Département d'informatique

HMIN218 - Web Sémantique

Projet Web Sémantique

Travail réalisé et soutenu par :
Abid Hamza
El Ayyadi Ikram
Lakhdar Idrissi Meryeme
Rahmouni Nadjib

Professeur référent :
Mme Isabelle Mougenot

Année universitaire : 2020-2021

Remerciements

Nous tenons à remercier Mme Isabelle Mougenot, notre professeur référent, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à nous améliorer dans le développement de notre projet et qui a su nous guider autant dans le code que dans l'organisation .

Table des matières

1 Introduction

Ce projet portant sur le web sémantique nous a permis d'enrichir nos connaissances théoriques et pratiques sur différents langages. Ceci permet également de rentrer dans la vie active et de découvrir plus précisément le milieu professionnel.
Le but du web sémantique est de permettre aux machines de discuter entre elles dans un web intelligent, où l'information ne serait plus stockée mais assimilée par les ordinateurs afin de répondre aux besoins des utilisateurs. Notre projet consiste à découvrir le domaine du web sémantique en passant par ses normes fondamentales :
RDF/RDFS
SPARQL
OWL.
L'élaboration de ce rapport a pour principale source nos connaissances acquises tout au long du module ainsi que nos recherches personnelles.

2 Conception et modélisation UML

2.1 Diagramme de Classe

L'intérêt de ce diagramme de classe est de permettre une meilleure compréhension et visualisation du fonctionnement de notre projet, il permet aussi de fournir une description indépendante de l'implémentation des types utilisés dans notre code.
Tout d'abord nous avons la classe "Concert" qui représente notre projet et qui est constituée d'au moins 30 tickets que nous avons mis comme classe aussi, ensuite il y'a la classe "Personne" qui a comme attributs nom, nationalité et âge et qui est la classe mère de la classe Artiste qui participe à un concert et la classe Client qui peut acheter un à plusieurs tickets. Par la suite, nous avons crée deux énumerations "EnumTicket" qui représente la liste des différents types de tickets qui sont présents dans le concert et l'énumeration "Style" qui représente les types de musique joués par les artistes.

