UniversMaps

Gregory Arnal

Sommaire

Présentation

Passionnées d'astronomie en quête de savoir ne cherchez plus, UniversMaps est la pour vous. Sur une seule carte retrouvez des informations sur toutes les planètes du systèmes solaire.

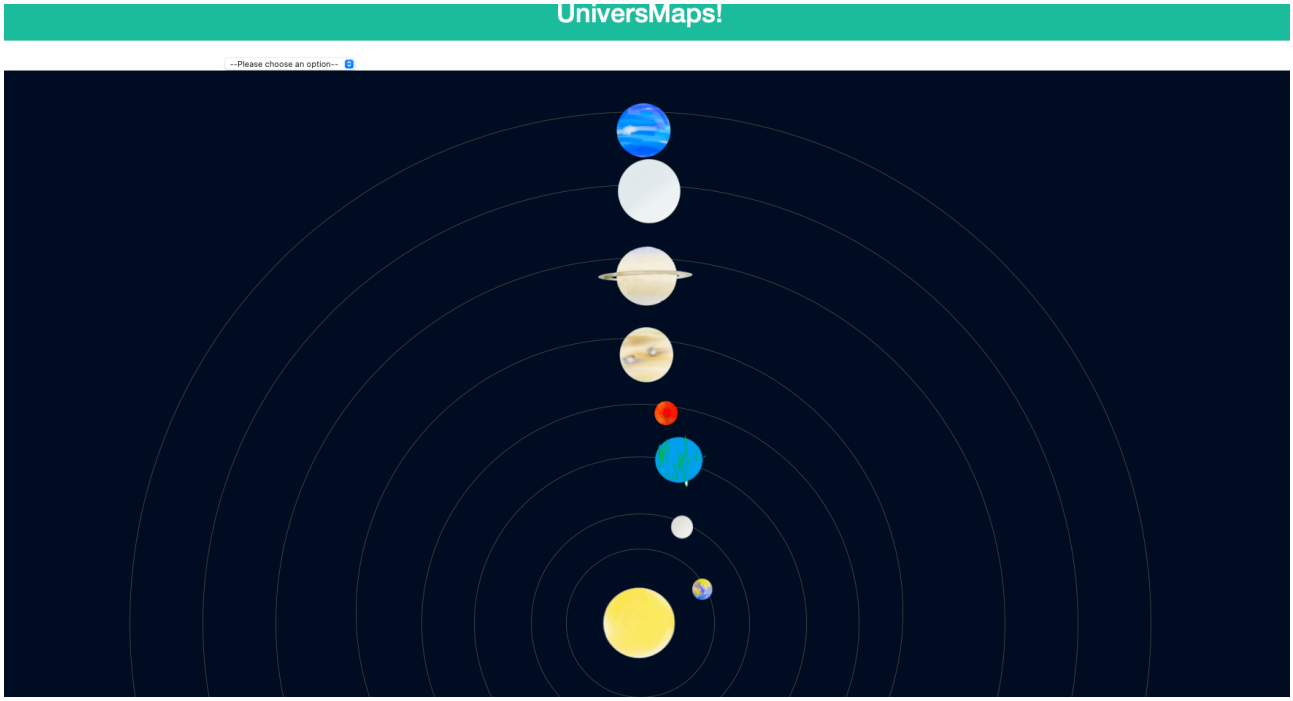
Pour chaque astres vous pouvez trouver les informations suivantes :

- Une description
- La vitesse de rotation moyenne
- La masse volumique
- Les objets orbitant autour
- Le perimetre
- L'aire de la planète
- La masse
- La distance depuis la terre
- La vitesse de libération
- La date de découverte
- Le nom de la personne l'ayant découvert
- Le nombres de satellites
- La période de rotation autour du soleil

Pour chaque astres les données retournées dépendent des données trouvées en ligne, certaines planètes possèdent moins d'information.

Pour la réalisation de ce projet j'ai utilisé les données disponible sur Wikidata et DbPedia. Pour assurer l'intégrité des données affichées elles ne sont pas enregistrées en cas de modification de la base.

Il s'agit principalement d'un projet personnelle, en effet je n'ai jusqu'à present pas trouvé d'outils permettant de regrouper de manière interactif et « user friendly » des données sur les astres du système solaire et autres.
Ce projet sera par la suite mis à jour avec une carte du système solaire en 3D dans laquelle l'utilisateur pourra se déplacer. L'experience sera également prolongé au delà du système solaire. Le but étant de donner envie au personne de s'intéresser à ce domaine passionnant sans devoir passer des heures à chercher des informations sur des sites obscures.



L’expérience utilisateur

L’expérience utilisateur est relativement simple, mais efficace.

Sur la page d’accueil nous seront face à une carte du système solaire avec les 8 planètes en rotation autour du soleil.

Une liste sur la gauche peut permettre de choisir des astres se trouvant en dehors du système solaire.

