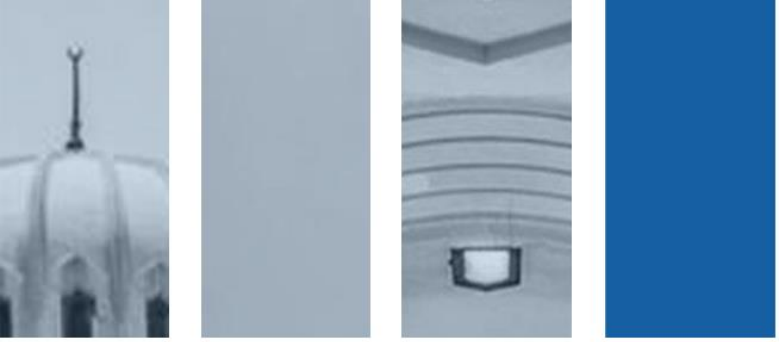


Music Ontology

Program: CESS

Course Code: CSE488

Course Name:

Ontologies and the Semantic Web

Submitted to:

Dr Ensaf Hussein

Eng Dina Amr

Submitted by:

1. *Aliaa Abd El Karim* *18P1241*
2. *Farah Loay* *18P7388*
3. *Malak Mohamed* *18P9114*
4. *Nouran Ahmed* *18P4496*
5. *Nourhan Ayman* *18P5425*

Ain Shams University
Faculty of Engineering

Spring 2023 Semester

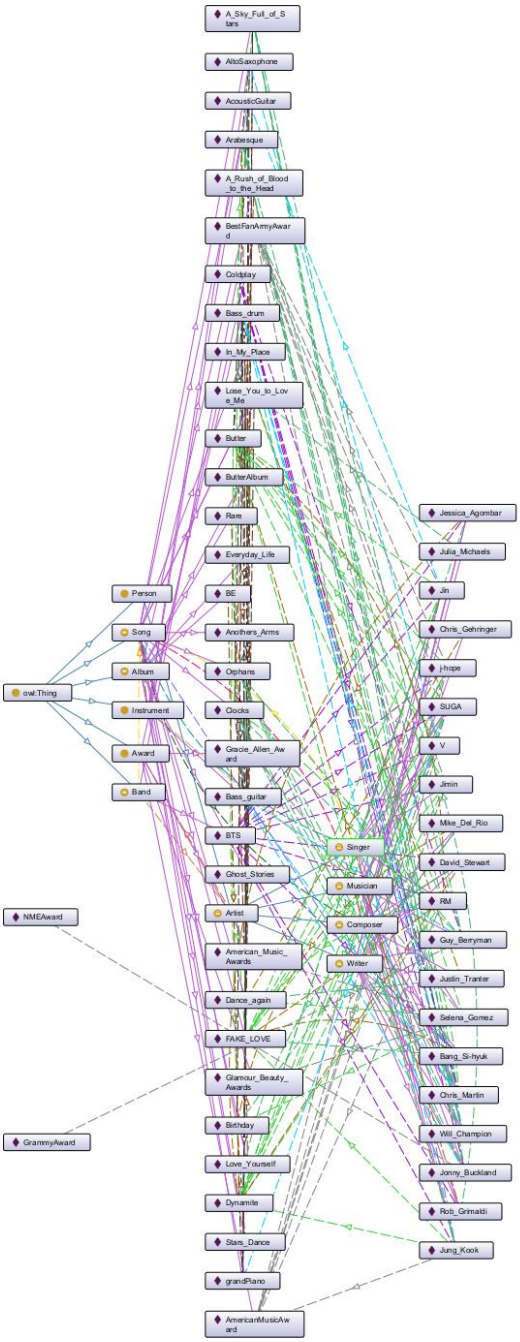
Students' Full Names:

Aliaa Abd El Karim	18P1241
Farah Loay Hamed Alanany	18P7388
Malak Mohamed Gadelrab Abdou Madkour	18P9114
Nouran Ahmed Abd Elhameed	18P4496
Nourhan Ayman Aly Reda Mohamed El Adel	18P5425

Table of Contents

1. INTRODUCTION

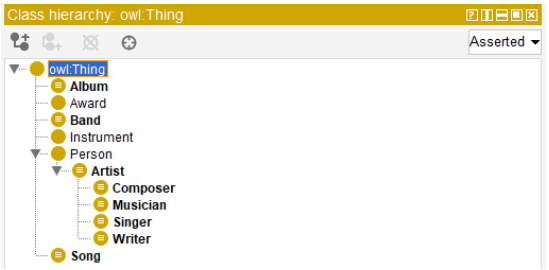
Our project is based on a Music Ontology that contains songs, their artists, including the singer singing, the musician playing the background music, the writer that wrote the lyrics and the composer. Our ontology also includes the bands and the artists they include, the awards won by the artist and the instruments played by the musician. We used the Protégé ontology editor developed by Stanford University to build our Music Ontology. Then we created a java application and added Jena jar files to be able to create a GUI frame where we will be able to write a SPARQL query, press execute and see the results of the query displayed in the form of a table. We also created a JavaScript web application that contains a text field for the queries, a button to run the query and a table to display the results.



2. CLASSES

Define the following classes:

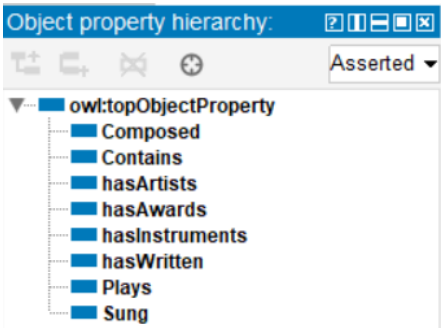
- Album: To contain the songs.
- Award: Given to artists.
- Band: To have artists that play songs together.
- Instrument: The instruments that the musician will play.
- Person: Parent class of Artist, for expansion purposes if wanted to add different type of people that are not artists.
- Artist: Parent class of Composer, Musician, Singer, and Writer, which are all considered artists.
- Composer: Composes songs.
- Musician: Plays instrument.
- Singer: Sings the songs.
- Writer: Write the song lyrics
- Song: The actual song class that has a name, language and other properties discussed in the following sections.



3. OBJECT PROPERTIES

➢ Define the following object properties:

- Composed: connects Composer to Song.
- Contains: connects Album to Song.
- hasArtists: connects Band to Artist.
- hasAwards: connects Artist to Award.
- hasInstruments: connects Song to Instrument.
- hasWritten: connects Writer to Song.
- Plays: connects Musician to Instrument.
- Sung: connects Band/Singer to Song.



4. RESTRICTIONS

Define minimum and maximum cardinality restrictions on the object properties:

- Each Album contains at least 1 and at most 15 songs:

  → Add a restriction on the Contains property with a minimum cardinality of 1 and a maximum cardinality of 15.

- Each Band has at least 2 Artists:

  → Add a restriction on the hasArtists property with a minimum cardinality of 2.

- Each Band sung at least 1 song:

  → Add a restriction on the Sung property with a minimum cardinality of 1.

- Each Song has at least 1 instrument:

  → Add a restriction on the hasInstruments property with a minimum cardinality of 1.

- Each Composer composed at least 1 Song:

  → Add a restriction on the Composed property with a minimum cardinality of 1.

- Each Musician Plays at least 1 Instrument:

  → Add a restriction on the Plays property with a minimum cardinality of 1.

- Each Singer sung at least 1 Song:

  → Add a restriction on the Sung property with a minimum cardinality of 1.

- Each Writer has Witten at least 1 Song:

  → Add a restriction on the hasWritten property with a minimum cardinality of 1.