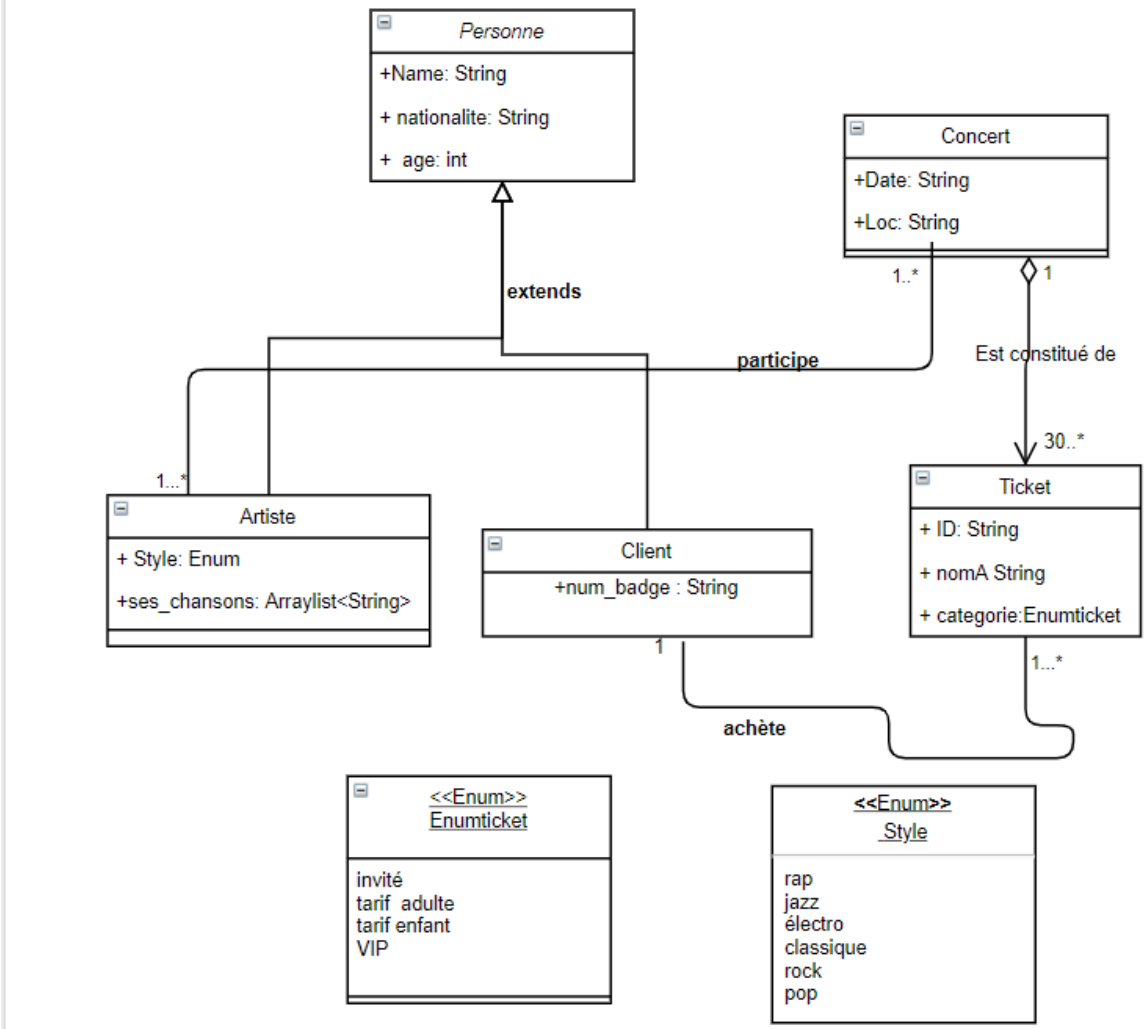


FIGURE 1 – Diagramme de Classe

2.2 Diagramme d'objet

Dans ce diagramme, nous avons initialisé les classes du diagramme UML avec des exemples concrets.