Il suffit alors de cliquer sur une planète ou de sélectionner un objet dans la liste pour afficher une popup qui contiendra les données trouvées en ligne.

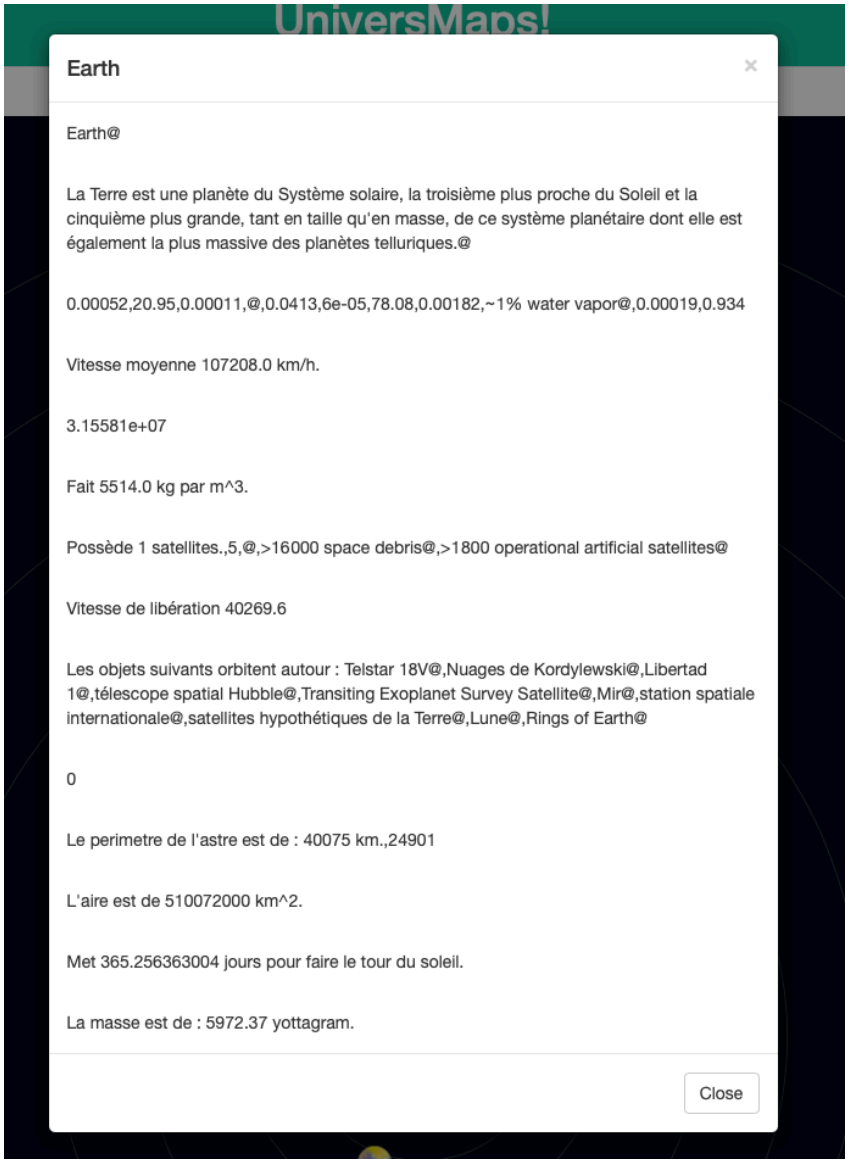
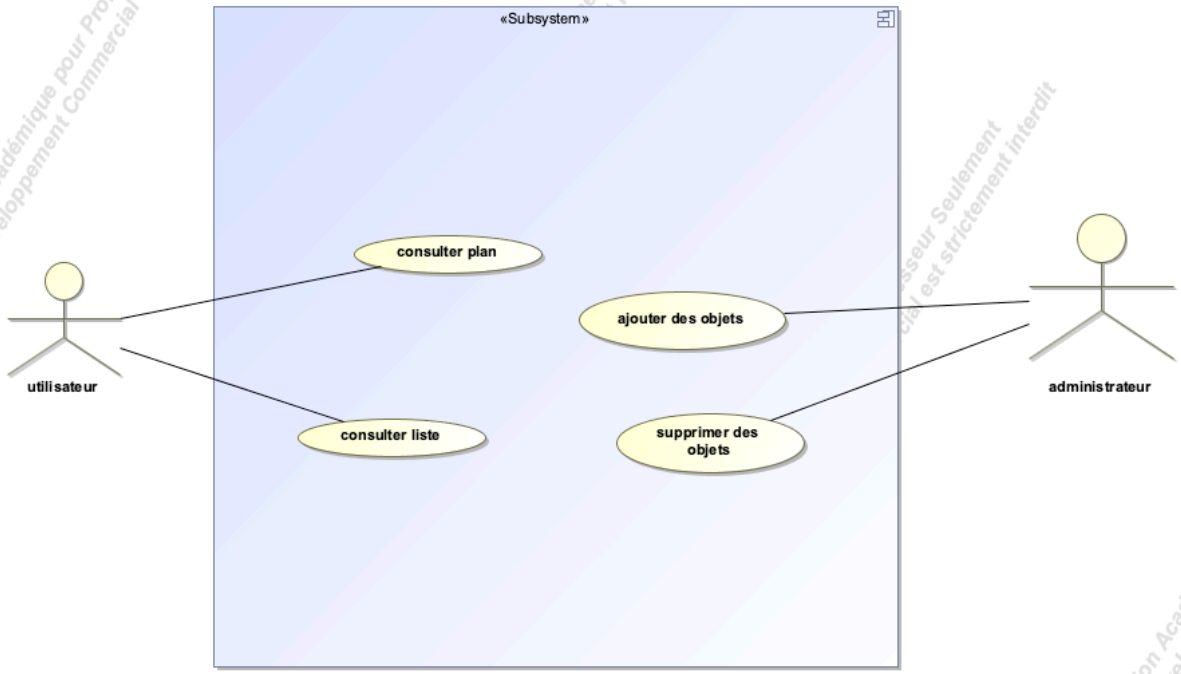