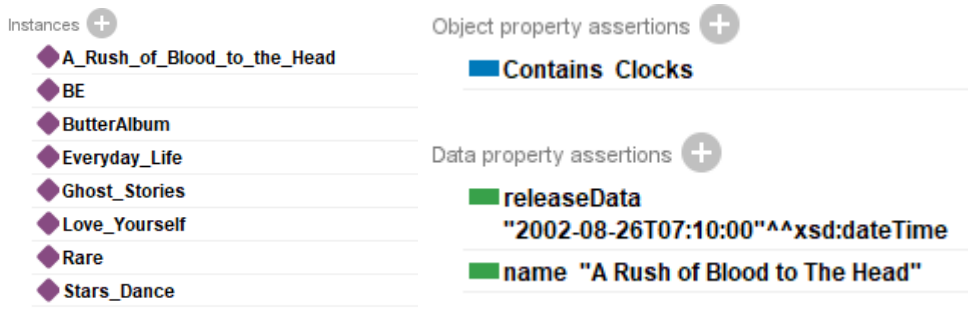
5. DATA PROPERTIES

Data Property	Domain	Range
- Age	- Person	- Integer
- Agent	- Singer	- String
- Country	- Band	- String
- Date	- Band - Award	- Data Time
- Description	- Song	- String
- Emotions	- Song	- String
- Gender	- Person	- String
- ID	- Song	- long
- Language	- Song	- String
- Name	- Song - Album - Band - Award - Instrument - Person	- String
- Nationality	- Person	- String
- Release Date	- Album - Song	- Date Time
- Time Duration	- Song	- Decimal
- Top Hits	- Artist	- String
- Type	- Award - Instrument	- String
- Views	- Song	- Decimal
- Genre	- Song	- String

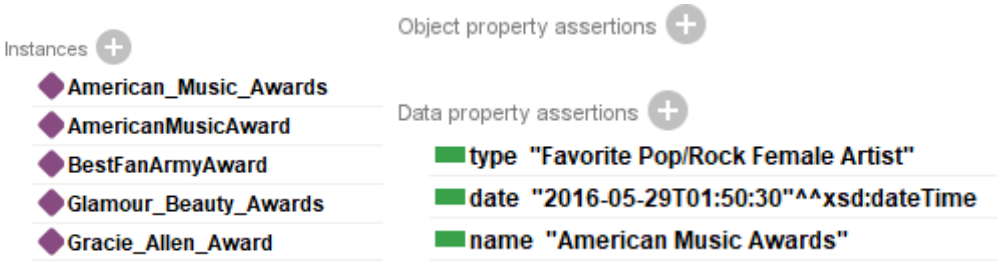
6. INSTANCES

After adding the classes, their data and object properties, along with their restrictions we added some instance to be able to query on. Some of these instances are mentioned below.

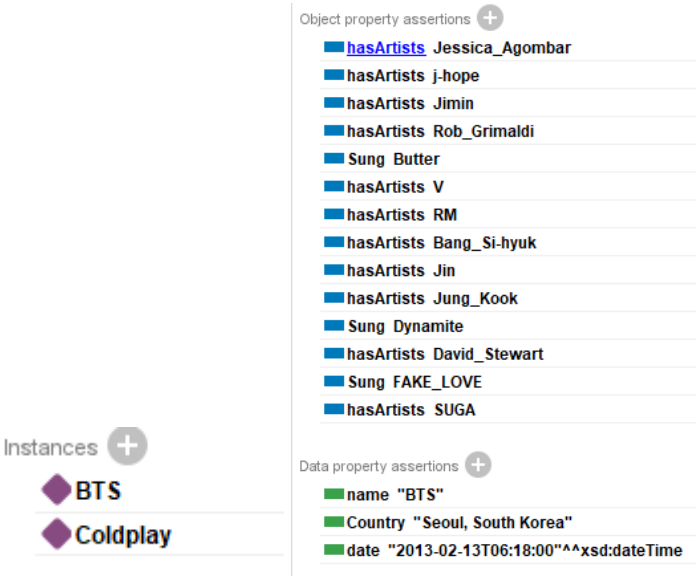
Instances of Album and example of data and object property assertions of the first instance:



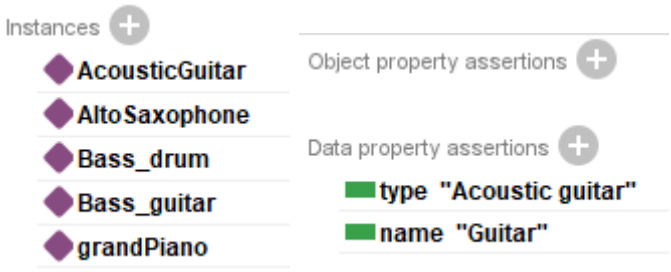
Instances of Award and example of data and object property assertions of the first instance:



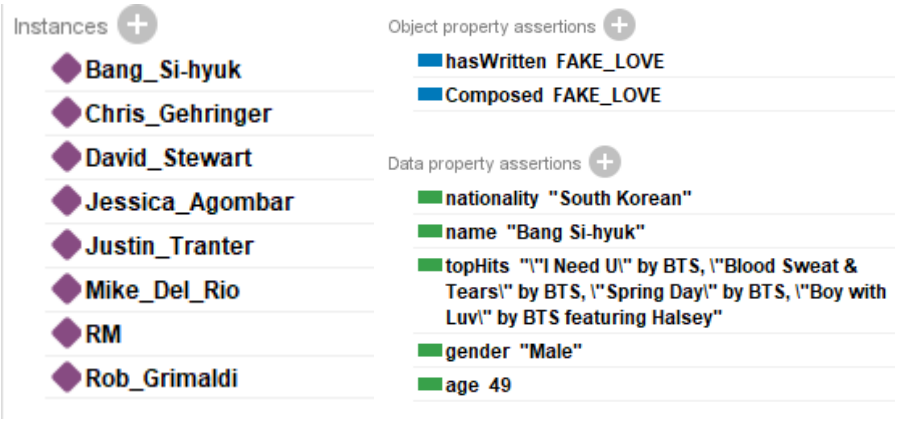
Instances of Band and example of data and object property assertions of the first instance:



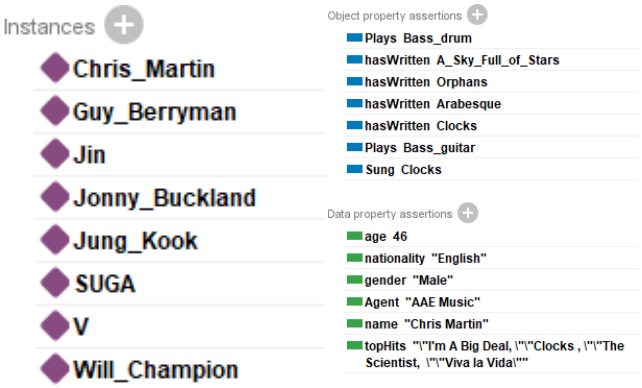
Instances of Instrument and example of data and object property assertions of the first instance:



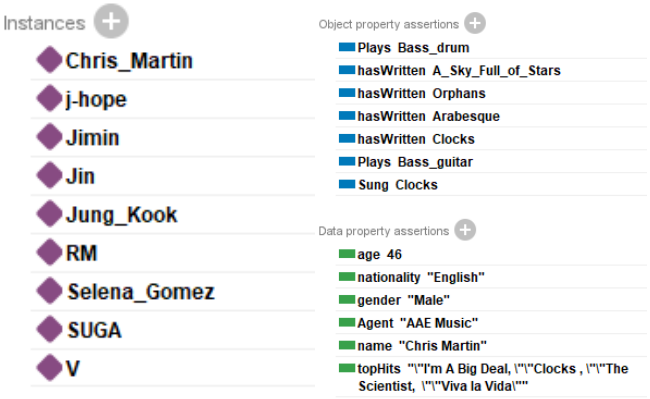
Instances of Composer and example of data and object property assertions of the first instance:



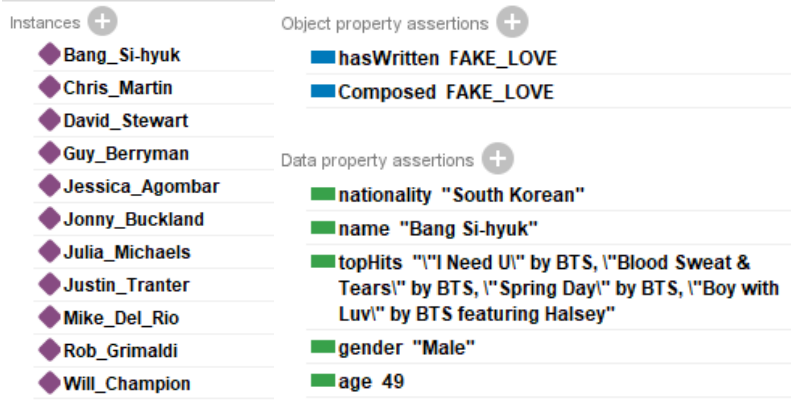
Instances of Musician and example of data and object property assertions of the first instance:



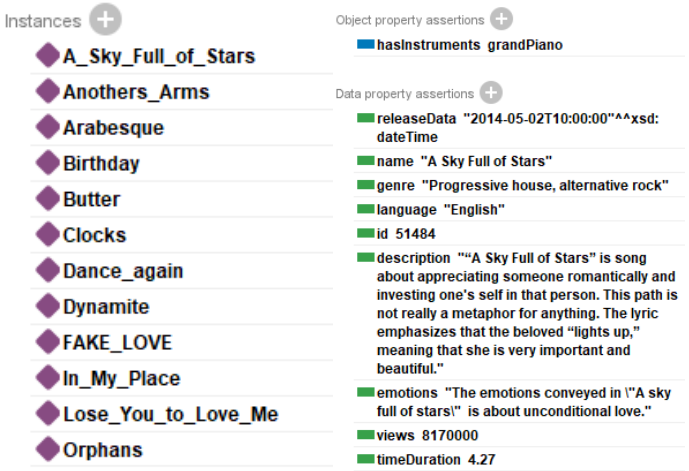
Instances of Singer and example of data and object property assertions of the first instance:



Instances of Writer and example of data and object property assertions of the first instance:



Instances of Song and example of data and object property assertions of the first instance:

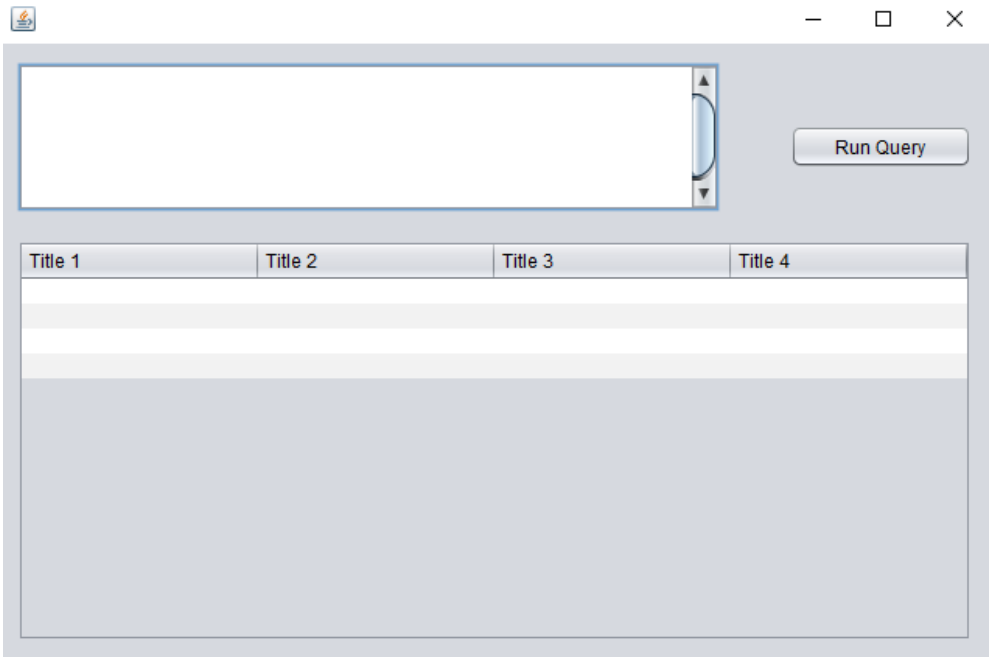


7. JAVA APPLICATION

After creating the java application, we added the Jena jar files in the Libraries folder, then we added the OpenOWL.java in our package, and added the directory path to our owl file.

```
        OntModel mode = null;
    mode = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM_RULE_INF );
    java.io.InputStream in = FileManager.get().open( "C:/music_ont.owl" ); // be sure file into c:\
```

Later, we added a JFrame form, and added a text field to write the queries, a button to run the query and a table to display the result of the query.



In the run_query_btnActionPerformed function, in the DisplayFrame.java, first we check if query field is empty we return, if not empty we assign the query string to variable q, then we assign the prefixes of our ontology, rdf, rdfs, and owl, along with our query all to variable querystring. GetResultAsString in the OpenOWL.java is called and the query string is passed to it, its result is assigned to the result variable.

Finally, if the results value is rien we return, otherwise we split the results into lines and split the lines to labels with respect to | character that's between the labels and put these labels in the column names, then we split the rest of the lines again and put these values in the cells of the table and set the model to the result table.

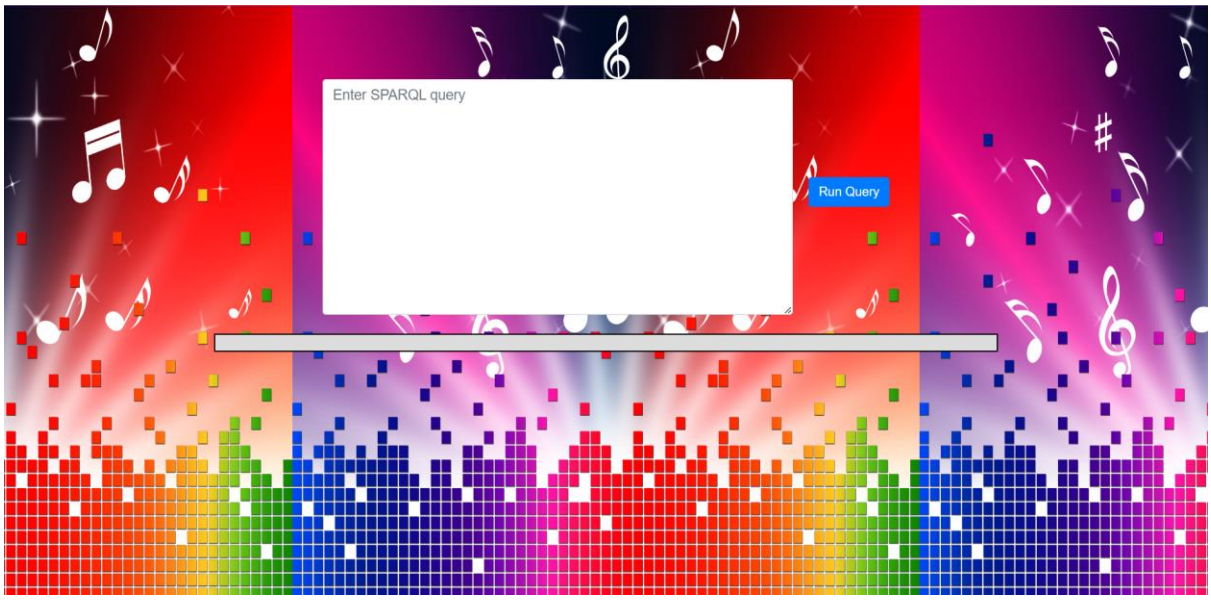
```
private void run_query_btnActionPerformed(java.awt.event.ActionEvent evt) {

    if("".equals(query_txt.getText())) // check if text area is empty
    {
    System.out.println("empty string");
    return;
    }

    String q = query_txt.getText(); // get query from text area

    try {
        // OntModel model = OpenOWL.OpenConnectOWL();
        String queryString;
        // form the query string
        queryString = "PREFIX m:<http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3/> "
                +"PREFIX owl: <http://www.w3.org/2002/07/owl#>"
                +"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"
                +"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" + q;


        String results = OpenOWL.GetResultAsString(queryString); //GetResultAsString gets query result as string
        System.out.println(results);
```

```
          if (results.equals("rien")){ //if the result is nothing
              return;
          }
          String []lines = results.split("\\R");//split the result by new line
          String [] labels = lines[1].split("\\|");//split each line by | charecter
          String [] colNames = new String [labels.length -1];

          // get column names from secound line
          for (int i=1; i< labels.length; i++){

              colNames[i-1] = labels[i].trim();
               System.out.println(labels[i]);

          }

          Object [][] output = new Object [lines.length-3][labels.length-1] ;
```

