Leveraging Semantic Web for Tourism Data

Submission by Group-15

Ankit Vutukuri
MS, Software Engineering
Arizona State University
Tempe, USA
avutuku1@asu.edu

Harmish Ganatra
MS, Software Engineering
Arizona State University
Tempe, USA
hganatra@asu.edu

Jubanjan Dhar
MS, Software Engineering
Arizona State University
Tempe, USA
jdhar1@asu.edu

Sudhanva Hebbale
MS, Software Engineering
Arizona State University
Tempe, USA
shebbale@asu.edu

***Abstract*—Tourism data on the web is collected from different sources that are not connected to one another. As a result, integrating these sources becomes quite difficult. Rather than connecting them using keywords, connecting similar properties makes it easy to model different relationships among them. Linked Open Data allows us to create a structure that defines the relationship between them. This paper describes a method to generate an ontology for the tourism domain by transforming relational data to Linked Open Data using different semantic web technologies. It also showcases an application that leverages this ontology to display places of interests for a city on a map.**

***Index Terms*—Web Semantics, Ontology, SPARQL**

I. Introduction

The tourism industry represents an important source of income for many countries across the globe. The domain is extremely rich in data but there are quite a few hiccups in producing meaningful results from this data. The current world wide web consists of data that is present in human readable form, mostly as textual data from multiple sources. In order to automate the task of retrieving meaning information easy, semantic web technologies can be used. The brains behind the creation of the semantic web is Tim Berners Lee, who defines the semantic web as follows:

"The Semantic Web" (Berners-Lee et al. [1]), that says "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

Web pages on the current web are not really structured with respect to the content, but the semantic web aims to solve this. The Semantic Web provides a clear meaning for content on web pages so as to provide a way for intelligent agents to execute tasks on the content to provide important information. So, the entire semantic web is maintained in the form of Linked Open Data. The method that the semantic web uses to provide structure is with the help of semantic web languages namely eXtensible Markup Language (XML)[1] and the Resource Description Framework (RDF)[2]. XML [2] is a markup language that assists in providing structure to documents in the form of tags. However, the downside of using XML is that it doesn't really provide context about the data being stored, which could be solved with the help of RDF [2]. RDF provides a structure to a web page with the help of XML, but the way this structure could be used to provide context about the content is with the help of a set of triples. A triple is a three structured tag that encodes data in the form of subjects, predicates and objects.

[1]https://www.w3.org/XML
[2]https://www.w3.org/RDF/

A. Linked Open Data

Linked Open Data realizes the vision of having globally accessible structured data on the internet. If data can be connected to other data, it will always have more value. Based on this principle, LOD provides an open environment for data to be created, connected and consumed on the internet [3]. In LOD, each entity (objects and relationships) is identified with a URI (Unique Resource Identifier) that can be accessed by anyone in the world [3]. For LOD to work, it needs to be strongly structured by using some recognized standards. This structure can be achieved by means of an ontology which allows information to be grouped together.

B. Ontology

In Semantic Web, Ontology [4] or Vocabulary can be defined as terms or relationships that can be used to group or explain a particular field or domain. They are specifically useful when there is any kind of ambiguity present to define the relationship between different fields or domains. OWL (Web Ontology Language)[5] can be defined as a family of languages which can be used to describe the complex relationships between different things or groups of things. OWL would help to make the knowledge consistent, which would make it possible to extract knowledge from the data programmatically [3].

C. SPARQL

SPARQL [4] is a standard query language for Linked Open Data (LOD) which enables users to query information from database repositories which can be mapped to a RDF. A typical SPARQL query is based on matching graph patterns in which each element (subject, predicate, object) can be represented by a variable, an exact match of this on the graph is needed

[3]https://www.w3.org/egov/wiki/LinkedOpenData/
[4]https://www.w3.org/standards/semanticweb/ontology
[5]https://www.w3.org/OWL/

to fulfill the pattern. SPARQL has 4 types of queries namely: ASK, SELECT, CONSTRUCT and DESCRIBE.

Our approach to solve the problem is a multi step process that involves making use of semi-structured data from the TourPedia [5] dataset. The first step is to process the data and filter out unimportant information. Next, we create an extensive vocabulary for the tourism domain. Using this, we will transform our dataset to RDF triples and publish them to a database hosted on a server. The final step involves developing an end-user application to display meaningful results by querying the linked data using SPARQL.

II. Problem Definition

When tourists need information about a place, they need to visit multiple websites/ apps. For instance, if a person is trying to get information about a place of attraction, he/she may need to visit one website for the details and another for the location and another for images. Sometimes, the information provided may not be desirable and its validity could also be questionable. This could be a cumbersome task and could lead to an unpleasant experience for the tourist.