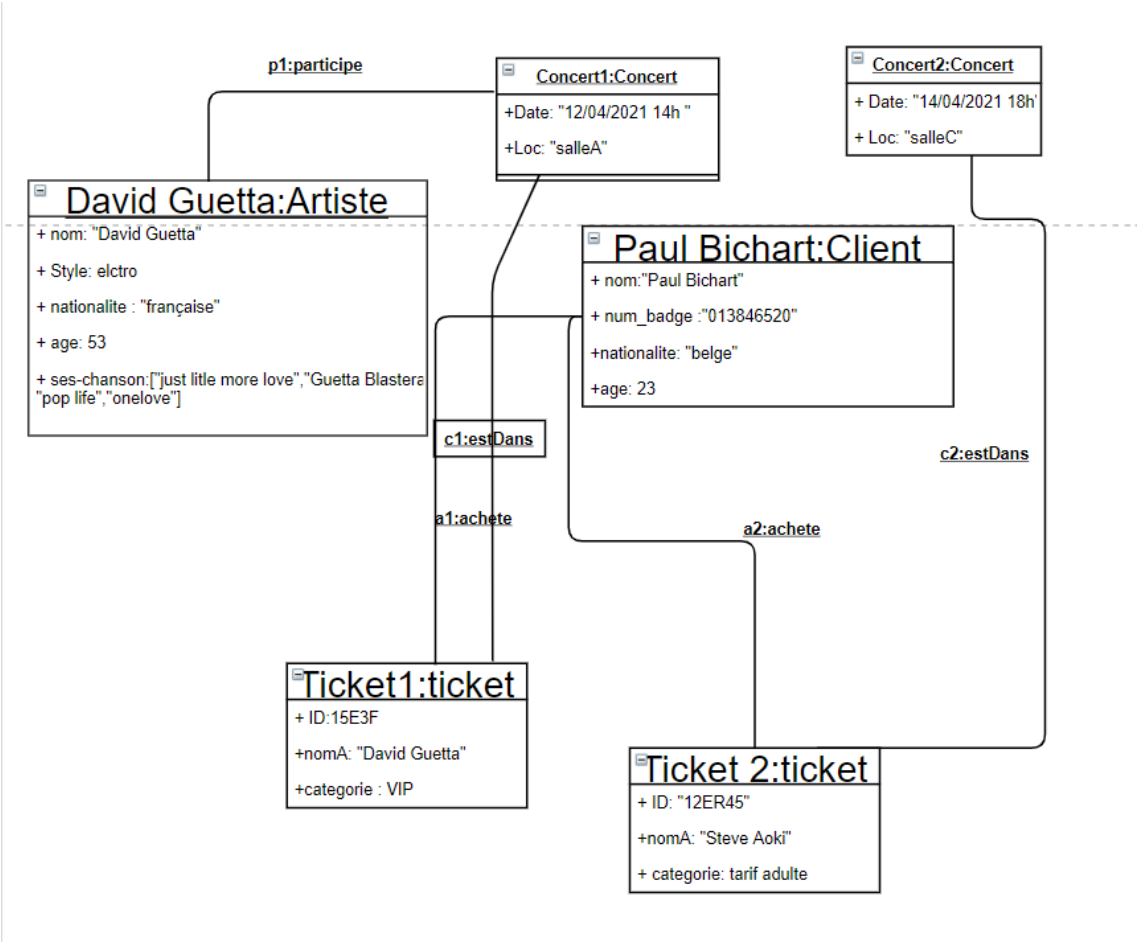


FIGURE 2 – Diagramme d'objet

3 Developpement

3.1 RDF/RDFS

Le RDF est la quatrième couche dans l'architecture du web sémantique et permet de créer des instances. Il est en couple avec le RDFS qui permet, lui, de déclarer des classes et des propriétés.

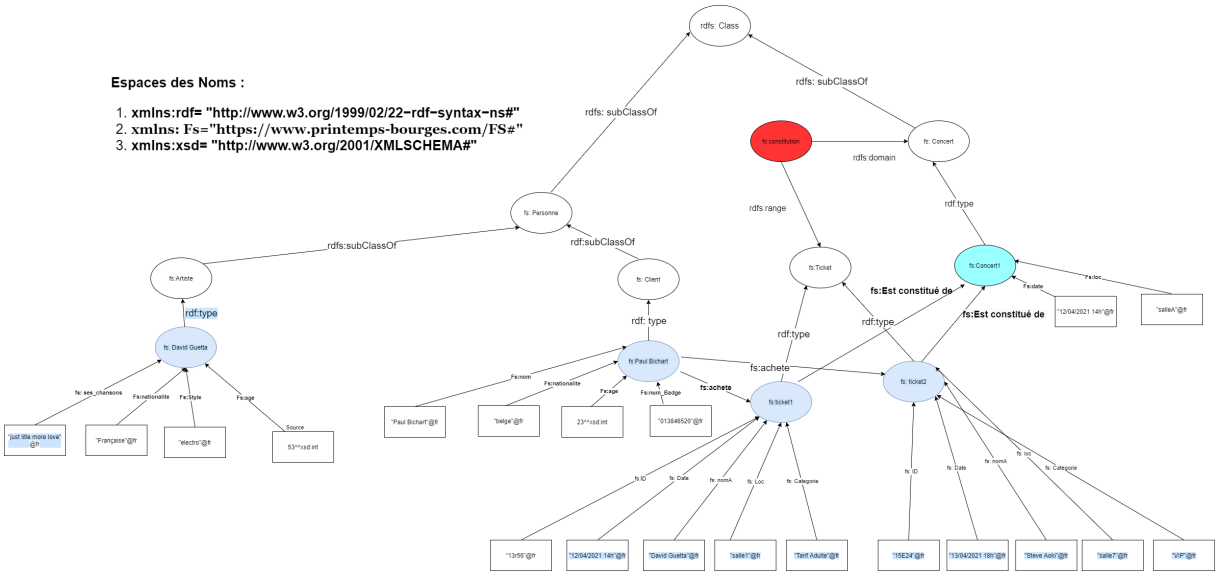


FIGURE 3 – RDF

Pour cette partie, nous avons importé la bibliothèque Jena afin de pouvoir utiliser la syntaxe RDF/XML. Ensuite nous avons créé les ressources Client, Concert, Artiste, Ticket et Personne. Ces ressources sont associées à des propriétés, par exemple un concert se définit par une date et une localisation, puis nous avons été amené à instancier toutes ces ressources (donner des ressources concrètes) et réussir à afficher les triplets.

3.2 Création des classes

Nous avons créé une classe Java pour chaque ressource qui sont donc les classes : Artiste, Client, Personne, Concert, Ticket...etc, basé sur notre modèle UML. A noter que la classe Personne est une classe mère de Artiste et CLient,
Une fois les classes créées nous leur avons associé des attributs, des constructeurs, des accesseurs et enfin des méthodes pour afficher l'objet de la classe une fois instancié.