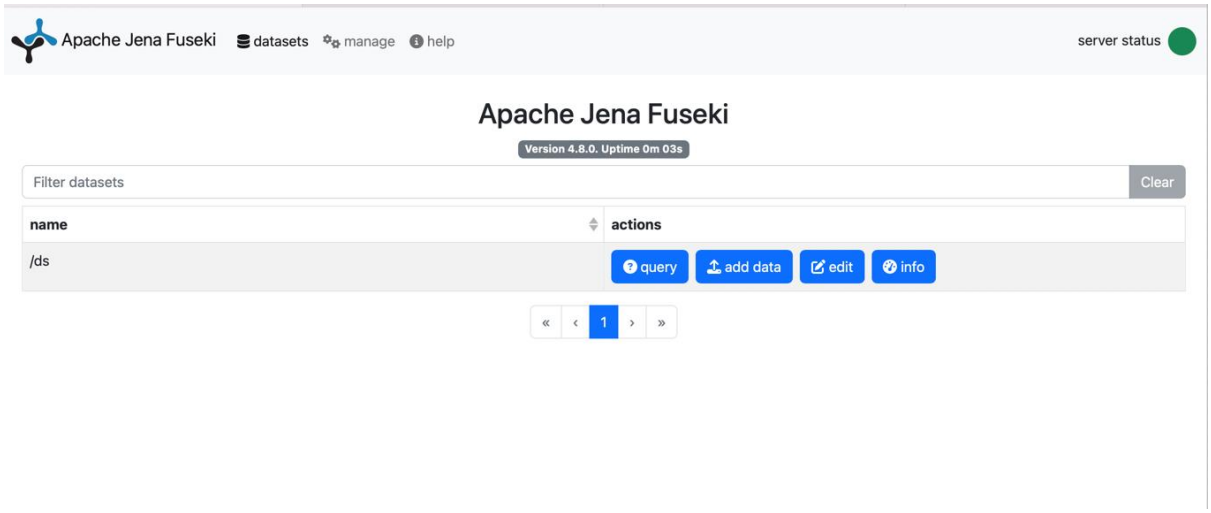
```
          // get the output of the query in 2d array by parsing the result string
          for ( int i =3; i < lines.length; i++) {

              String [] line = lines[i].split("\\|");
              for ( int j =1 ; j < line.length; j++){
                  output [i-3][j-1]=line[j].trim();
              }
          }

          res_table.setModel(new javax.swing.table.DefaultTableModel(output ,colNames)); // fill the table

        } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

8. JAVASCRIPT WEBAPP

Our web application, which enables users to input a SPARQL query and view the results in a tabular format, is created using HTML and JavaScript code. The HTML code is used to specify the layout of a web page, including the text area, background image, and results table. The query input, button interaction with the server, and result presentation in the table are all handled by the JavaScript code.



In order to produce a dataset that can be accessed via the SPARQL query language, we first uploaded our owl file to the Apache Jena Fuseki Server.



After that, we'll use JavaScript to send requests to the Fuseki server and display the results. When a user writes a SPARQL query into the text area and clicks the "Run Query" button, JavaScript code iterates through each line of the query input, determining if it begins with the keyword "PREFIX," and adding that line to the prefixes array if it does. If not, the line is added to the "query" String. The final SPARQL query is then created and transmitted to the server using the fetch () method by combining the prefix declarations and the main query into a single string. The query's results are returned by the server in JSON format. The results are then parsed by JavaScript code and shown in a table.

```
/*When the User pressed on the Run query Buttton, then this function is called*/
Button.addEventListener('click', () => {
  // Get the query input value
  const queryInputValue = queryInput.value.trim();
  if (!queryInputValue) {
    alert('Please enter a SPARQL query');
    return;
  }
  //At First, we split the input into lines
  const lines = queryInputValue.split('\n');
  /*It initializes an empty array called "prefixes" to store any prefix declarations found in the query input,
  and an empty string called "query" to store the main query.*/
  const prefixes = [];
  let query = '';
  /*It loops through each line of the query input and checks if it starts with the keyword "PREFIX" or not.
  If it does, it adds the line to the "prefixes" array. If it doesn't, it adds the line to the "query" string.*/
  for (const line of lines) {
    if (line.trim().toUpperCase().startsWith('PREFIX')) {
      prefixes.push(line);
    } else {
      query += line + '\n';
    }
  }
```

```
  /*If there is no prefixes declarations entered by the user, then the default prefixes will be added to the
  prefixes array*/
  if (prefixes.length === 0) {
    //Use the default onesss
    prefixes.push('PREFIX owl: <http://www.w3.org/2002/07/owl#>\n');
    prefixes.push('PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n');
    prefixes.push('PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n');
    prefixes.push('PREFIX ex: <http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3/>\n');
  }


  /* Then we will uses the join() method to concatenate the elements of the "prefixes" array into a single string
  concatenate this string with the query string */

  const queryWithPrefixes = prefixes.join('') + query;

 /* To handle the different types of queries, we make a regular expression to retrieve the type of the query*/
  const queryTypeRegex = /(ASK|SELECT|DESCRIBE)/i;
  const match = queryTypeRegex.exec(queryInputValue.trim());
  //Get the First matched String
  const queryType =  match ? match[1].toUpperCase() : null;

  //Indicates the location of our server on Apache jena Fureski
  let endpoint = 'http://localhost:3030/ds/query';
```

```
    /*We used the fetch() function to send a GET request to the server at the URL
    "http://localhost:3030/ds/query", where "ds" is the name of the dataset being queried.
    The query parameter is added to the URL using template literals, and the "encodeURIComponent()"
    function is used to encode the query string to ensure it is URL-safe.*/
    fetch(endpoint + '?query=' + encodeURIComponent(queryWithPrefixes))
.then(response => {
  if (queryType === 'DESCRIBE') {
    // If the query is a DESCRIBE query, parse the response as text returns a JavaScript object called "data"
    return response.text();
  } else {
    // If the query is a SELECT query, parse the response as JSON and returns a JavaScript object called "data"
    return response.json();
  }
})
.then(data => {
  console.log(data);
  if (queryType === 'ASK') {
    /*Handle ASK query results by getting the value of the boolean variable of json format
    and assigned it to result variable*/
    const result = data.boolean;
    resultsTable.querySelector('thead tr').innerHTML = '';
    resultsTable.querySelector('tbody').innerHTML = `<tr><td>${result}</td></tr>`;
```

```
  } else if (queryType === 'DESCRIBE') {
      // Handle DESCRIBE query results
      try {
        // Parse the response data as an RDF graph using rdflib.js
        const store = $rdf.graph();
        $rdf.parse(data, store, endpoint, 'text/turtle');

        // Get the triples from the graph and construct a table
         const results = store.statementsMatching();
         const tableRows = results.map(result => {
          const subject = result.subject.value;
          const predicate = result.predicate.value;
          const object = result.object.value;
          return `<tr><td>${subject}</td><td>${predicate}</td><td>${object}</td></tr>`;
        });
        resultsTable.querySelector('thead tr').innerHTML = '<th>Subject</th><th>Predicate</th><th>Object</th>';
        resultsTable.querySelector('tbody').innerHTML = tableRows.join('');
      } catch (error) {
        // If there was an error parsing the response data as Turtle, log the error and set the data variable to nul
        console.error(error);
        data = null;
      }
    }
```

```
    else {
    // Handle SELECT query results
    const results = data.results.bindings;
    /*the "map()" and "join()" functions are used to transform the results into HTML table rows and cells*/
    const tableRows = results.map(result => {
     /*Each row in the table corresponds to a single query result, and each cell in the row corresponds to
     a single variable in the SELECT clause.*/
      const variables = Object.keys(result);
      const rowCells = variables.map(variable => {
        return `<td>${result[variable].value}</td>`;
      });
      return `<tr>${rowCells.join('')}</tr>`;
    });
    const variables = Object.keys(results[0]);
    const tableHeaders = variables.map(variable => {
      return `<th>${variable}</th>`;
    });

    /*the code updates the HTML table in the web application by setting the innerHTML property of the table's <thead
    resultsTable.querySelector('thead tr').innerHTML = tableHeaders.join('');
    resultsTable.querySelector('tbody').innerHTML = tableRows.join('');
  }

  // Return the data variable
  return data;
})
.catch(error => console.error(error));