III. Related Works

Creating an extensive ontology to house entities and their relationships is an integral part of previous works in the field of Semantic Web. As this paper deals with doing something along the same lines, we have drawn inspiration from works across domains such as tourism and architecture. Also, previous works making use of pre-existing datasets and vocabularies such as Schema.org, DBPedia, Dublin Core, SKOS, RDA-group2 have been reviewed to understand how to leverage built in datasets and vocabularies.

Soualah-Alila et.al [6] looks into modeling a comprehensive ontology from existing tourism data sources. Also, the paper looks into merging Schema.org with a TIF(Tour In France) ontology to mitigate the problem of unreliability and readability of tourism based data. The main motivation behind the work was that different tourism based models exist that focus on different areas of tourism but none of them are comprehensive or deal with all the needs of the tourism domain. The methodology utilized to perform merging was using OWL relations to integrate similar components from the Schema.org and the TIF dataset and converting them into an entity called Information Object(IO). This closely resembles the work of work we intended to do in this paper in terms of combining different pre-existing datasets by merging similar components to build a one-stop comprehensive ontology. However, our work in this paper is more extensive as the entire ontology will be used to serve user queries by displaying relevant information in the form of maps and intuitive tables.

García et.al [7] describes a method for publishing Linked Open Data in the tourism domain and is mainly focused on achieving this for small and medium sized DMO (Destination Management Organizations). They also mention the lack of existing integrated LOD transformation and publication methodologies in the tourism domain. The methodology proposed in the paper consists of three steps : pre-processing, triplification and publication. This has been implemented using open source tools such as Protege, LOD Refine and SPARQL. Drawing inspiration from this, our paper aims to make use of a similar approach, involving pre-processing and triplification of relational data and make use of open source technologies for the process. Another aspect of similarity with the above paper is the creation of an end-user application which will leverage the published linked open data to generate meaningful results.

Boer et.al [8] describes an end-to-end application portraying comprehensive information about the Amsterdam museum. The dataset comprises 70000 objects along with its relationships to each object in the museum. The data is constructed using pre-existing data models such as the Europeana Data Model, utilizing Dublin Core, SKOS, RDA-group2 elements and the OAI-ORE model. The vocabularies are extended from the state-of-art vocabularies available in the web such as GeoNames and DBpedia. The data in the model is again stored in the form of RDF triples and modeled using the Web Ontology Language. The end result showcases a detailed web application that shows information about each object and its associativity with other objects in the museum. The application also allows for user queries to retrieve important information about objects in the museum. This work is analogous to the work we intend to do in terms of making use of pre-existing datasets and vocabularies to serve the specific needs of our application.

Cresci et.al [9] proposed an application called Tourpedia which they deem to be the DBpedia for tourism. It accounts for more than half a million places and all are divided into 4 categories namely: accommodations, restaurants, points of interest and attractions. They also suggest that Tourpedia can have multiple applications like it can be used as name entity disambiguation in tourist domain or to retrieve most widely favored places of interests etc. The architecture consists of the Data Extraction, NE (Named Entity) Repository, Web API, Web application and Linked Data. The data extraction module consists of ad-hoc scrapers which are used to pull data from social media and booking sites. The NE repository consists of data about Names and Reviews about places specifically to the tourism domain. The web application allows for a convenient way for users to browse through all the information about places and their respective reviews which are translated to a rating score which is also reflected in the page. Additionally they also provide a SPARQL(through a D2R server) and Web API endpoint for other developers should they choose to integrate this on their projects.

Mahmud et al. [10] presents an approach where legacy data in the form of CSV could be converted into RDF data with Web of Linked Data. As most of the data that we would get would be in the form of JSON, as we would be using the REST API service, we would require a standard technique for its conversion to RDF triples. This paper provided us with one such algorithm which involved: 1) Parsing the CSV file 2) Complementing the CSV Data with the metadata 3) Converting them to the required RDF triples

IV. Approach and High-level System Design

A. Data set

We have used three different datasets for getting the data. As we would be creating an application based on the idea of smart tours, it was quite necessary for us to first chose a geographical location for which we could implement our idea. We chose the city of London for showcasing the smart tours, as London is one of the most visited city in the world. [6] We chose three different datasets for London, such that it could be of the most use for our use case of smart tours.

1) London Stations Dataset: The first dataset that we chose is London Stations. This dataset would contain information for all the tube and mainline stations placed in London. [7] This information includes the location of these stations: OS-X and OS-Y, and Latitude and Longitude. It also provides us with the zones and the postcode for each of these stations. This dataset contains 652 rows representing information regarding 652 stations in London.