Diagramme UWE

Diagramme cas d'utilisation

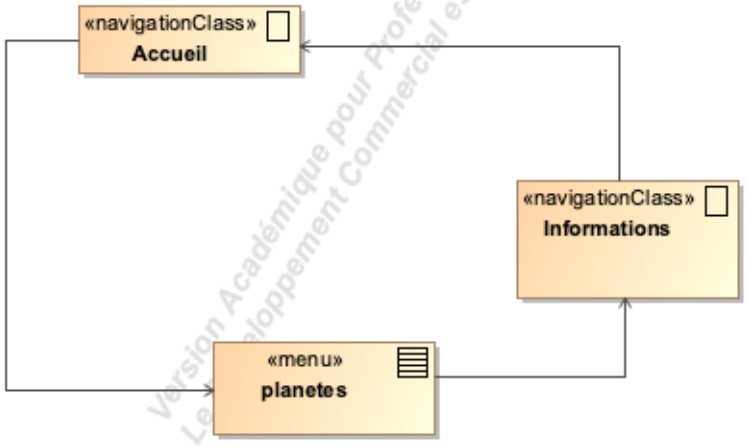


Le diagramme de cas d'utilisation est relativement simple, en effet l'utilisateur ne peut que consulter la carte ou la liste des astres pour avoir les informations. De plus l'administrateur du site peut ajouter ou supprimer des astres via un fichier de configuration :

```
planet = Sun,Mercury,Venus,Earth,Mars,Jupiter,Saturn,Uranus,Neptune
```

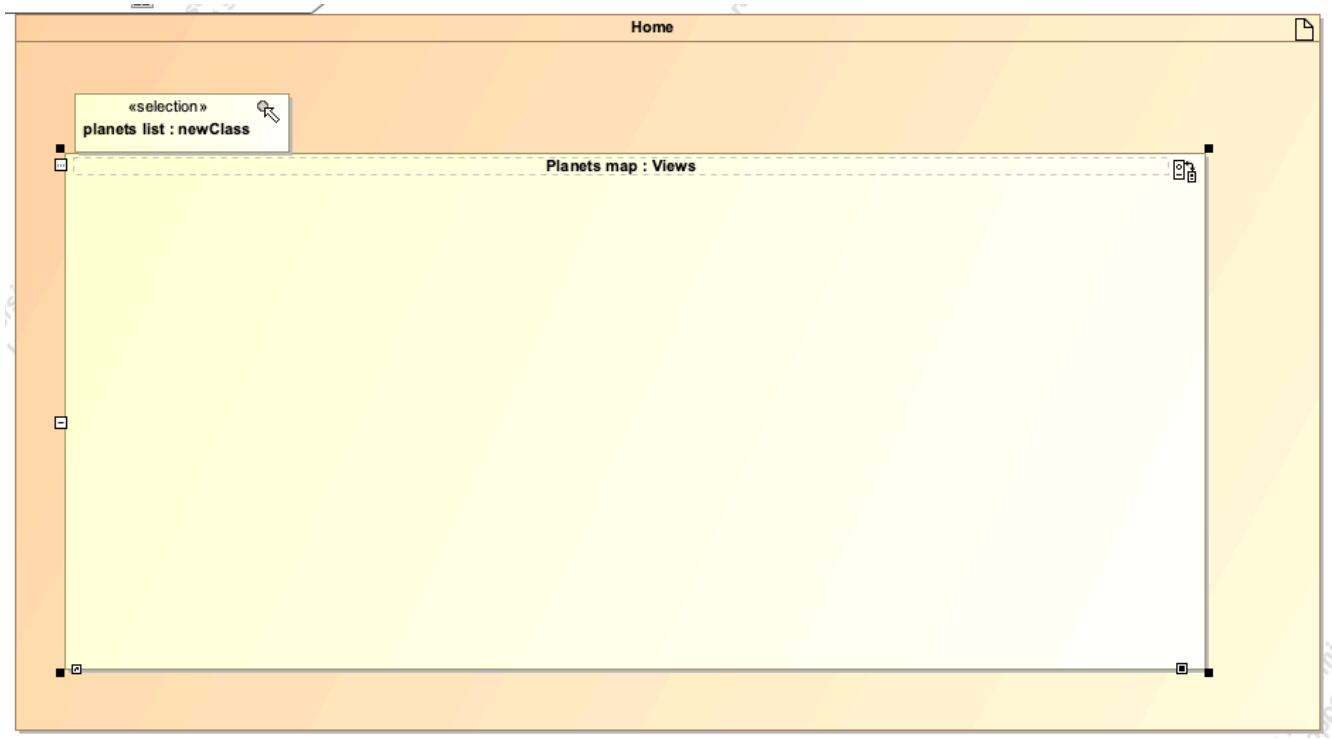
Dans le fichier config.properties nous pouvons déclarer une liste de planètes. En association il faut mettre en place une image de la planète et ajouter la partie CSS avec la dimension de l'orbite et une vitesse de rotation. Cela ajoutera automatiquement les planètes sur l'interface.