  });
```

9. SPARQL QUERY AND RESULT

Select Queries

1) List the songs(name, genre, views) sung by Selena Gomez ordered by release data and the album that each song belong to.

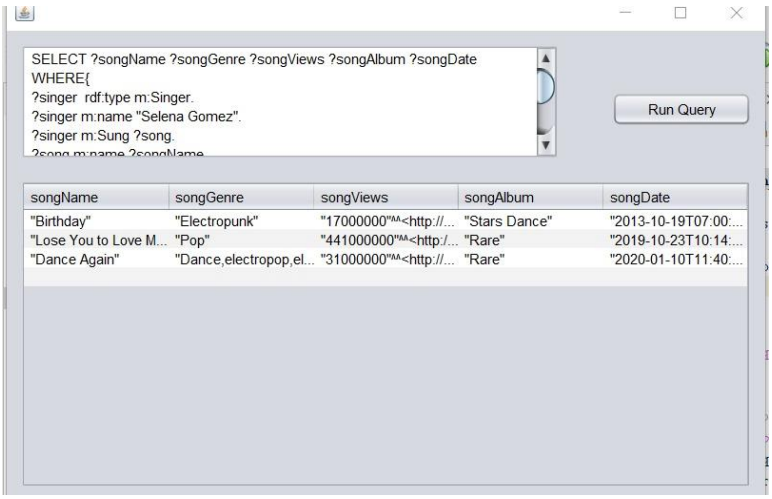
The query selects five variables to be returned in the results:

- "?songName": the name of the song
- "?songGenre": the genre of the song
- "?songViews": the number of views of the song
- "?songAlbum": the name of the album that contains the song
- "?songDate": the release date of the song

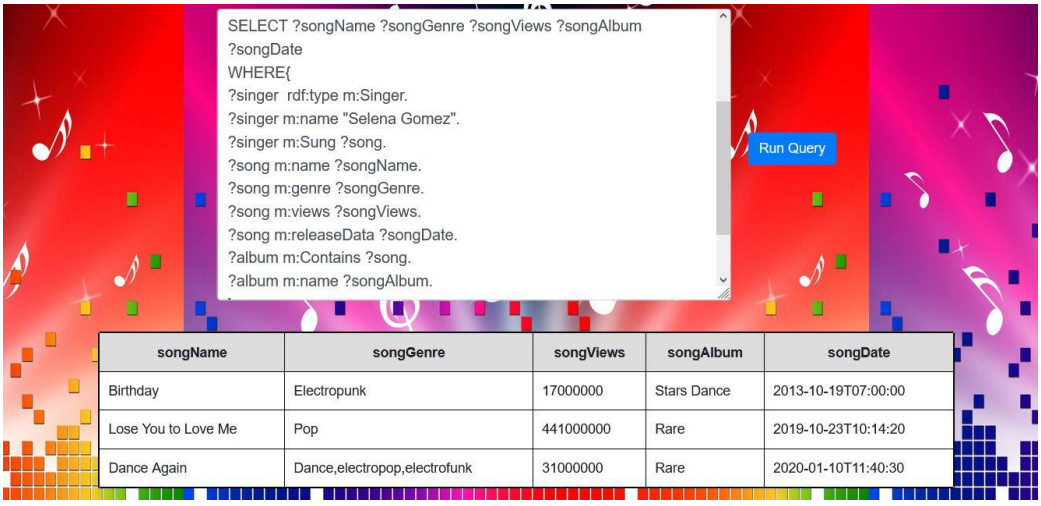
The WHERE clause specifies the conditions that the selected data must meet:

- "?singer" is an instance of the "Singer" class
- The name of the singer is "Selena Gomez"
- "?singer" has sung a song.
- And this song must have a Name, genre, number of views, release date, belongs to an album, and this album has a name.

Java App Result



Webapp Result



2) List the awards taken by Selena Gomez with the date and type.

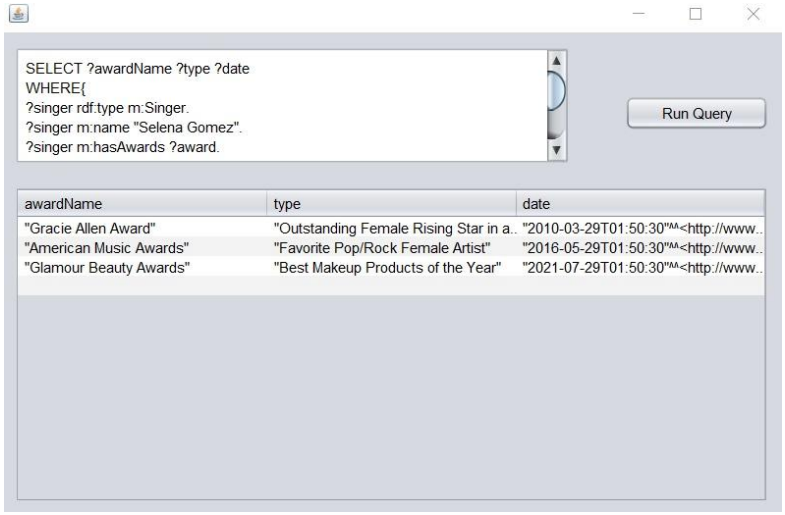
The query selects three variables to be returned in the results:

- "?awardName": the name of the award
- "?type": the type of the award
- "?date": the date on which the award was received

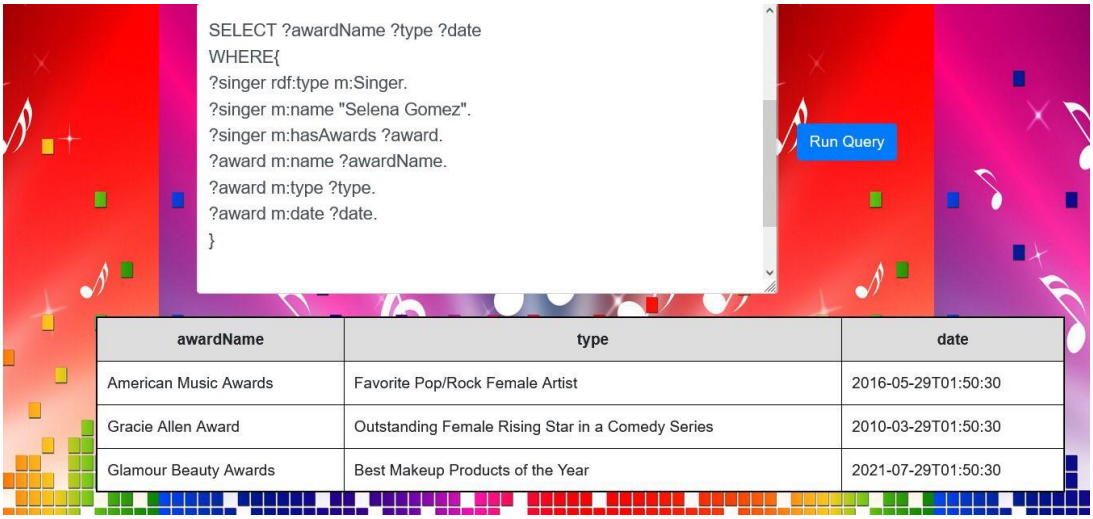
The WHERE clause specifies the conditions that the selected data must meet:

- "?singer" is an instance of the "Singer" class
- The name of the singer is "Selena Gomez"
- "?singer" has received an award
- And this award must have a Name, type, and date on which it was received

Java App Result



Web App Result



3) List the songs of the album everyday life or ghost stories with the instrument used in every song.

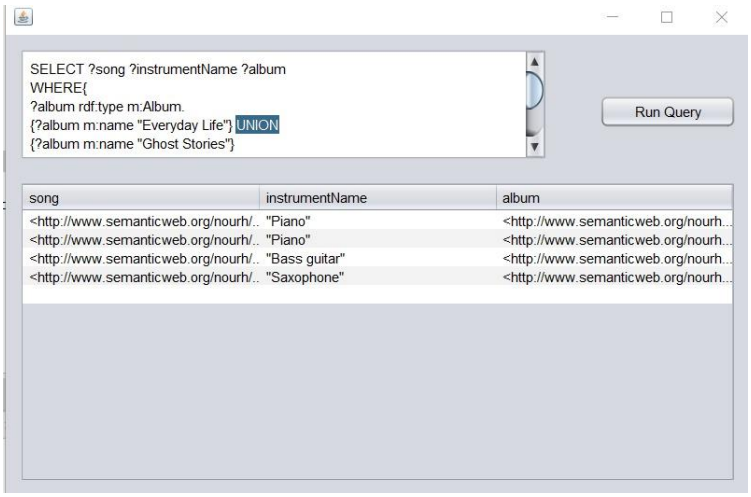
The query selects three variables to be returned in the results:

- "?song": the name of the song
- "?instrumentName": the name of the instrument used in the song
- "?album": the name of the album of the song

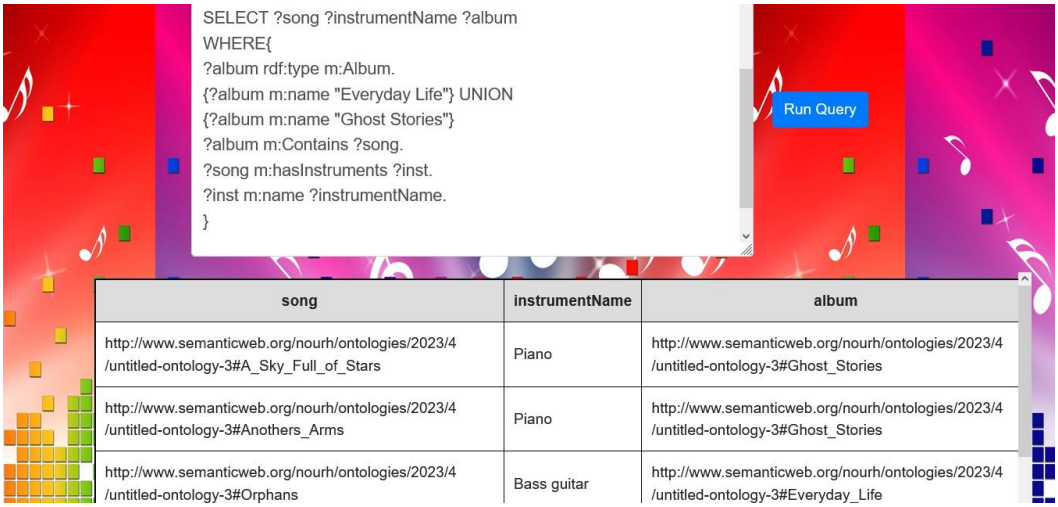
The WHERE clause specifies the conditions that the selected data must meet:

- "?album" is an instance of the "Album" class
- The name of the album is either "Everyday Life" or "Ghost Stories"
- This album contains a song