2) London POI Dataset: The second dataset that we chose is London POI. It contains data regarding the places of interests located within London. [8] This dataset contains 80,510 rows representing information for 80,501 places of interests in London. Each of these rows would have a field – details, which would contain a URL that would provide us with the following information for all these places:

- address – address of the place
- category – poi in our case
- id - the unique identifier of the place
- lat -Latitude
- long - Longitude
- name – Name of the place
- phone_number – National phone number associated to the place
- international_phone_number – International phone number associated to the place
- website – URL of the website associated to the place
- icon – picture associated to the place
- description - description of the place in the six languages of the OpeNER project
- external_urls - external URLs associated with the place. It contains the URLs of Foursquare, Facebook and Google-Places
- statistics - statistics associated to the place; they are retrieved from Foursquare and Facebook
- subCategory - The category provided by the source. It is more specific than the field category
- polarity - The opinion about the place

3) London Pubs Dataset: The third dataset that we chose is Every Pub in England. This would contain information regarding all the pubs present in England. [9] This information includes the name of the place, location of these pubs: address, Easting and northing, and Latitude and Longitude. It also provides us with the name of the local authority and the postcode for each of these pubs. This dataset contains 51,566 rows representing information regarding 51,566 Pubs in UK.

As the second and the third dataset contains massive amounts of data, we did sample the data, based on our use-case.

B. Ontology Design

Each dataset has been converted into its own ontology with a different schema. The description for each is as follows:

1) Stations : The Stations ontology is mainly spread across three main classes: TubeStation, Zone and Train. As the name indicates the TubeStation class contains information about the station while the Zone class contains information about the zone in which the station is located. The two classes are connected using an objectProperty: isInZone that links the TubeStation to the Zone. The TubeStation class contains the following data properties: Firstly, it contains dataProperty: hasName that corresponds to the name of the station. Secondly, the zone corresponding to the station area is represented by dataProperty: hasZone and dataProperty: hasPostCode. The hasPlatformCount data property represents the amount of platforms in the station. The hasWheelChairAccessible data property describes if the station has wheelchair access. The hasCovidSafetyRating and hasPopularityRating are used to represent how safe and popular the station is in the pandemic. The visualization of the ontology is shown in the following figure:

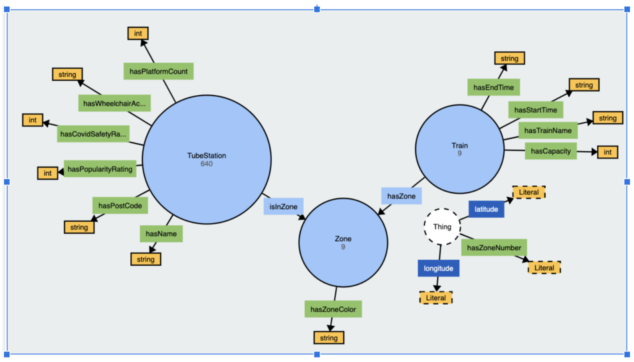


Fig. 1. Station Ontology.

2) POI: The POI ontology is mainly spread across four main classes: Subcategory, Place of Interest(POI), Reviews and Foursquare. The Subcategory class contains information about the type of place it is such as a hotel or monument etc. The review class as the name indicates has information about the reviews given by people about the place. The Statistics class contains information such as no.of people that have visited the place, the no.of likes etc. The ontology contains multiple object properties that connect two classes to each other namely: The Subcategory and the POI class are connected using the hasSubCategory objectProperty. The

[6]https://www.reuters.com/article/us-thailand-tourism/bangkok-tops-paris-london-as-worlds-most-visited-city-mastercard-idUSKCN1VP2FG

[7]https://www.doogal.co.uk/london$_stations.php$

[8]http://tour-pedia.org/about/datasets.html

[9]https://www.kaggle.com/rtatman/every-pub-in-england

POI and Statistics class are connected using the hasStatistics objectProperty while the POI class and Review class are connected using the hasReview objectProperty. The Review class comprises two more classes that encompass reviews written in two platforms namely: Foursquare and Facebook. Each of them would contain reviews written in their corresponding platform. Each class comprises its own data properties that are as follows: The Review class contains several data properties such as hasWordCount, hasText, hasSource, hasReviewPolarity and hasDate. These properties basically display information about the review written in a particular platform indicated by hasSource. The Foursquare Class contains information about the no.of times people have visited the place or no.of likes the place has received. The data properties of the Foursquare class are as follows: hasUsersCount, hasTipCount, hasPrice, hasCheckins and hasLikes. This information provides a good understanding of how well maintained the place is. The POI class itself contains information about the place of interest, which consists of the following data properties: hasAddress, hasDescription, hasId, hasPhoneNumber, hasPolarity, hasSubcategory and hasWebsite, hasCovidSafetyRating, hasPopularityRating. The visualization of the ontology is shown in the following figure:

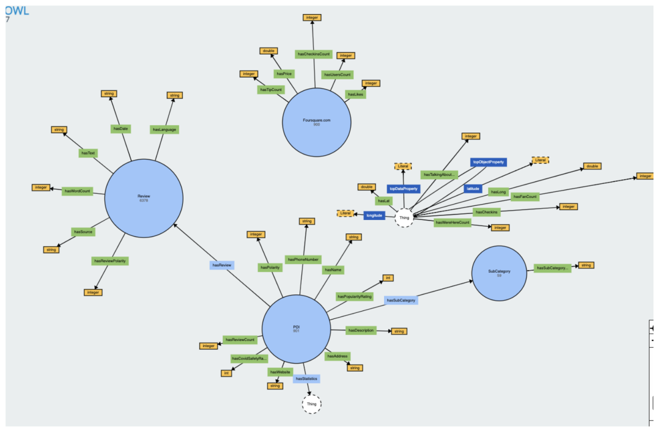


Fig. 2. POI Ontology.

3) Pubs: The pub ontology consists of four classes namely: Pubs, PopularItems, LocalAuthority and PostCodes. The LocalAuthority and Pub classes are connected using an objectProperty hasLocalAuthority. The PostCode class and the Pub class are connected using an object property hasPostCodes. The PopularItems class and the Pubs class are connected using an object property hasPopularItem. The PostCode class has just one data value namely :hasName. The LocalAuthority class also contains one data property namely: hasName. The PopularItem class contains the following properties: hasPopularItem, hasCategory and serves. The Pub class contains multiple data properties namely: hasID, hasName, hasAddress, hasRating. These properties display information about the pub itself. The hasCovidSafetyRating and hasPopularityRating are used to represent how safe and popular the station is in the pandemic. The visualization of the ontology is shown in the following figure:

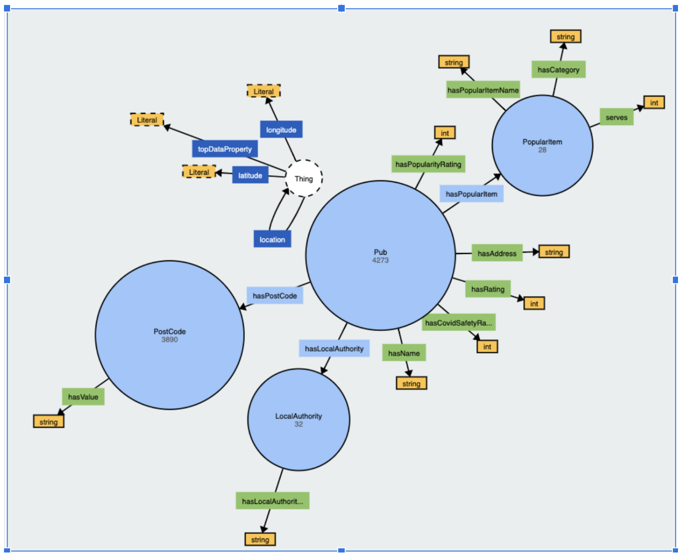


Fig. 3. Pub Ontology.

C. Data collection and processing

Data was pulled from three sources - London Station dataset, the Tourpedia POI dataset and the PUB dataset. In the case of the London station and the Pubs datasets the data was readily available in CSV format so we just had to convert them to an excel sheet using. The Toupedia website had CSV files for POI and Pubs datasets. Empirically we found that the details column in each data-point redirected to a link which had more in-depth details so we used this to get all data. Then we used these URLs to make a HTTP GET request to fetch the data which were in JSON format. We used the Pandas Library [11] in Python to create a data-frame where column names reflected the keys then subsequenty we mapped all the values to these columns. There was also some text preprocessing done to remove HTML tags and finally it was stored in an excel sheet.

D. Plugin

Once the data collection and processing step is complete, the data will be stored in well structured excel workbooks. The next step is to transform this data into RDF triples using the ontology that was created in Protege. Our approach involves making use of the Cellfie plugin[12] which comes pre installed as a part of Protege 5.x distribution. Cellfie supports the creation of OWL ontologies from spreadsheets through a flexible mapping mechanism. This is done by writing a bunch of transformation rules to define the mapping between the tabular data in the excel sheets and OWL axiom structure. These rules are written using the popular Manchester Syntax[13]

[11]https://pandas.pydata.org
[12]https://github.com/protegeproject/cellfie-plugin
[13]https://www.w3.org/TR/owl2-manchester-syntax/

that is extremely user-friendly. The rules allow the data to be accurately mapped to the ontology by creating individuals in the ontology and assigning them with the appropriate classes, sub classes and properties. For the scope of our project, we would require multiple such transformation rules for each of our three ontologies.