Diagramme de navigation



Le diagramme de navigation reprend les pages visibles par l'utilisateur depuis la page d'accueil l'utilisateur aura accès à un « menu » avec les planètes la sélection de l'une d'entre elle permettra la recherche et l'affichage des données.

Diagramme de presentation



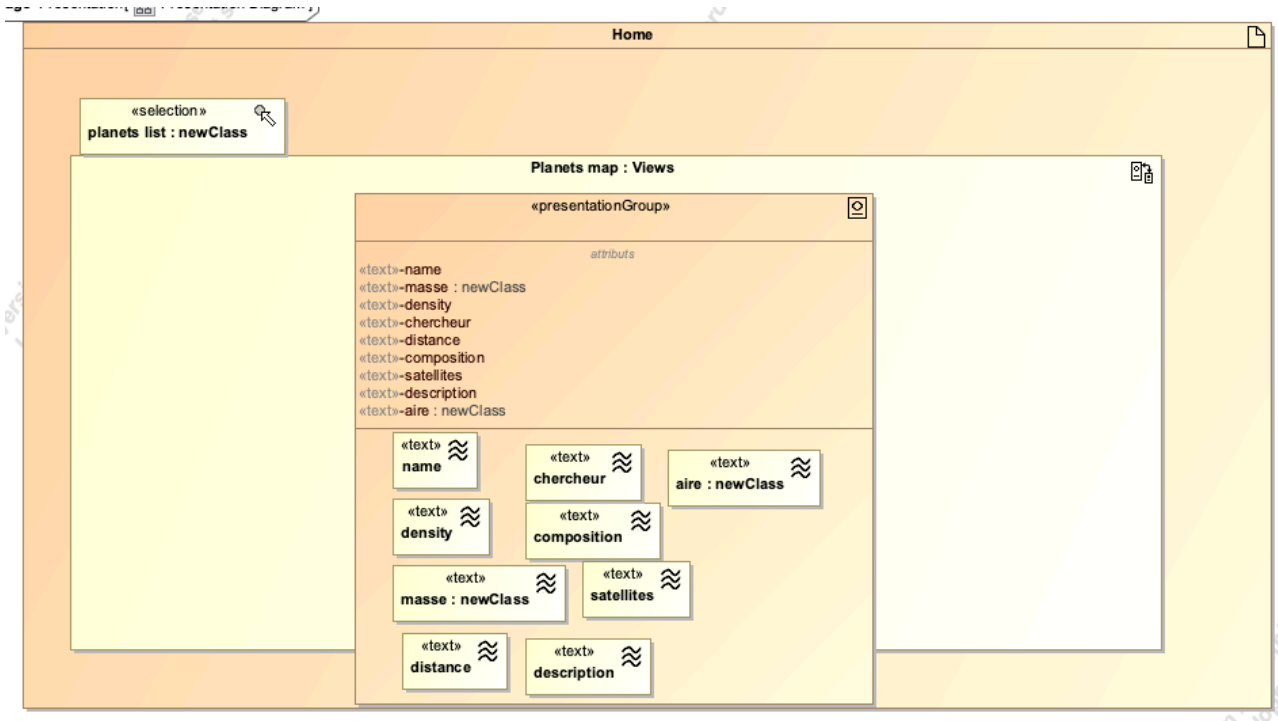
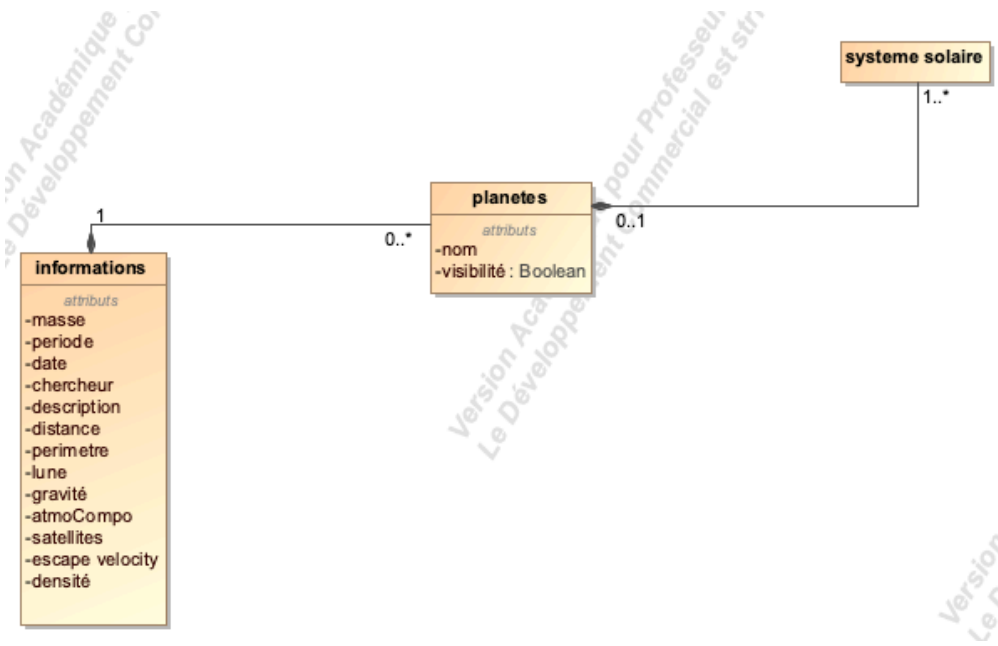
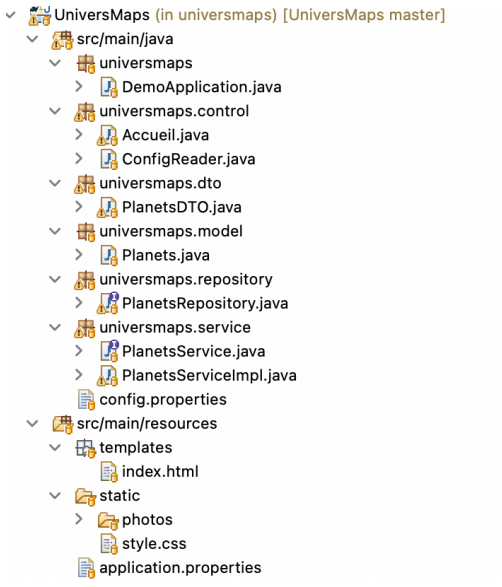


Diagramme de contenu



Description technique

Pour la réalisation de ce projet j'ai utilisé le langage java, avec la librairie Jena qui me permet de faire les requêtes sparql sur Dbpedia et WikiData. Pour la création de l'application web j'utilise le framework Spring et pour finir pour le serveur web j'utilise TomCat.



Récupération des données

Une classe va venir faire la passage des informations entre l'interface graphique et le service de gestion. Cette classe est : Accueil, on va trouver dedans une méthode accueil qui sera appelé par default lors du chargement de la page et une méthode dataLoad qui sera appelé à l'affichage de la popup.

Le service permettant de récupérer les données s'appelle planetsservice.

Lors du chargement de la page nous allons si cela n'a pas déjà était fait initialiser les planètes. Pour cela la méthode instantiate va récupérer les planètes indiquées dans le fichier de configurations et initialiser les objets équivalent.

Suite à ça nous ajoutons les données voulu au model selon différents identifiants. Cela servira à mettre en place la carte et à remplir la liste déroulante.

Comme indiqué la méthode dataLoad sera lancé par une requête get sur l'url :

Popup/{name}

- {name} représentant le nom de la planète, par défaut cette parti de l'url sera mise dans une variable.

Nous utilisons ensuite la méthode SearchData(name) de planetsservice qui nous retourne un JSONObject contenant les données. Les données sont ensuite envoyé à la partie AJAX via un ResponseEntity. Cela me permet de ne pas recharger la page, les mises à jours sont faites par la partie Ajax. Si aucune données n'est trouvées cela sera mentionné.