- The song has an instrument
- And this instrument has a name

Java App Result



Web App Result



4) List the names and the ages of the members of the band Coldplay and awards if they have.

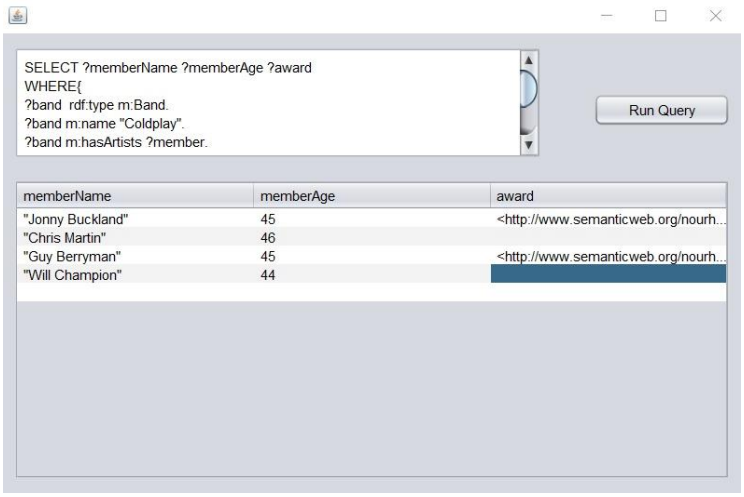
The query selects three variables to be returned in the results:

- "?memberName": the name of the band member
- "?memberAge": the age of the band member
- "?award": the name of any award received by the band member (optional)

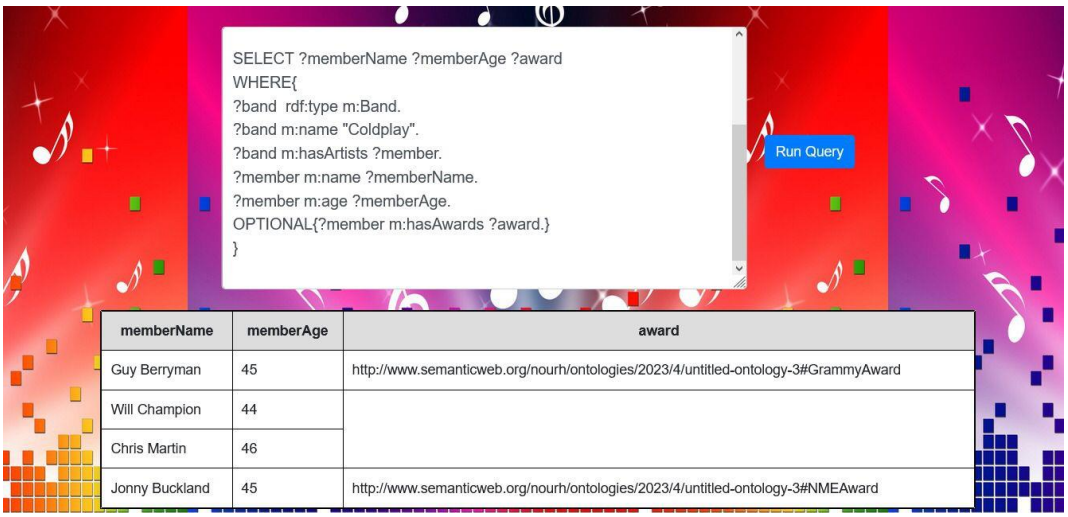
The WHERE clause specifies the conditions that the selected data must meet:

- "?band" is an instance of the "Band" class
- The name of the band is "Coldplay"
- "?band" has artists (members)
- The artist has name, and age.
- And the artist may have awards

Java App Result



Web App Result



5) List the songs of BTS band that exceed million views and their duration.

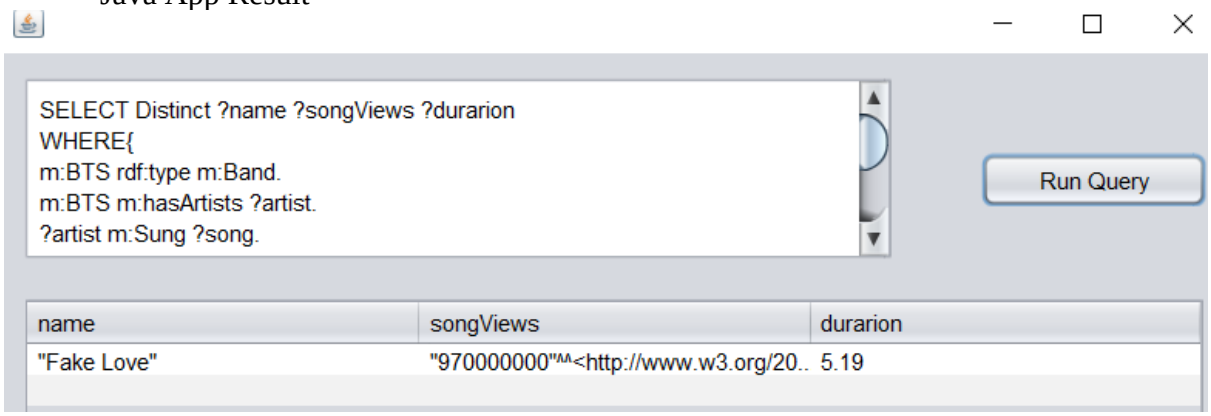
The query selects three variables to be returned in the results:

- "?name": the name of the song
- "?songViews": the number of views of the song
- "?duration": the duration of the song

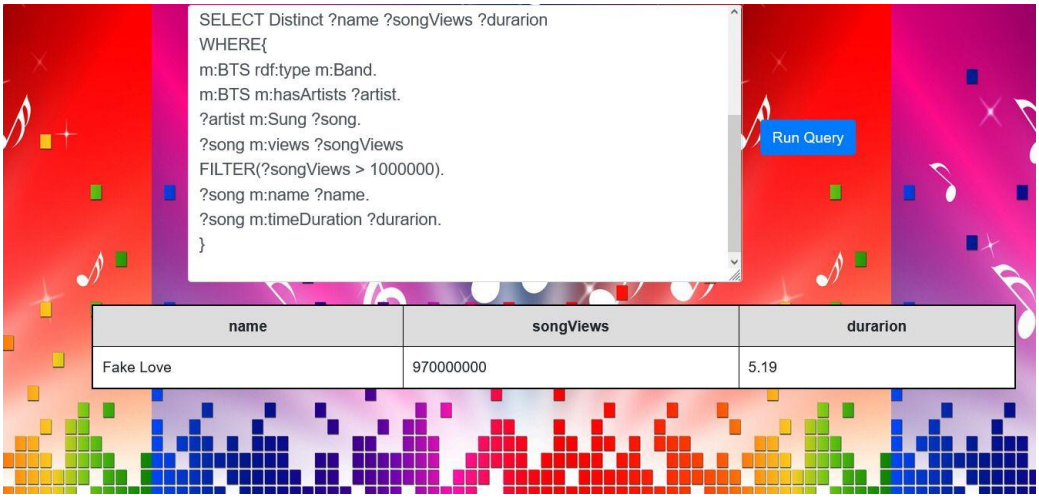
The WHERE clause specifies the conditions that the selected data must meet:

- The band BTS is an instance of the "Band" class
- The band BTS has artists
- The artist has sung a song
- The value of the variable "?songViews" is specified for "?song" and filtered to only select songs with more than one million views.
- The song has a name, and time duration

Java App Result



Web App Result



6) List the Ids , names , genre and language of BTS songs.

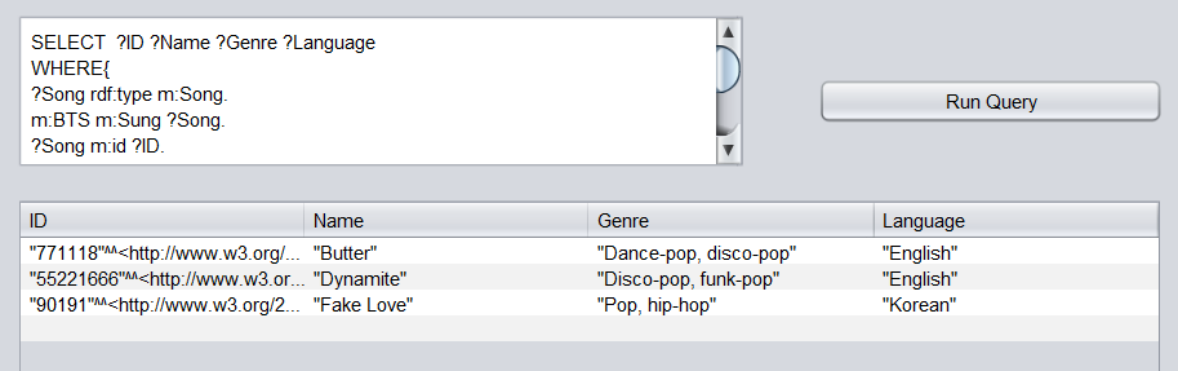
The query selects four variables to be returned in the results:

- "?ID": the id of the song
- "?Name": name of the song
- "?Genre": genre of the song
- "?Language": Language of the song

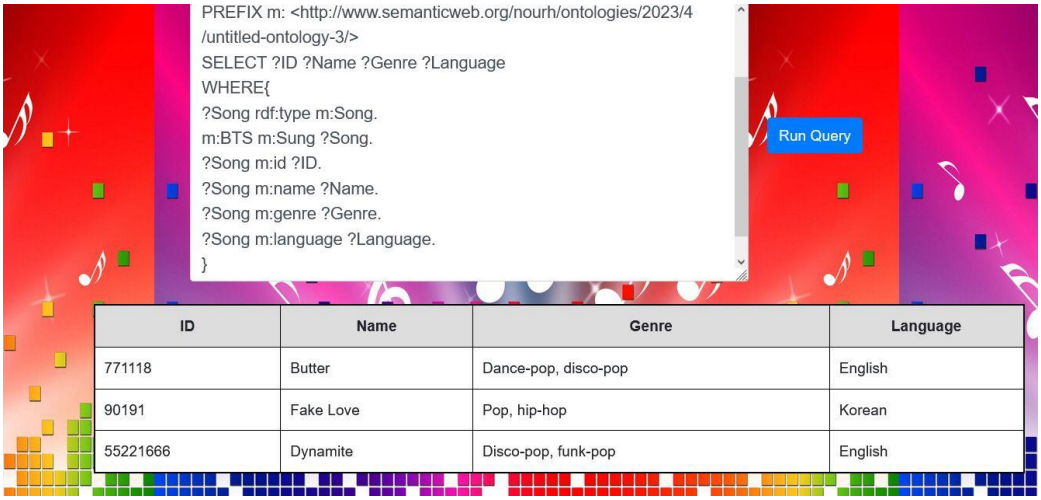
The WHERE clause specifies the conditions that the selected data must meet:

- "?Song" is an instance of the "Song" class
- The band BTS sung the song
- And the song has id.

Java App Result



Web App Result



7) List the artist names , ages and top hits of the BTS band that sung songs.