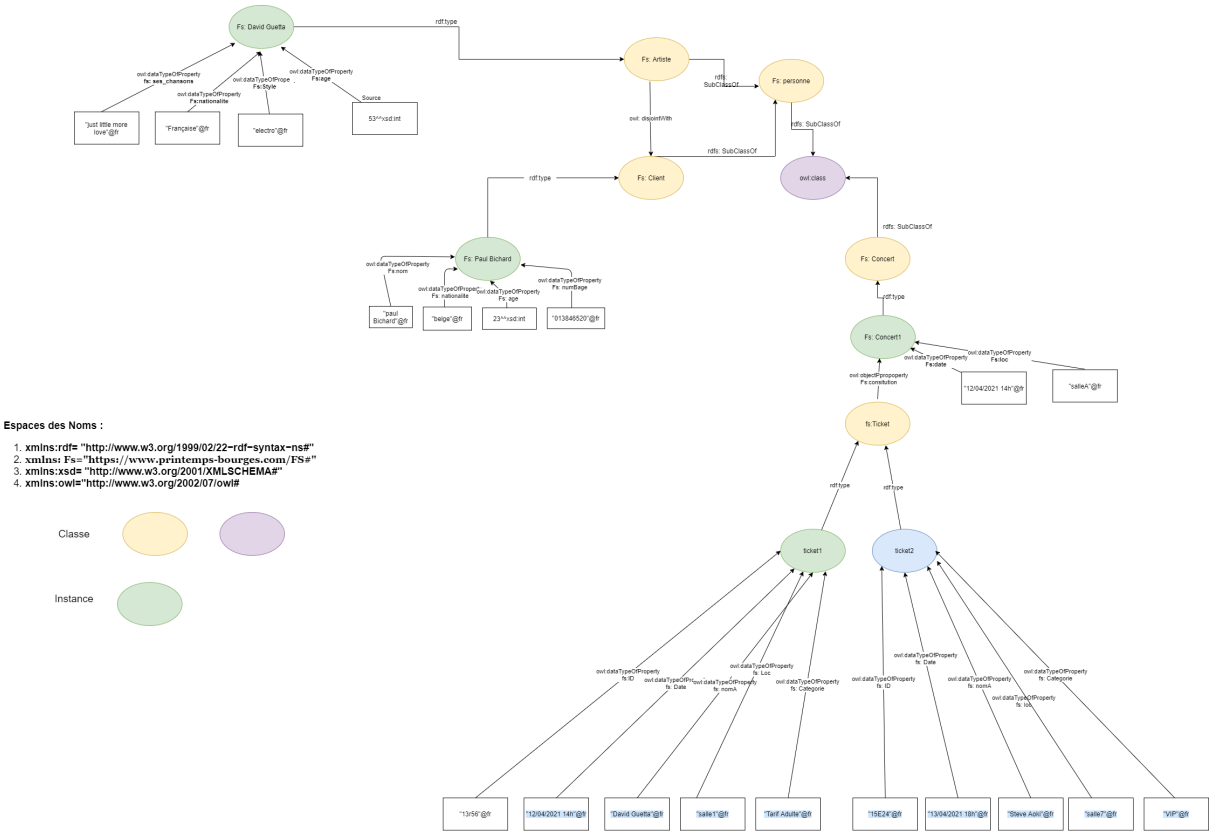
```java
package Hmin218_Projet;

import java.util.ArrayList;

import org.apache.jena.rdf.model.Model;

public class Artiste extends Personne {

    private Enum style; //les styles musicales de l'artiste
    private ArrayList<String> ListeSes_chansons = new ArrayList<>(); //la localisation du concert
    //accesseurs
    public Enum getStyle() {
        return style;
    }

    public void setStyle(Enum style) {
        this.style = style;
    }

    public ArrayList<String> getListeSes_chansons() {
        return ListeSes_chansons;
    }

    //constructeurs

    public Artiste() { //constructeur vide
    }
    public Artiste(String uri, String name, String nationalit, int age, Enum style, ArrayList<String> ses_chansons) {
        super(uri, name, nationalit, age);
        this.style = style;
        this.ListeSes_chansons = ses_chansons;
    }
}
```