La méthode dataLoad est appelée a ce moment :

```
function makeRequest(name) {

  httpRequest = new XMLHttpRequest();
  if (!httpRequest) {
    alert('Abandon :( Impossible de créer une instance de XMLHTTP');
    return false;
  }
  planet = name;

  httpRequest.onreadystatechange = alertContents;
  httpRequest.open('GET', '/popup/'+ name);
  httpRequest.send();

}
```

Lors d'un changement d'état de httpRequest cela lancera la method alertContents qui lira et affichera les données.

```
function alertContents() {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        if (httpRequest.status === 200) {
          var data=httpRequest.responseText;
          var jsonResponse = JSON.parse(data);
          var len = Object.keys(jsonResponse).length;
          var keys = Object.keys(jsonResponse);
          if (len == 0){document.getElementById("data"+planet+"load").innerHTML = "Pas de données...";}
          else{document.getElementById("data"+planet+"load").style.display = 'none';}
          for (var i = 0; i < len-1; i++)
          {
              var name = keys[i];
              document.getElementById("data"+planet+""+name).innerHTML =  jsonResponse[name];
              document.getElementById("data"+planet+""+name).style.display = 'block';
          }
          //document.getElementById("data"+planet+).innerHTML = jsonResponse["abstractd"];
          httpRequest = -1
        } else {
          alert('Il y a eu un problème avec la requête.');
        }
      }

}
```

Ici si la valeur du statut de la requête vaut 200 cela veut dire que nous avons reçu une réponse sans erreur, dans le cas contraire cela affichera une alerte à l'écran. Dans le cas ou tout est bon la données récupérées est alors mis au format json. Si le nombre de clé vaut 0 « pas de données… » sera affiché, sinon cela placera la données dans la div associée. Une fois fini le httprequest est mis à la valeur -1 pour « effacer » les données précédemment reçu.

Requête Sparql

Type de données

Il y'a en tout 20 données pouvant être récupérés.
Les données sont récupérés sur Wikidata et sur dbpedia via les requêtes suivantes :

```
SELECT  distinct ?childLabel ?date ?dist ?perimeter ?area ?
diameter ?period ?mass ?discovererLabel where {

        ?planet ?planetLabel '"+ name +"' .
        ?planet wdt:P361 ?Q7879772 .
        ?s dbp:name ?name .
        OPTIONAL { ?planet wdt:P398 ?child . }
        OPTIONAL { ?planet wdt:P2583 ?dist . }
        OPTIONAL { ?planet wdt:P2046 ?area . }
        OPTIONAL { ?planet wdt:P2547 ?perimeter . }
        OPTIONAL { ?planet wdt:P2286 ?diameter . }
        OPTIONAL { ?planet wdt:P2146 ?period . }
        OPTIONAL { ?planet wdt:P575 ?date . }
        OPTIONAL { ?planet wdt:P61 ?discoverer . }
        OPTIONAL { ?planet wdt:P2067 ?mass . }
        SERVICE wikibase:label {
                        bd:serviceParam wikibase:language
                "[AUTO_LANGUAGE],fr,ar,be,bg,bn,ca,cs,da,de,el,en,
                es,et,fa,fi,he,hi,hu,hy,id,it,ja,jv,ko,nb,nl,eo,pa,pl,pt,ro,r
                u,sh,sk,sr,sv,sw,te,th,tr,uk,yue,vec,vi,zh"
    }
}
```

WikiData

```
Select *
where{
        OPTIONAL {?s dbo:density ?density . }
        OPTIONAL {?s dbo:averageSpeed ?averageSpeed . }
        ?s dbp:name ?name .
        OPTIONAL {?s dbp:satellites ?satellites . }
        OPTIONAL {?s dbp:surfaceGrav ?surfaceGrav . }
        OPTIONAL {?s dbp:surfacePressure ?surfacePressure . }
        OPTIONAL {?s dbo:apoapsis ?apoapsis . }
        OPTIONAL {?s dbo:escapeVelocity ?escapeVelocity . }
        OPTIONAL {?s dbo:formerName ?formerName . }
        OPTIONAL {?s dbo:orbitalPeriod ?orbitalPeriod . }
        OPTIONAL {?s dbo:periapsis ?periapsis . }
        OPTIONAL {?s dbo:abstract ?abstract . }
        filter not exists{ filter contains(str(?abstractd), 'Erde') . }
         filter langMatches(lang(?abstract), 'fr') .
         filter strends(str(?s), '"+ name +"') .
         filter (str(?name) = '"+ uriname +"')
}
```

Dbpédia

Les requêtes sont réalisées grace à Jena, les données récupérées sont alors ajoutées à un json.