E. Implementation

1) Data Generation: The data generation phase primarily comprises loading the original dataset in Excel format into a format that can be easily manipulated. To perform this, Python's Pandas package [R] was used to load the original data into a dataframe. A data frame is generally thought of as an Excel document in which different forms of data manipulation can be performed such as arithmetic, logical, boolean or even table operations. As mentioned previously, three different datasets were used to construct the application namely: Place of Interests, Pubs and Stations. The following process was followed for generating the data:

The Place of Interest dataset consists of 50000 triples, in which a subset of 1000 non-empty fields data were selected in random. The entire dataset was divided into 4 main classes namely: POI, SubCategory, Reviews and Statistics. SubCategory as the name indicates consists of unique subcategories in the entire data. A mapping table between POI and SubCategory was constructed using their respective IDs. A review and POI mapping table was also constructed by incorporating one to many relationships between the IDs. Additionally, we introduce 2 new features in the POI class namely: popularityRating and covidSafetyRating. As the name indicates, they hold ratings in terms of popularity ranging from 1 to 10 and safety ranging from 1 to 5 respectively.

The Pubs dataset was extracted completely from the original dataset. Additionally, it was divided into 4 sub classes namely: Pubs, PostCodes, LocalAuthorities and PopularItems. PostCodes and LocalAuthorities help to uniquely identify the Pub and are mapped to the Pub class using their respective IDs. Popular Items as the name indicates consists of popular items in the menu for each pub. It is mapped with a one to many relationship with respect to their IDs. We also introduce popularityRating and covidSafetyRating in the Pub class. We also perform data exploding to eliminate multiple values in a particular row using data exploding techniques in Pandas.

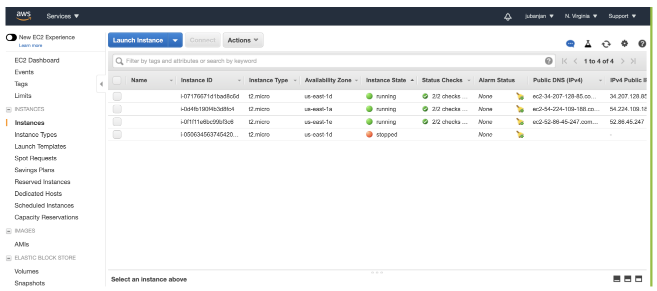
The Stations dataset was extracted completely from the original dataset. Additionally, it was divided into 3 sub classes namely: Stations, Trains, Zones. Zones are mapped to Stations using a one-to-many relationship with respect to the IDs while Trains and Zones are mapped using a one-to-one relationship. The Stations dataset also consists of 2 new features namely: IsWheelChairAccessible and Platform Count. We also introduce popularityRating and covidSafetyRating in the Stations class.

Lastly, each sub class is stored in a separate data frame and with the help of the writer package in Pandas, we can export each data frame into separate. To make the process of using the Cellfie plugin easier, we store each data frame corresponding to each dataset in different sheets under the same Excel file. All these files are exported into Excel files and utilized by the Cellfie plugin to load instances into the ontology.

2) Mapping: Mapping the data from the excel spreadsheets to the three ontologies in the project was achieved using the Cellfie plugin for Protege. The plugin makes use of transformative json rules to map the data from the appropriate rows and columns to the Classes and Properties in the ontologies. The structure of the data in our spreadsheets was such that the data corresponding to each Class in the ontology was in a separate sheet in the excel workbook. In the sheet, each row corresponds to one individual and the columns contain the values for the data properties. To map the object properties, we used a separate excel sheet containing the class to class mapping for each object property.

The Cellfie plugin rules were written in accordance with the structure described above. The rules specify attributes such as sheet name, start column, end column, etc. For each individual(row in the excel file), we can specify from which column to take the value for a given property.

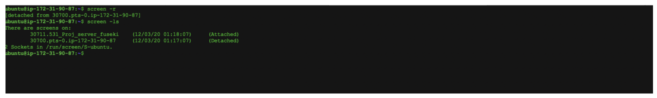
3) Configuring Fuseki Server: To set up servers on 3 different locations we use Amazon AWS EC2 instances. More specifically we used Ubuntu Server 20.04 LTS (HVM), SSD Volume Type. All the settings were kept at default except the inbound rules for the instance, where 2 new rules were added. for custom TCP, port range 3030 and source "0.0.0.0/0" and with another with the same parameters and source "::/0". After



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	i-07176671d1bad8c6d	t2.micro	us-east-1d	running	2/2 checks ...	None	ec2-34-207-128-85.co...	34.207.128.85
	i-0d4fb190f4b3d8fc4	t2.micro	us-east-1a	running	2/2 checks ...	None	ec2-54-224-109-188.co...	54.224.109.18
	i-0f1f11e6bc99bf3c6	t2.micro	us-east-1e	running	2/2 checks ...	None	ec2-52-86-45-247.com...	52.86.45.247
	i-050634563745420...	t2.micro	us-east-1d	stopped		None		-

Fig. 4. AWS Instances.

successfully starting the instance, and logging into the Ubuntu server with ssh and authorization key, few overhead tasks were carried out like installing Java and getting the Fuseki server tar file. After successfully installing them both, Linux screen was configured to independently run a session to solve the broken pipe issue. After having opened a screen session, the "shiro.ini" file is configured to allow access remotely. Following this step we start the server with the following command "java -jar fuseki-server.jar".