FIGURE 4 – Création de la Classe Artiste

3.3 Enrichissement du modèle initial

3.3.1 Ontologie

L'ontologie peut mener à bien la communication grâce à la spécification explicite qu'elle offre pour un domaine. En plus, les ontologies assurent la consistance et éliminent l'ambiguïté dans les représentations des connaissances appartenant à un domaine spécifique. Elles permettent de prendre en considération les perspectives des utilisateurs. Dans notre projet, nous avons utilisé l'ontologie OWL qui est très expressive. Dans ce langage, on distingue les propriétés qui relient des individus à des individus (propriétés d'objets). Nous nous sommes basé sur le graphe RDF pour construire notre ontologie OWL, en respectant les conditions de ce dernier. L'ontologie permet de décrire des classes, des propriétés, une logique et les instances. Cela permet d'élaborer des triplets RDF dans la mesure où une classe et une propriété ont des URI déclarées dans notre ontologie.



Espaces des Noms :

1. **xmlns:rdf= "http://www.w3.org/1999/02/22-rdf-syntax-ns#"**
2. **xmlns: Fs="https://www.printemps-bourges.com/FS#"**
3. **xmlns:xsd= "http://www.w3.org/2001/XMLSCHEMA#"**
4. **xmlns:owl="http://www.w3.org/2002/07/owl#**

FIGURE 5 – OWL

4 SPARQL

4.1 Généralités

Le langage qui permet d'interroger le RDF, c'est le SPARQL. Il offre les moyens de parcourir un graphe RDF. Le SPARQL est considéré comme une technologie clé du Web sémantique. En plus d'offrir un langage de requêtes , le SPARQL fournit également un langage de résultats permettant de représenter les réponses d'une requête ainsi qu'un protocole permettant de soumettre une requête à un serveur distant.

4.2 Requetes SPARQL

Grâce à la bibliothèque Jena que nous avons importé, nous avons pu créer des modèles et construire des requêtes de type Query et les afficher par la suite.
Une requête SPARQL a pour but de trouver un ou plusieurs membres d'un ensemble de triplets. Pour ce faire, chaque triplet est exprimé selon la syntaxe N3 dont certains membres (le sujet et/ou le prédicat et/ou l'objet) sont inconnus et sont remplacés par des variables. Dans les requêtes SPARQL les variables sont précédées d'un " ?", l'identification des préfixes permet de ne pas répéter les URI dans toute la requête, SELECT identifie les variables à renvoyer dans la réponse, WHERE défini les conditions à respecter et qui sont écrites sous forme de triplet.
Prenons l'exemple de cette requête : String rdq = prolog1 + NL + prolog2 + NL + prolog3 + NL + "SELECT ?class (count(?Artiste) as ?nbre d'Artiste) WHERE " + " ?Artiste rdf :type ?Personne ." ;
L'objectif initial de la requête est de trouver toutes les ressources reliées à la ressource représentant Artiste avec la propriété rdf :type qui permet de définir les hiérarchie de la classe (Tout Artiste est une personne).

```java
package Hmin218_Projet;

import java.util.Iterator;

public class Sparql_2 {
    public static final String NL = System.getProperty("line.separator");

    public static void main(String[] args) {
        Model m = ModelFactory.createDefaultModel();
        String rdf_file = "festival.n3";
        m.read(rdf_file);
        String Fs ="https://www.printemps-bourges.com/FS#";
        String prolog1 = "PREFIX rdf: <"+Fs+">";
        String prolog2 = "PREFIX rdf: <"+RDF.getURI()+">" ;
        String prolog3 = "PREFIX rdfs: <"+RDFS.getURI()+">" ;

        //affichons le nombre d'artiste en fonction de concert "class" quand y'a au moins un concert

        String rdq = prolog1 + NL + prolog2 + NL + prolog3 + NL +
                "SELECT ?class (count(?Artiste) as ?nbre d'Artiste) WHERE { " + "?Artiste rdf:type ?Personne .}" ;
        Query query = QueryFactory.create(rdq);
        QueryExecution qexec = QueryExecutionFactory.create(query, m);
        query.serialize(new IndentedWriter(System.out,true));
        System.out.println();
        try {
            ResultSet rs = qexec.execSelect();
            ResultSetFormatter.out(System.out, rs, query);
        }

        finally {qexec.close();}


    }

}
```

FIGURE 6 – Sparql Artiste

5 TripleStore

Dans cette partie nous avons conçu une base de données pour stocker et récupérer les données RDF crées précédemment. À noter que la récupération de ces données se fait sous forme de triplet, nous n'avons donc pas besoin de phase d'initialisation pour enregistrer de nouvelles données.