Ici je trie une partie en ne prenant pas en compte la donnée de la clé « s » car elles ne sont pas nécessaire pour l'affichage, je les garde dans la requêtes pour des raisons de contrôle en cas de soucis avec la cohérence des données. De plus lors que je traite le la données relative a une image je ne split pas mon résultat, qui en temps normal est split pour enlever le type de la données.

```
Query query = QueryFactory.create(szQuery);

// Create the Execution Factory using the given Endpoint
QueryExecution qexec = QueryExecutionFactory.sparqlService(szEndpoint, query);


// Set Timeout
((QueryEngineHTTP) qexec).addParam("timeout", "10000");

// Execute Query
int iCount = 0;
ResultSet rs = qexec.execSelect();
//System.out.println("get result ");
while (rs.hasNext()) {
    // Get Result
    QuerySolution qs = rs.next ();

    // Get Variable Names
    Iterator<String> itVars = qs.varNames();

    // Count
    iCount++;
    //System.out.println("Result " + iCount + ": ");

    // Display Result
    while (itVars.hasNext()) {
        String szVar = itVars.next().toString();
        String szVal = qs.get(szVar).toString();

        if (!szVar.equals("s")) {
            String val;
            if(szVar.equals("img")) {
                val = szVal;
            }else {
                val = szVal.split("http")[0];
                val = val.substring(0,val.length()-2);
            }

            String[] data;
            if (json.has(szVar)) {
                try {
                    data = (String[]) json.get(szVar);
                }catch(Exception e){
                    data = new String[1];
                    data[0]= (String) json.get(szVar);
                }

                if (!Arrays.asList(data).contains(val)) {
                    String[] result = new String[data.length + 1];
                     for (int i = 0; i <= data.length -1 ;i++) {
                         result[i] = data[i];
                     }

                     result[result.length-1] = val;
                     json.put(szVar, result);
                }

            }else {
                json.put(szVar, val);
```

En cas de valeurs multiples pour une meme clé on récupère les valeurs déjà existantes et on les mets dans un tableau puis on remet le tableau dans le json.

```
if (json.has(szVar)) {
    try {
        data = (String[]) json.get(szVar);
    }catch(Exception e){
        data = new String[1];
        data[0]= (String) json.get(szVar);
    }

    if (!Arrays.asList(data).contains(val)) {
        String[] result = new String[data.length + 1];
         for (int i = 0; i <= data.length -1 ;i++) {
             result[i] = data[i];
         }

         result[result.length-1] = val;
         json.put(szVar, result);
    }

}else {
    json.put(szVar, val);

}
```

De ce fait toutes les valeurs sont sauvegardés. Une fois fini on renvoi l'objet json qui sera ensuite mis en forme avec du texte et un trie sera effectué sur certaines valeurs multiples

---

Choix techniques

Comme cela est visible dans les requêtes sparql il y'a un filter assez particulier, en effet pour une obscure raison le abstract de la Terre est un champs multiples avec une rapide description de Erde une commune de Conthey. J'ai donc supprimé cette valeur avec le filter.

Pour la réalisation des requêtes lorsque je veux afficher les plantes du système solaire je sélectionne les données eu ayant comme contrainte que l'objet fasse partie du système. Tandis que pour la liste de planète je sélectionne des objets appartenant à une entité planète.

Préparation des données

Une fois les données récupérées la méthode cleanData va venir lire les données dans le json et selon le type de la données mettre en forme la valeur dans une phrase pour l'affichage.

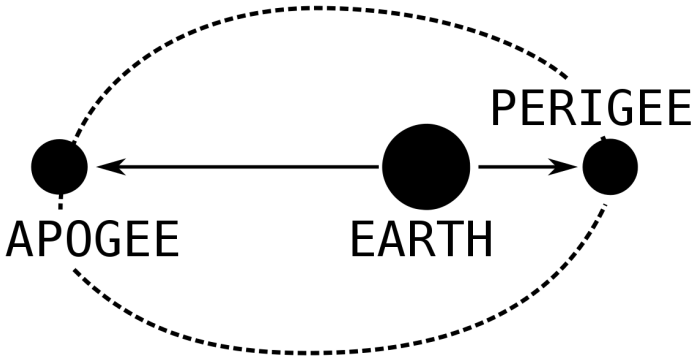
```
public void cleanData(JSONObject json) throws JSONException {
    Iterator keys = json.keys();
    while(keys.hasNext()) {
        String key = (String) keys.next();
        String[] data;
        try {
            data = (String[]) json.get(key);
        }catch(Exception e){
            data = new String[1];
            data[0]= (String) json.get(key);
        }
        if (key.equals("dist") && data.length >= 2) {
            int a;
            int b;
            if(Integer.parseInt(data[0])>Integer.parseInt(data[1])) {
                a = 0;
                b = 1;
            }else {
                a = 1;
                b = 0;
            }
            data[a]= "Distance de la terre à l'apogée : " + data[a] +" km.\n";
            data[b]= "Distance de la terre au périgée : " + data[b] +" km.";

        }else if (key.equals("mass")) {
            data[0] = "La masse est de : " + data[0] + " yottagram.";
        }else if (key.equals("satellites")) {
            data[0] = "Possède " + data[0] + " satellites.";
        }else if (key.equals("volume")) {
            data[0] = "Fait " + data[0] + " m^3";
        }else if (key.equals("averageSpeed")) {
            data[0] = "Vitesse moyenne " + data[0] + " km/h.";
        }else if (key.equals("area")) {
            data[0] = "L'aire est de " + data[0] + " km^2.";
        }else if (key.equals("period")) {
            data[0] = "Met " + data[0] + " jours pour faire le tour du soleil.";
        }else if (key.equals("density")) {
            data[0] = "Fait " + data[0] + " kg par m^3.";
        }else if (key.equals("surfaceGrav")) {
            data[0] = "La gravité est de " + data[0];
        }else if (key.equals("escapeVelocity")) {
            data[0] = "Vitesse de libération " + data[0];
        }else if (key.equals("date")) {
            data[0] = "Découvert le " + data[0];
        }else if (key.equals("discoverer")) {
            data[0] = "Découvert par " + data[0];
        }else if (key.equals("childLabel")) {
            data[0] = "Les objets suivants orbitent autour : " + data[0];
        }else if (key.equals("perimeter")) {
            data[0] = "Le perimetre de l'astre est de : " + data[0] + " km.";
        }else if (key.equals("diameter")) {
            data[0] = "Le diameter de l'astre est de : " + data[0] + " km.";
        }

        json.put(key, data);
    }
}
```

