PainterPalette Knowledge Graph Project

LAI Mengyi

https://github.com/mia889766/LAI-Mengyi_TP.git

data source

For this assignment, I used the "Painter Palette" dataset published on Kaggle by Mihály Hanics, which contains biographies and metadata for thousands of painters. The original CSV was downloaded from the Kaggle dataset page (https://www.kaggle.com/datasets/mihalyhanics/painterpalette). The dataset lists each artist's name, gender, birth and death details, nationality, movements, styles, exhibition statistics, and lists of pupils, teachers, and collaborators.

Project steps

1 Data extraction and cleaning

I downloaded the CSV file and inspected the columns. Using OpenRefine, I cleaned inconsistent strings (e.g., multiple movement values separated by commas) and trimmed whitespace.

2 RDF transformation with OpenRefine

The cleaned CSV was converted to RDF using OpenRefine's RDF extension. I defined mappings that associated rows with an :Artist instance, created blank nodes for identity, birth and death events, styles and exhibition statistics, and connected them with domain-specific properties (see the mapping description below).

3 Ontology design

I designed an RDFS schema capturing key classes such as Artist, Identity, BirthEvent, DeathEvent, WikiArtStats, StyleStat, ExhibitionStat, and Relationship, as well as properties linking them (e.g., :identity, :birth, :death, :wikiart_picture, :styles, :paintingsexhibited, :relationship). Each property has an explicit domain and range and English labels. The schema was encoded in Turtle.

4 Loading data into an RDF store

I used Apache Jena Fuseki to create a dataset called PainterPalette and loaded both the ontology and the generated RDF triples. Fuseki provides a SPARQL endpoint where queries can be executed.

5 Developing SPARQL queries

Based on the specification of the web interface, I wrote a set of SPARQL queries to list artists with their metadata, find relationships (teachers, pupils, friends and coworkers), count artworks per style or movement, and link to external resources.

6 Linking to external datasets

To enrich the artists, I executed additional SPARQL queries against Wikidata. For example, I matched artists by name and retrieved their Wikidata URI.

7 Web application

I created a simple HTML application with le view: a graph view that visualises the social network of painters. The frontend uses the d3sparql library to send SPARQL queries to Fuseki and draw graphs with D3.js.

8 Repository and documentation

All source files are stored in a Git repository accessible at https://github.com/mia889766/LAI-Mengyi_TP.git The repository also contains a README explaining how to install Fuseki, load the data, and run the web application locally.

Mapping the CSV to RDF in OpenRefine

In the OpenRefine RDF mapping, I declared a base URI http://127.0.0.1:3333/ and constructed resource identifiers from the artist name (with spaces removed). Each row produced an Artist individual connected to blank nodes representing associated information:

• The :identity property links each Artist to an Identity node with simple literal properties: name, gender, citizenship, and nationality.

• The :birth and :death properties point to BirthEvent and DeathEvent nodes. These nodes store the place and year as XSD literals.

• WikiArtStats nodes hold the first and last year of artwork and the number of works available on WikiArt.

• For each style, I created a StyleStat node with :stylesname and :stylescount.

• The ExhibitionStat node stores where paintings were exhibited and a count of such exhibitions.

• The Relationship node stores lists of pupils, teachers, and friendsandcoworkers extracted from the CSV lists.

• Additional literal properties such as :movement, :Type, :occupations, :locations, and :Contemporary are attached directly to the artist. This mapping ensures that each class's domain and range align with the ontology and that data is normalised for easier SPARQL querying