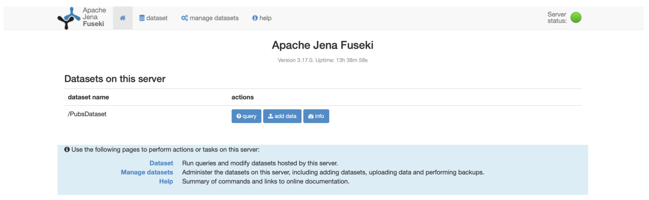
```
ubuntu@ip-172-31-90-87:~$ screen -r
[detached from 30700.pts-0.ip-172-31-90-87]
ubuntu@ip-172-31-90-87:~$ screen -ls
There are screens on:
        30711.531_Proj_server_fuseki    (12/03/20 01:18:07)     (Attached)
        30700.pts-0.ip-172-31-90-87     (12/03/20 01:17:07)     (Detached)
2 Sockets in /run/screen/S-ubuntu.
ubuntu@ip-172-31-90-87:~$
```

Fig. 5. Linux Screen

Similarly the aforementioned steps were carried out for all 3 EC2 instances. The query end points are as follows:

- Server1: Dataset1
- Server2: Dataset2
- Server3: Dataset3

To upload the datasets we head over to EC2 public domain address (eg: ec2-52-86-45-247.compute-1.amazonaws.com:3030). On the fuseki dashboard we create a new dataset and upload the owl file under that dataset. After successful upload we can start querying the server for the given dataset with the endpoints.



Apache Jena Fuseki

Version 3.17.0. Uptime: 13h 38m 58s

Datasets on this server

dataset name	actions
/PubsDataset	query add data info

Use the following pages to perform actions or tasks on this server:

Dataset	Run queries and modify datasets hosted by this server.
Manage datasets	Administer the datasets on this server, including adding datasets, uploading data and performing backups.
Help	Summary of commands and links to online documentation.

Fig. 6. Fuseki Servers.

4) SPARQL Querying: SPARQL serves as a main component to pick up relevant data corresponding to the user's input. Throughout the application, different filters have been used which are facilitated using the respective SPARQL query. In the broad sense, all the different queries used can be categorized into 3 main categories namely:

1) Queries that only fetch information about a particular place
2) Queries that search information in the entire database with respect to a keyword.
3) Queries that show nearby places with respect to the selected place.
4) Queries that attach some filters to the data portrayed on the screen

To accomplish the first type of query, a very simple SPARQL query that picks up all the field with respect to the click place in the following way: To accomplish the first type of query, a very simple SPARQL query that picks up all the field with respect to the click place in the following way:

```
PREFIX poi: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-2#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX pub: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020/
10/untitled-ontology-14#>
PREFIX station: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020/10/
untitled-ontology-8#>
PREFIX po: <http://purl.org/ontology/po/>

SELECT DISTINCT ?name ?address ?popularityRating ?covidRating ?popularItemName
WHERE {
  SERVICE <http://ec2-52-86-45-247.compute-1.amazonaws.com:3030/PubsDataset> {

  <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
  /10/untitled-ontology-14#100773> pub:hasName ?name .
  <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
  /10/untitled-ontology-14#100773> pub:hasAddress ?address .
  <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
  /10/untitled-ontology-14#100773> pub:hasPopularityRating ?popularityRating .
  <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
  /10/untitled-ontology-14#100773> pub:hasCovidSafetyRating ?covidRating .
  <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
  /10/untitled-ontology-14#100773> pub:hasPopularItem ?popularItem.
  ?popularItem pub:hasPopularItemName ?popularItemName.
  }
}
```

Listing 1. SPARQL query

The second query is used to facilitate the search bar in the application. This query looks at the keyword typed by the user and tries to match each item with the keyword. It is constructed in the following way:

```
PREFIX poi: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-2#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX pub: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-14#>
PREFIX station: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-8#>

SELECT ?name
WHERE {
        {
      ?subject rdf:type poi:POI.
      ?subject poi:hasName ?name.
        }

        UNION
        {
                SERVICE <http://ec2-52-86-45-247.compute-1.amazonaws.com:3030
                /dataset2>{
                ?subject rdf:type pub:Pub .
                ?subject pub:hasName ?name.

                }
        }
        UNION
        {
        SERVICE <http://ec2-54-224-109-188.compute-1.amazonaws.com:3030
        /StationsDataset>{
                ?subject rdf:type station:TubeStation .
                ?subject station:hasName ?name.
                }
        }
        FILTER(contains(?name, "London")).
}
LIMIT 50
```