La données dist donne deux informations, la distance à l'apogée et au périgée.

Je cast donc les données pour les comparer et ainsi savoir quelle données correspond à quels informations. Pour des raisons de taille j'ai du caster en long, le type int ne pouvant pas prendre en charge certaine valeur au vu de leur taille.



Le try/catch à pour but de récupérer les données. Ne sachant pas si pour une clés nous avons une valeur unique ou un tableau, donc dans le cas ou il s'agit d'une valeur simple cela déclenchera le catch qui initialisera un tableau à une taille de 1. Dans ce tableau nous ajoutons donc notre données.

```
String key = (String) keys.next();
String[] data;
try {
    data = (String[]) json.get(key);
}catch(Exception e){
    data = new String[1];
    data[0]= (String) json.get(key);
}
```

Affichage des données

Une fois les données récupérées par la partie AJAX la donnée sera attribué à la div selon l'identifiant de la données. Une fois une valeur assigné à la DIV elle sera affichée dans la popup.

```
var data=httpRequest.responseText;
var jsonResponse = JSON.parse(data);
var len = Object.keys(jsonResponse).length;
var keys = Object.keys(jsonResponse);
if (len == 0){document.getElementById("data"+planet+"load").innerHTML = "Pas de données...";}
else{
    document.getElementById("data"+planet+"load").style.display = 'none';
    }
for (var i = 0; i < len-1; i++)
{
    var name = keys[i];
    if (name == 'img'){
        document.getElementById("data"+planet+"img").src =  jsonResponse[name];
        document.getElementById("data"+planet).style.display = 'block';
    }else{
        console.log(name + " and " + jsonResponse[name])
        document.getElementById("data"+planet+""+name).innerHTML =  jsonResponse[name];
        document.getElementById("data"+planet+""+name).style.display = 'block';
    }
```

Dans la partie HTML j'initialise des balises qui contiendront les données pour chaque planètes. Lors de l'initialisation une variable allplanets est créée et contient le nom de toutes les planètes trouvées. Avec thymeleaf, pour chaque item de la liste sera mis en place les balises de la popup.

```
<div th:each="planet : ${allplanets}">
            <!-- Modal -->
      <div class="modal fade" th:id="${planet.name+'modal'}" role="dialog">
        <div class="modal-dialog">

          <!-- Modal content-->
          <div class="modal-content">
            <div class="modal-header">
              <button type="button" class="close" data-dismiss="modal">&times;</button>
              <h4 class="modal-title" th:text="${planet.name}"></h4>
            </div>
            <div th:id="${'data'+planet.name+'load'}" class="modal-body">
              <p>Chargement des données.</p>
            </div>
            <div th:id="${'data'+planet.name+'name'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'abstractd'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'atmosphereComposition'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'averageSpeed'}"style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'formerName'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'orbitalPeriod'}" style="display:none;" class="modal-body">
              <p></p>
            </div>

            <div th:id="${'data'+planet.name+'periapsis'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'surfaceGrav'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'density'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'apoapsis'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'volume'}" style="display:none;" class="modal-body">
              <p></p>
            </div>
            <div th:id="${'data'+planet.name+'satellites'}" style="display:none;" class="modal-body">
              <p></p>
```

Bibliographie

https://jena.apache.org/documentation/query/index.html
https://docs.spring.io/spring-framework/docs/current/reference/html/
https://www.w3.org/TR/rdf-sparql-query/
https://wiki.dbpedia.org/
https://www.wikidata.org/wiki/Wikidata:Main_Page