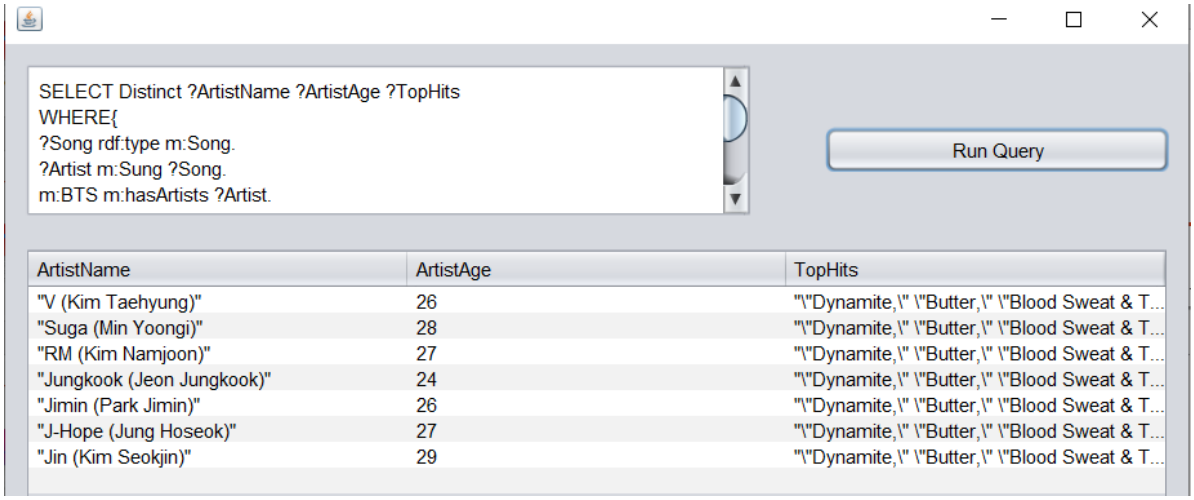
The query selects three variables to be returned in the results:

- "?ArtistName": the id of the song
- "?ArtistAge": name of the song
- "?TopHits": genre of the song

The WHERE clause specifies the conditions that the selected data must meet:

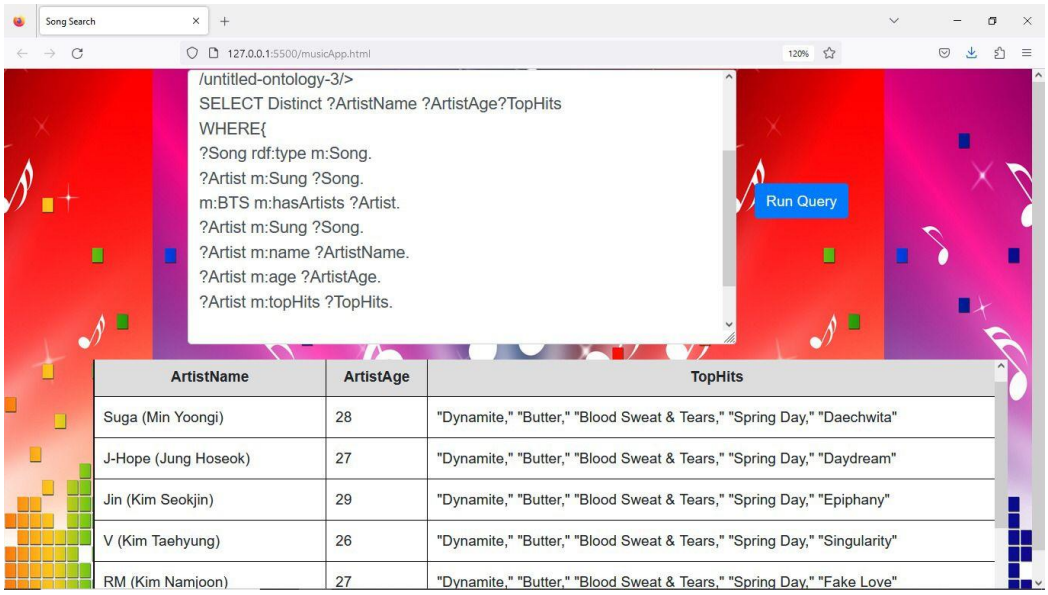
- "?Song" is an instance of the "Song" class
- "?Artist" sung the song.
- "?Artist" is a member of BTS band.

Java App Result



ArtistName	ArtistAge	TopHits
"V (Kim Taehyung)"	26	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...
"Suga (Min Yoongi)"	28	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...
"RM (Kim Namjoon)"	27	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...
"Jungkook (Jeon Jungkook)"	24	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...
"Jimin (Park Jimin)"	26	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...
"J-Hope (Jung Hoseok)"	27	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...
"Jin (Kim Seokjin)"	29	"\"Dynamite,\" \"Butter,\" \"Blood Sweat & T...

Web App Result



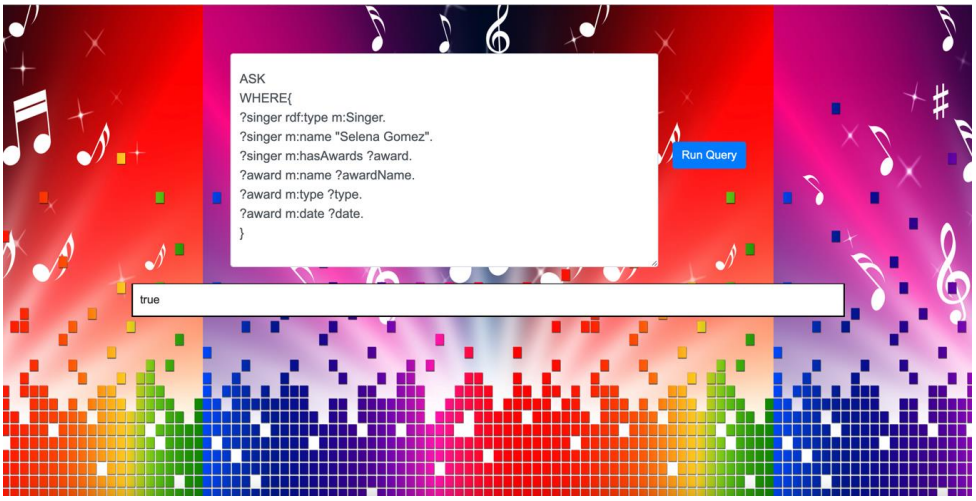
ArtistName	ArtistAge	TopHits
Suga (Min Yoongi)	28	"Dynamite," "Butter," "Blood Sweat & Tears," "Spring Day," "Daechwita"
J-Hope (Jung Hoseok)	27	"Dynamite," "Butter," "Blood Sweat & Tears," "Spring Day," "Daydream"
Jin (Kim Seokjin)	29	"Dynamite," "Butter," "Blood Sweat & Tears," "Spring Day," "Epiphany"
V (Kim Taehyung)	26	"Dynamite," "Butter," "Blood Sweat & Tears," "Spring Day," "Singularity"
RM (Kim Namjoon)	27	"Dynamite," "Butter," "Blood Sweat & Tears," "Spring Day," "Fake Love"

Ask Queries

1) Ask if there exists a singer whose name is Selena Gomez , and has award. The award has a name, type and date.

The WHERE clause specifies the conditions that the selected data must meet:

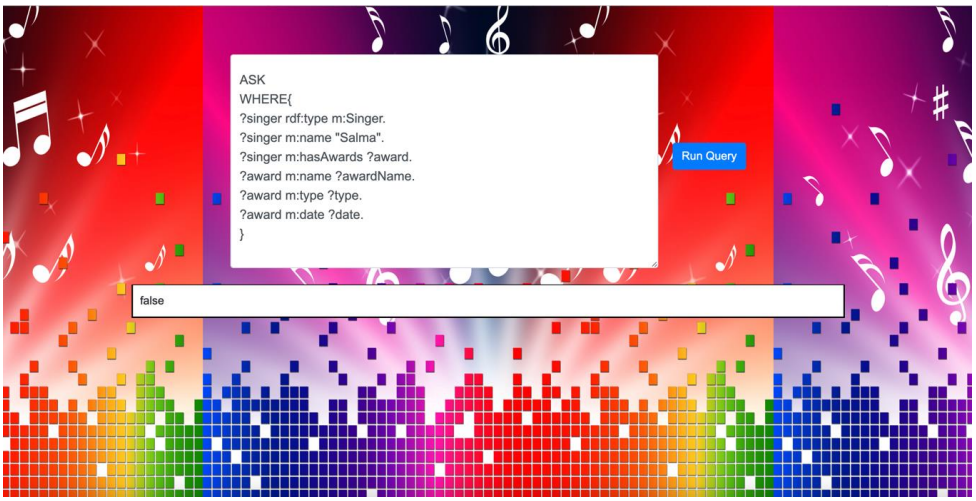
- "?singer" is an instance of the "Singer" class
- The name of the singer is "Selena Gomez"
- "?singer" has received an award
- And this award must have a Name, type, and date on which it was received



2) Ask if there exists a singer whose name is Salma , and has award. The award has a name, type and date.