Listing 2. SPARQL query

The third query is a query that is used to find nearby places when the user selects a particular place. These nearby places can be any in POIs or Pubs or Stations that are sorted with respect to their distance to the place. The place here inserted using the latitude and longitude. To perform this query, we make a small assumption that the earth's curvature would not cause a significant delta as we are only showing nearest places within 10 miles of the place. This assumption helps us run these advanced queries swiftly without any latency. The query is constructed as follows:

```
PREFIX poi: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-2#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX pub: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-14#>
PREFIX station: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-8#>
PREFIX st: <http://ns.inria.fr/sparql-template/>
PREFIX po: <http://purl.org/ontology/po/>


SELECT ?distance ?subject
WHERE {
        {
    ?subject rdf:type poi:POI .
    ?subject geo:lat ?lat .
    ?subject geo:long ?long .
        BIND(xsd:double(?lat) - xsd:double(51.556) AS ?first) .
        BIND(?first * ?first AS ?firstSQ) .
        BIND(xsd:double(?long) - xsd:double(-0.336) AS ?second) .
        BIND(?second * ?second AS ?secondSQ) .
        BIND(?firstSQ + ?secondSQ AS ?distance) .
  }
  UNION
  {
    SERVICE <http://ec2-52-86-45-247.compute-1.amazonaws.com:3030/PubsDataset> {
      ?subject rdf:type pub:Pub .
      ?subject geo:lat ?lat .
      ?subject geo:long ?long .
      BIND(xsd:double(?lat) - xsd:double(51.556) AS ?first) .
      BIND(?first * ?first AS ?firstSQ) .
```

```
        BIND(xsd:double(?long) - xsd:double(-0.336) AS ?second) .
        BIND(?second * ?second AS ?secondSQ) .
        BIND(?firstSQ + ?secondSQ AS ?distance) .
      }
  }
  UNION
  {
          SERVICE <http://ec2-54-224-109-188.compute-1.amazonaws.com:3030
          /StationsDataset> {
          ?subject rdf:type station:TubeStation .
    ?subject geo:lat ?lat .
    ?subject geo:long ?long .
          BIND(xsd:double(?lat) - xsd:double(51.556) AS ?first) .
          BIND(?first * ?first AS ?firstSQ) .
          BIND(xsd:double(?long) - xsd:double(-0.336) AS ?second) .
          BIND(?second * ?second AS ?secondSQ) .
          BIND(?firstSQ + ?secondSQ AS ?distance) .
          }
  }
}
ORDER BY ?distance
LIMIT 20
```

Listing 3. SPARQL query

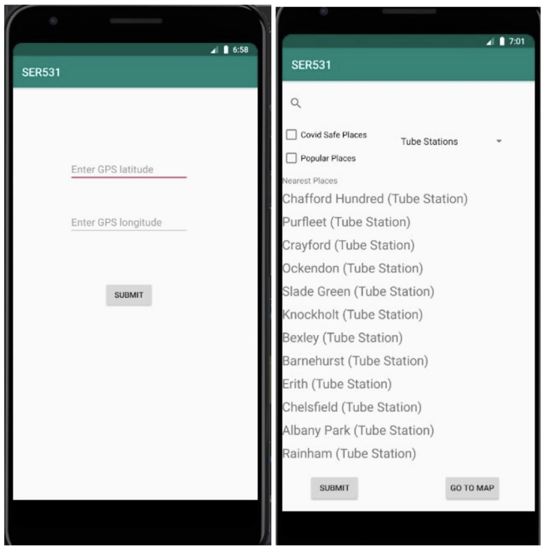
The fourth type of query is used to filter results based on two filters namely: covidSafetyRating and popularityRating. These two filters could be used in conjunction with each other and the query shows results of all places where these ratings are very high. The query is constructed in the following manner:

```
PREFIX poi: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-2#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX pub: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-14#>
PREFIX station: <http://www.semanticweb.org/sudhanvahebbale/ontologies/2020
/10/untitled-ontology-8#>
PREFIX st: <http://ns.inria.fr/sparql-template/>
PREFIX po: <http://purl.org/ontology/po/>

SELECT ?distance ?subject
WHERE {
          {
        ?subject geo:lat ?lat .
        ?subject geo:long ?long .
        BIND(xsd:double(?lat) - xsd:double(51.556) AS ?first) .
        BIND(?first * ?first AS ?firstSQ) .
        BIND(xsd:double(?long) - xsd:double(-0.336) AS ?second) .
        BIND(?second * ?second AS ?secondSQ) .
        BIND(?firstSQ + ?secondSQ AS ?distance) .
            ?subject poi:hasCovidSafetyRating ?covidRating .
            ?subject poi:hasPopularityRating ?popularRating .
        FILTER(xsd:int(?covidRating) > 4 && xsd:int(?popularRating) > 8).
  }
  UNION
  {
          SERVICE <http://ec2-52-86-45-247.compute-1.amazonaws.com:3030/PubsDataset> {
          ?subject geo:lat ?lat .
          ?subject geo:long ?long .
          BIND(xsd:double(?lat) - xsd:double(51.556) AS ?first) .
          BIND(?first * ?first AS ?firstSQ) .
          BIND(xsd:double(?long) - xsd:double(-0.336) AS ?second) .
          BIND(?second * ?second AS ?secondSQ) .
          BIND(?firstSQ + ?secondSQ AS ?distance) .
          ?subject pub:hasCovidSafetyRating ?covidRating .
              ?subject pub:hasPopularityRating ?popularRating .
          FILTER(xsd:int(?covidRating) > 4 && xsd:int(?popularRating) > 8).
      }
  }
  UNION
  {
          SERVICE <http://ec2-54-224-109-188.compute-1.amazonaws.com:3030
          /StationsDataset> {
          ?subject geo:lat ?lat .
          ?subject geo:long ?long .
          BIND(xsd:double(?lat) - xsd:double(51.556) AS ?first) .
          BIND(?first * ?first AS ?firstSQ) .
          BIND(xsd:double(?long) - xsd:double(-0.336) AS ?second) .
          BIND(?second * ?second AS ?secondSQ) .
          BIND(?firstSQ + ?secondSQ AS ?distance) .
          ?subject station:hasCovidSafetyRating ?covidRating .
              ?subject station:hasPopularityRating ?popularRating .
          FILTER(xsd:int(?covidRating) > 4 && xsd:int(?popularRating) > 8).

      }
  }
}
ORDER BY ?distance ?covidRating ?popularRating
LIMIT 50
```

