)="en" ) and (
regex(?abstract,"Zeus( |,|;|\\.)")  and  (regex(?type,"mytholog","i")
||   regex(?abstract,"mytholog","i")     ||   regex(?abstract,"  god(
|,|;|.//)","i")  )  ))
}
```

Requête n°10 (cf var artQuery dans script.js) : Cette requête permet de récupérer les oeuvres d'art dans lesquels il y a une référence au Dieu recherché. On regarde donc les films qui

possèdent une ou plusieurs occurrences du nom du Dieu recherché dans leur “abstract”. Comme pour les requêtes précédente, la difficulté est de pouvoir faire la différence entre le cas où le nom du Dieu retrouvé fait bien référence au Dieu grec, et le cas où il fait référence à un autre personnage n’ayant pas de lien avec la mythologie grecque. Pour cela nous regardons donc si le film a des liens avec la mythologie grecque ou pas.

```
select Distinct(STR(?label)) as ?artPiece, ?uri where{
      ?uri rdf:type dbo:Artwork;
           dbo:abstract ?abstract;
           rdfs:label ?label;
           dct:subject ?subject;
           rdf:type ?type.
        Filter(( lang(?label)="en" and lang(?abstract)="en" ) and (
regex(?abstract,"Zeus( |,|;|\\\\.)") and (regex(?type,"mytholog","i")
||                      regex(?abstract,"mytholog","i")           ||
regex(?subject,"mytholog","i")      ||      regex(?abstract,"   god(
|,|;|.//)","i")  )  ))
}
```

Requête n°11 (cf var autoCompleteQuery dans autoComplete.js) : Récupère le nom de tous les dieux grecs.

```
select DISTINCT STR(?n)  where {
                ?uri dbp:name ?n;
                dbp:godOf ?go;
                dbp:type ?t.
                  Filter(regex(?t,".*Greek.*") and isLiteral(?n) and
datatype(?n) = rdf:langString)
}
```