6 Gestion de Projet

6.1 Diagramme de Gantt Prévisionnel

Nous avons prévu de terminer le développement du projet 2 semaines avant la date de la présentation, pour pouvoir se concentrer sur le rapport et les différents bugs que nous pouvons rencontrer au niveau du code. Nous n'avons pas eu de complications en terme de répartitions des tâches. Nous nous sommes répartis les tâches en identifiant les parties importantes du projet . Les trois premières tâches : choisir une thématique, diagramme de classe et diagramme d'objet, étant primordiales ont fait que nous voulions tous y contribuer afin que cela nous aide à comprendre le projet et à avoir une bonne visualisation de RDF / RDFS. Hamza et Nadjib se sont chargés principalement de RDF/RDFS, OWL, alors que Ikram et Meryeme se sont concentrées sur le langage SPARQL. Cependant nous nous entraidons sur la globalité du projet.

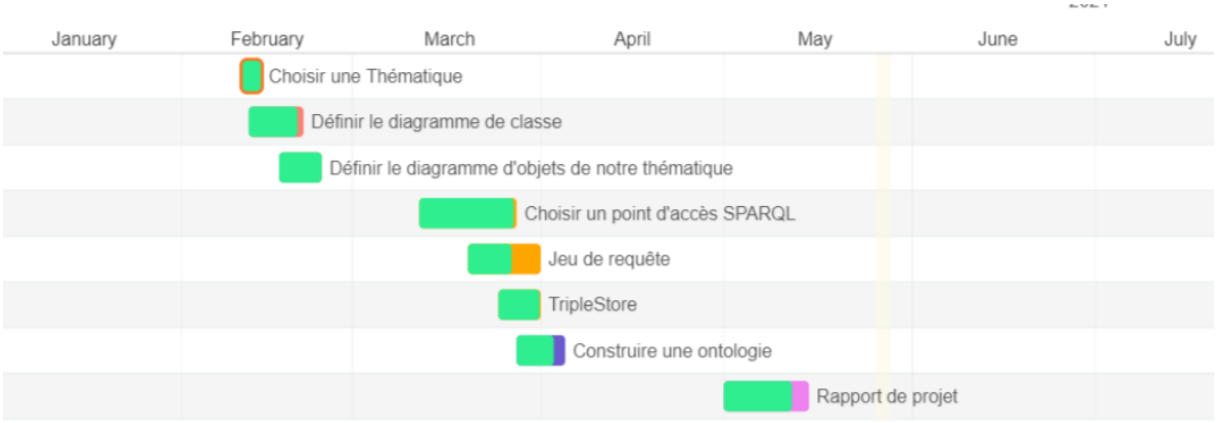


FIGURE 7 – Diagramme de Gantt Prévisionnel

6.2 Diagramme de Gantt final

En comparaison avec le diagramme de Gantt prévisionnel, nous constatons que certaines tâches ont pris plus de temps que prévu. Une meilleure organisation aurait pu nous aider à éviter cela.