RDFS/OWL schema and sample data

Below is a fragment of the ontology encoded in Turtle. The ontology declares the classes and properties along with labels, comments, domains, and ranges:

```
@prefix :      <http://127.0.0.1:3333/> .
@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#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

: a owl:Ontology ;
  rdfs:label "PainterPalette schema (RDFS/OWL + SKOS)"@en ;
  rdfs:comment "Schema for PainterPalette RDF exported from OpenRefine RDF Transform."@en .
:Artist a rdfs:Class ;
  rdfs:label "Artist"@en .
:Identity a rdfs:Class ;
  rdfs:label "Identity"@en .
:BirthEvent a rdfs:Class ;
  rdfs:label "Birth event"@en ;
  rdfs:comment "birth place + birth year."@en .
:DeathEvent a rdfs:Class ;
  rdfs:label "Death event"@en ;
  rdfs:comment "death place + death year."@en .
:WikiArtStats a rdfs:Class ;
  rdfs:label "WikiArt picture stat"@en ;
  rdfs:comment "wikiart_pictures_count + FirstYear + LastYear."@en .
:StyleStat a rdfs:Class ;
  rdfs:label "Style stat"@en ;
  rdfs:comment "a style concept + its count ."@en .
:ExhibitionStat a rdfs:Class ;
  rdfs:label "Exhibition stat"@en ;
  rdfs:comment " PaintingsExhibitedAt + PaintingsExhibitedAtCount."@en .
:Relationship a rdfs:Class ;
  rdfs:label "Relationship record"@en ;
  rdfs:comment "pupils / teachers / friends-and-coworkers lists for an artist."@en .
:identity a rdf:Property ;
  rdfs:label "identity"@en ;
  rdfs:domain :Artist ;
  rdfs:range :Identity .
:birth a rdf:Property ;
  rdfs:label "birth"@en ;
  rdfs:domain :Artist ;
  rdfs:range :BirthEvent .
:death a rdf:Property ;
  rdfs:label "death"@en ;
  rdfs:domain :Artist ;
  rdfs:range :DeathEvent .
:wikiart_picture a rdf:Property ;
  rdfs:label "wikiart_picture"@en ;
  rdfs:domain :Artist ;
  rdfs:range :WikiArtStats .
:styles a rdf:Property ;
  rdfs:label "styles"@en ;
  rdfs:domain :Artist ;
  rdfs:range :StyleStat .
:paintingsexhibited a rdf:Property ;
  rdfs:label "paintings exhibited"@en ;
  rdfs:domain :Artist ;
  rdfs:range :ExhibitionStat .
:relationship a rdf:Property ;
  rdfs:label "relationship"@en ;
  rdfs:domain :Artist ;
  rdfs:range :Relationship .
:locations a rdf:Property ;
  rdfs:label "locations"@en ;
  rdfs:domain :Artist ;
  rdfs:range xsd:string .
```

```
:Contemporary a rdf:Property ;
  rdfs:label "Contemporary"@en ;
  rdfs:domain :Artist ;
  rdfs:range xsd:boolean .
:movement a rdf:Property ;
  rdfs:label "movement"@en ;
  rdfs:domain :Artist ;
  rdfs:range xsd:string .
:Type a rdf:Property ;
  rdfs:label "Type"@en ;
  rdfs:domain :Artist ;
  rdfs:range xsd:string .
:occupations a rdf:Property ;
  rdfs:label "occupations"@en ;
  rdfs:domain :Artist ;
  rdfs:range xsd:string .
:name a rdf:Property ;
  rdfs:label "name"@en ;
  rdfs:domain :Identity ;
  rdfs:range xsd:string .
:gender a rdf:Property ;
  rdfs:label "gender"@en ;
  rdfs:domain :Identity ;
  rdfs:range xsd:string .
:citizenship a rdf:Property ;
  rdfs:label "citizenship"@en ;
  rdfs:domain :Identity ;
  rdfs:range xsd:string .
:nationality a rdf:Property ;
  rdfs:label "nationality"@en ;
  rdfs:domain :Identity ;
  rdfs:range xsd:string .
:birth_place a rdf:Property ;
  rdfs:label "birth place"@en ;
  rdfs:domain :BirthEvent ;
  rdfs:range xsd:string .
:birth_year a rdf:Property ;
  rdfs:label "birth year"@en ;
  rdfs:domain :BirthEvent ;
  rdfs:range xsd:int .
:death_place a rdf:Property ;
  rdfs:label "death place"@en ;
  rdfs:domain :DeathEvent ;
  rdfs:range xsd:string .
:death_year a rdf:Property ;
  rdfs:label "death year"@en ;
  rdfs:domain :DeathEvent ;
  rdfs:range xsd:int .
:firstyear a rdf:Property ;
  rdfs:label "first year"@en ;
  rdfs:domain :WikiArtStats ;
  rdfs:range xsd:int .
:lastyear a rdf:Property ;
  rdfs:label "last year"@en ;
  rdfs:domain :WikiArtStats ;
  rdfs:range xsd:int .
:count a rdf:Property ;
  rdfs:label "count"@en ;
  rdfs:domain :WikiArtStats ;
  rdfs:range xsd:int .
:stylesname a rdf:Property ;
  rdfs:label "style name"@en ;
  rdfs:domain :StyleStat ;
  rdfs:range xsd:string .
:stylescount a rdf:Property ;
  rdfs:label "style count"@en ;
  rdfs:domain :StyleStat ;
  rdfs:range xsd:int .
:paintingsexhibitedat a rdf:Property ;
  rdfs:label "paintings exhibited at"@en ;
  rdfs:domain :ExhibitionStat ;
  rdfs:range xsd:string .
:paintingsexhibitedatcount a rdf:Property ;
  rdfs:label "paintings exhibited at count"@en ;
  rdfs:domain :ExhibitionStat ;
  rdfs:range xsd:int .
:pupils a rdf:Property ;
  rdfs:label "pupils"@en ;
  rdfs:domain :Relationship ;
  rdfs:range xsd:string .
:teachers a rdf:Property ;
  rdfs:label "teachers"@en ;
  rdfs:domain :Relationship ;
  rdfs:range xsd:string .
:friendsandcoworkers a rdf:Property ;
  rdfs:label "friends and coworkers"@en ;
  rdfs:domain :Relationship ;
  rdfs:range xsd:string .
```