The WHERE clause specifies the conditions that the selected data must meet:

- "?singer" is an instance of the "Singer" class
- The name of the singer is "Salma"
- "?singer" has received an award
- And this award must have a Name, type, and date on which it was received



Describe Query

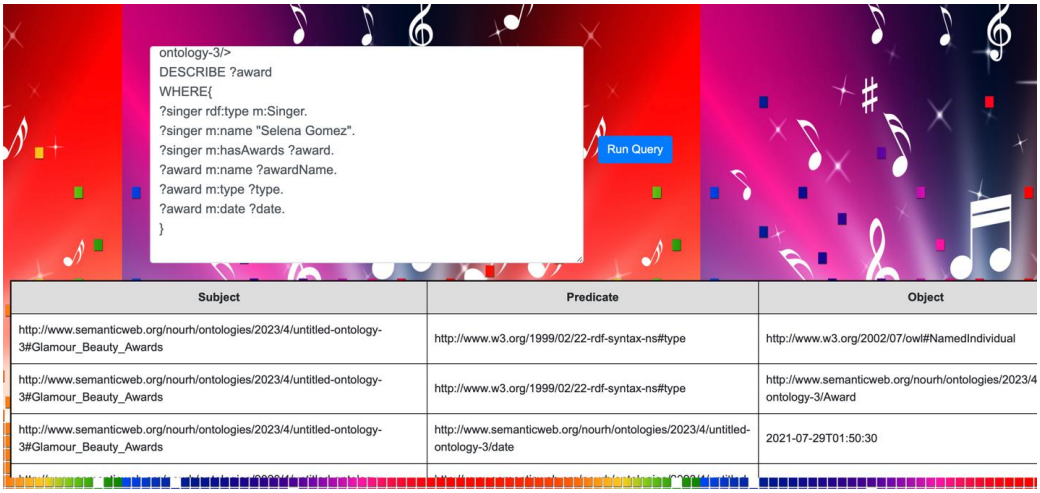
1) Describe the awards received by a singer whose name is Selena Gomez .The award has a name, type, and date.

The query Describes one variable to be returned in the results:

- "?award" : any award received by Selena Gomez.

The WHERE clause specifies the conditions that the selected data must meet:

- "?singer" is an instance of the "Singer" class
- The name of the singer is "Salma"
- "?singer" has received an award
- And this award must have a Name, type, and date on which it was received



Subject	Predicate	Object
http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3#Glamour_Beauty_Awards	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3#Glamour_Beauty_Awards	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3/Award
http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3#Glamour_Beauty_Awards	http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3/date	2021-07-29T01:50:30

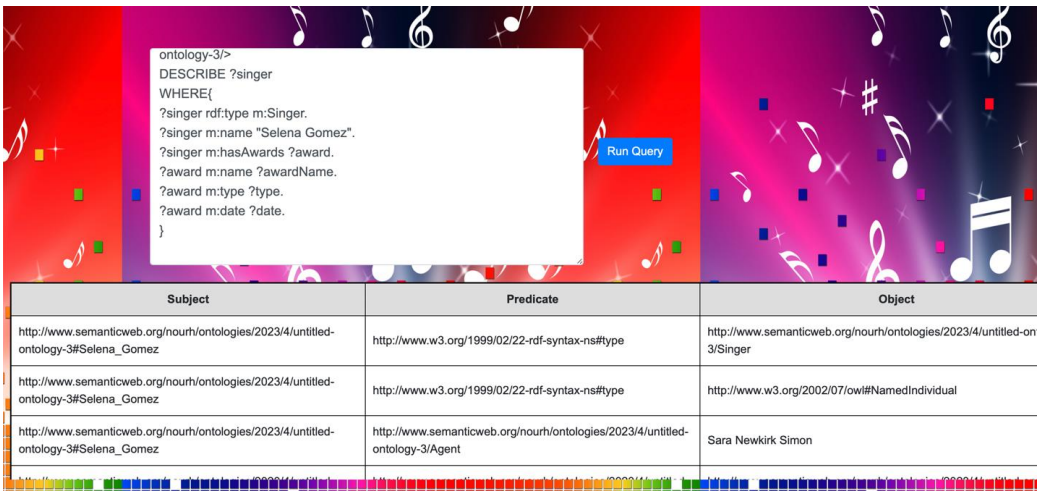
2) Describe the singer who received an award and whose name is Selena Gomez. The award has a name, type, and date.

The query Describes one variable to be returned in the results:

- "?singer" : any singer who satisfies the where clause.

The WHERE clause specifies the conditions that the selected data must meet:

- "?singer" is an instance of the "Singer" class
- The name of the singer is "Salma"
- "?singer" has received an award
- And this award must have a Name, type, and date on which it was received



Subject	Predicate	Object
http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3#Selena_Gomez	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3/Singer
http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3#Selena_Gomez	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3#Selena_Gomez	http://www.semanticweb.org/nourh/ontologies/2023/4/untitled-ontology-3/Agent	Sara Newkirk Simon