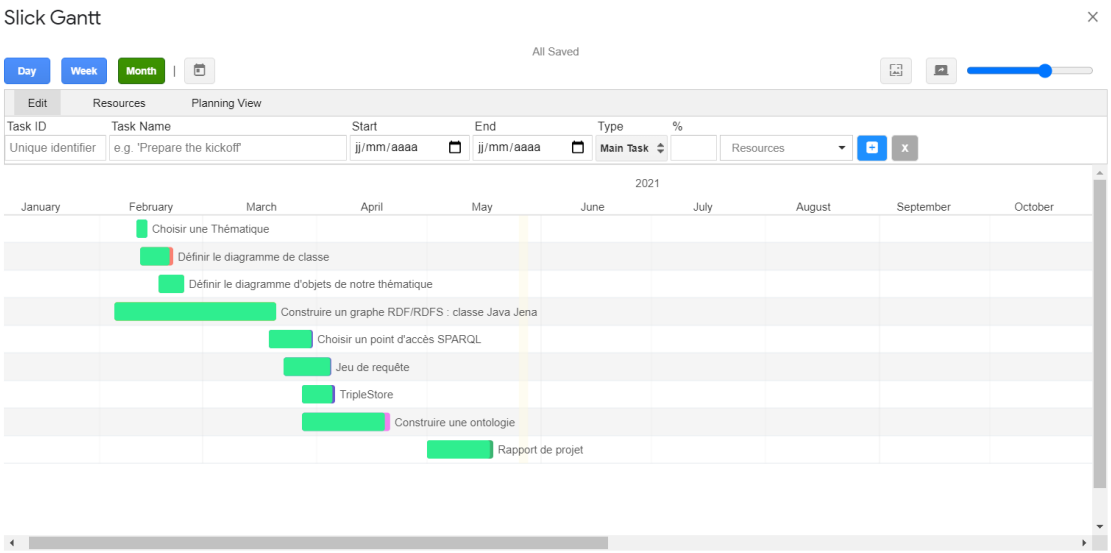


FIGURE 8 – Diagramme de Gantt Final

7 Conclusion

Dans ce projet, nous avons appris à construire un graphe RDF/RDFS en passant par la conception UML, jeu de requêtes pour interroger notre RDF et une ontologie OWL pour enrichir notre modèle.
Nous avons utilisé la bibliothèque Jena lors de la réalisation de ce travail, une API Java que nous avons pu découvrir, pour faciliter le développement d'application pour le web sémantique. Elle permet de manipuler (lecture, écriture et traitement) des fichiers RDF, ainsi que des ontologies RDFS et OWL. Elle fournit aussi une implémentation du moteur SPARQL pour faire des requêtes sur des données RDF, ce projet nous a donc permis d'enrichir nos connaissances en java et découvrir une nouvelle manière d'utiliser ce langage.

8 Annexes

8.1 RDF

```
//prop Ticket
            Property NomA= m.createProperty(Fs+"NomA");
            Property ID= m.createProperty(Fs+"ID");
            Property categorie= m.createProperty(Fs+"categorie");
            Property date= m.createProperty(Fs+"date");
            Property loc= m.createProperty(Fs+"loc");
//ticket
            ticket1.addProperty((Property)ticket1,m.createLiteral("RZER31",XSD.getURI() + "String"));

            // affichage des triplets
            // N3 (ou TURTLE), N-TRIPLE, RDF/XML, JSON-LD
            m.write(System.out, "TURTLE");
            try {FileOutputStream outStream = new FileOutputStream("bourges.n3");
            m.write(outStream, "N3");
            outStream.close();
            }
            catch (FileNotFoundException e) {
                System.out.println("file not found" );
            }
            catch (IOException e)
              {System.out.println("IO problem");}
            }
            catch (Exception e) {
            System.out.println("failure" + e);
        }
```

FIGURE 9 – creation, initialisation et affichage d'une ressource

8.2 SPARQL

```java
public class Sparql_1 {
    public static final String NL = System.getProperty("line.separator");

    public static void main(String[] args) {
        Model m = ModelFactory.createDefaultModel();
        String rdf_file = "festival.n3";
        String prolog1 = "PREFIX rdf: <"+RDF.getURI()+">" ;
        m.read(rdf_file);
        String rdq = prolog1 + NL +
                "SELECT ?s ?p ?o WHERE { ?s ?p ?o}" ;
        Query query = QueryFactory.create(rdq);
        QueryExecution qexec = QueryExecutionFactory.create(query, m);
        try {
            Iterator<QuerySolution> results= qexec.execSelect();
            RDFVisitor aVisitor = new Un_Visiteur();
            System.out.println("tous les triplets");
            for (;results.hasNext();) {
                QuerySolution sol = results.next();
                RDFNode s = sol.get("s");
                RDFNode p = sol.get("p");
                RDFNode o = sol.get("o");
                System.out.print(s.visitWith(aVisitor)+"    ");
                System.out.print(p.visitWith(aVisitor)+"    ");
                System.out.println(o.visitWith(aVisitor));
            }

        }

        finally {qexec.close();}
```

FIGURE 10 – requête SPARQL

8.3 TripleStore

```java
package Hmin218_Projet;
import org.apache.jena.query.Dataset;
import org.apache.jena.tdb.TDBFactory;

public class Wikidata2 {
        public static void main(String[]args)
        {
            String directory="festival";
            Dataset dataset=TDBFactory.createDataset(directory);

            String sparqlQueryString="SELECT * WHERE "+ "" ?s ?P <http://www.festival_bourge.fr/David_Guetta>} "  ;
            QueryExecution qexec.execSelect();
            ResultSet results =qexec.execSelect();
            ResultSetFormatter.out(results);
            qexec.close();
            dataset.close();
             }
}
```

FIGURE 11 – wikidata query