The exported RDF records follow this schema. Here is a sample instance representing the sculptor and painter Frederik Bouttats the Younger:

```
:FrederikBouttatstheYounger
        rdf:type             :Artist;
        :Type                "Sculpture" , "Painting";
        :birth               [ rdf:type      :BirthEvent;
                               :birth_place  "Antwerp";
                               :birth_year   "1620"^^xsd:int
                             ];
        :death               [ rdf:type      :DeathEvent;
                               :death_place  "Antwerp";
                               :death_year   "1676"^^xsd:int
                             ];
        :identity            [ rdf:type      :Identity;
                               :citizenship  "Southern Netherlands";
                               :gender       "male";
                               :name         "Frederik Bouttats the Younger"
                             ];
        :locations           "Antwerp";
        :occupations         "printmaker" , "painter" , "art dealer";
        :paintingsexhibited  [ rdf:type                    :ExhibitionStat;
                               :paintingsexhibitedat       "Antwerp";
                               :paintingsexhibitedatcount  "2"^^xsd:int
                             ] .
```

SPARQL queries

To draw relationships in the graph view, I combined two queries. One lists the artists who have at least one relationship and the other extracts all edges:

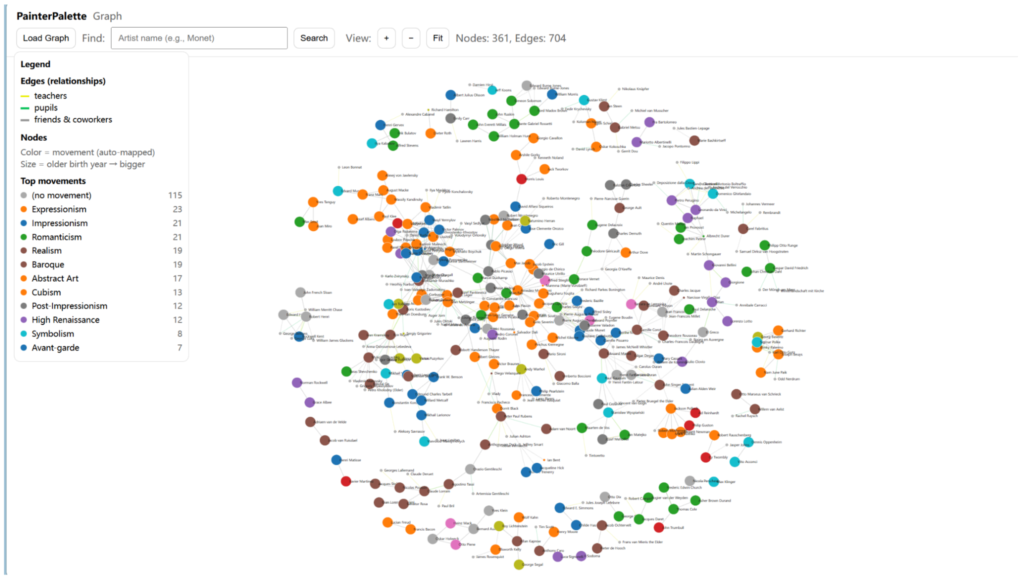
```
function qArtists(){
  return [
    sparqlPrefix(),
    "",
    "SELECT DISTINCT ?artist ?name ?year ?mov WHERE {",
    "  ?artist a :Artist ; :identity ?id .",
    "  ?id :name ?name .",
    "  OPTIONAL { ?artist :birth ?b . ?b :birth_year ?year . }",
    "  OPTIONAL { ?artist :movement ?mov }",
    "",
    "  FILTER EXISTS {",
    "    ?artist :relationship ?r .",
    "    { ?r :teachers ?t } UNION { ?r :pupils ?t } UNION { ?r :friendsandcoworkers ?t }",
    "  }",
    "}",
    "ORDER BY LCASE(STR(?name))"
  ].join("\n");
}
```

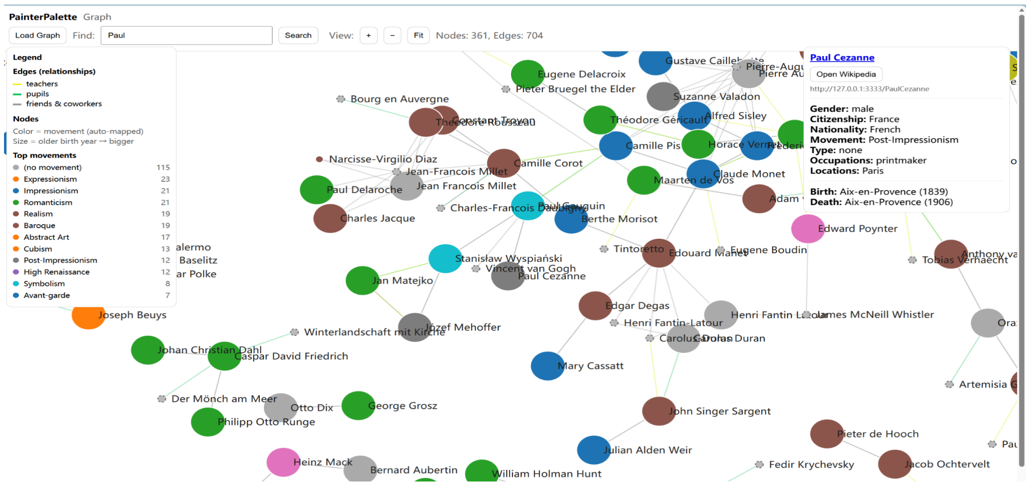
```
function qEdges(){
  return [
    sparqlPrefix(),
    "",
    "SELECT ?a1 ?name1 ?year1 ?mov1 ?relType ?targetName WHERE {",
    "  ?a1 a :Artist ; :identity ?id1 .",
    "  ?id1 :name ?name1 .",
    "  OPTIONAL { ?a1 :birth ?b1 . ?b1 :birth_year ?year1 . }",
    "  OPTIONAL { ?a1 :movement ?mov1 }",
    "",
    "  ?a1 :relationship ?r .",
    "  { ?r :teachers ?targetName . BIND(\"teachers\" AS ?relType) }",
    "  UNION",
    "  { ?r :pupils ?targetName . BIND(\"pupils\" AS ?relType) }",
    "  UNION",
    "  { ?r :friendsandcoworkers ?targetName . BIND(\"friends\" AS ?relType) }",
    "}"
  ].join("\n");
}
```

There is also a query function that displays detailed information about the artist.

```
function qDetails(artistIRI){
  return [
    "PREFIX : <" + BASE + ">",
    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>",
    "SELECT ?name ?gender ?citizenship ?nationality ?movement ?type ?occupations ?locations",
    "       ?birthPlace ?birthYear ?deathPlace ?deathYear",
    "WHERE {",
    "  BIND(<" + artistIRI + "> AS ?artist)",
    "  ?artist a :Artist ; :identity ?id .",
    "  ?id :name ?name .",
    "  OPTIONAL { ?id :gender ?gender }",
    "  OPTIONAL { ?id :citizenship ?citizenship }",
    "  OPTIONAL { ?id :nationality ?nationality }",
    "  OPTIONAL { ?artist :movement ?movement }",
    "  OPTIONAL { ?artist :Type ?type }",
    "  OPTIONAL { ?artist :occupations ?occupations }",
    "  OPTIONAL { ?artist :locations ?locations }",
    "  OPTIONAL { ?artist :birth ?b .",
    "            OPTIONAL { ?b :birth_place ?birthPlace }",
    "            OPTIONAL { ?b :birth_year ?birthYear } }",
    "  OPTIONAL { ?artist :death ?d .",
    "            OPTIONAL { ?d :death_place ?deathPlace }",
    "            OPTIONAL { ?d :death_year ?deathYear } }",
    "}",
    "LIMIT 1"
  ].join("\n");
}
```

Application screenshots





Paul Cezanne

http://127.0.0.1:3333/PaulCezanne

Gender: male
Citizenship: France
Nationality: French
Movement: Post-Impressionism
Type: none
Occupations: printmaker
Locations: Paris

Birth: Aix-en-Provence (1839)
Death: Aix-en-Provence (1906)

Code produced by LLM

Using OpenRefine, I cleaned the data . The cleaned CSV was converted to RDF using OpenRefine's RDF extension.RDFS schema and SPARQL queries used in the application were written manually by myself to meet the requirements. For the visualization code I used the d3sparql example (http://biohackathon.org/d3sparql/) as a starting point. I asked the LLM to modify this code to create the framework for the page (`paintergraph.html`), including helpers to send SPARQL queries via fetch, handle user input and build a force-directed graph with D3.js. I then improved and beautified the generated code myself: different relationships (teachers, pupils and friends) were rendered with distinct edge colours, nodes were coloured based on the artists' movements, and UI elements such as the legend and search controls were polished. Thus, while the LLM helped scaffold the initial HTML file, the final ontology, SPARQLqueries and visual refinements were the result of manual work.