Listing 4. SPARQL query

F. Front End Application / Result

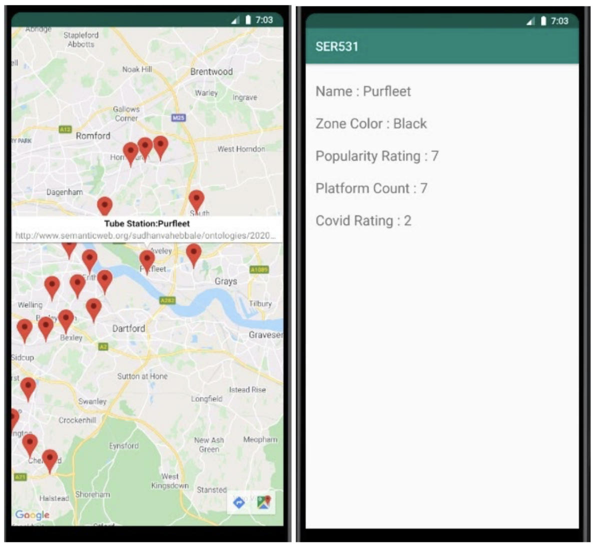
The front-end for the application is done using Java with the help of an Android application as SPARQL has a good set of tools to work with using Aquin & Nikolov [11], Liu et.al [12]. As mentioned before the user interactive application allows the user to not only view different results from the POI or Stations or Pubs datasets but also provides recommendations of nearby places with respect to the current place. The Android app starts off by requesting the coordinates of a place the user wishes to explore. This is done because the entire dataset is picked for the London city and picking the user's location could hamper results. The UI page is as shown below:



Chafford Hundred (Tube Station)
Purfleet (Tube Station)
Crayford (Tube Station)
Ockendon (Tube Station)
Slade Green (Tube Station)
Knockholt (Tube Station)
Bexley (Tube Station)
Barnehurst (Tube Station)
Erith (Tube Station)
Chelsfield (Tube Station)
Albany Park (Tube Station)
Rainham (Tube Station)

Fig. 7. Landing Page.

Once the user enters the location of the place he/she wants to explore, all nearby places including POIs, Pubs and Stations are shown in the app. The nearest place section shows all places nearby the current location. If the user wants to visualize this on a map, clicking on the Go To Map button would show the results on a map. The UI is as shown below:



Name : Purfleet
Zone Color : Black
Popularity Rating : 7
Platform Count : 7
Covid Rating : 2

Fig. 8. Filter Results.

The application also shows a couple of filters the user can play around with. The dropdown box contains 4 options: Look at all nearest places near the location, nearest POI, nearest Pubs or nearest Station near the location. The search box allows users to perform a keyword search that provides all places with the specified keyword in London. The checkboxes contain two checks namely: Covid Rating and Popularity Rating. These checkboxes can be set in conjunction and the results are sorted with respect to the covid ratings and/or popularity ratings. In the map, if any marker is clicked, details of the place are displayed in a new page shown in the above image.

1) Experimental Setup: For the development of the application, a windows system was used with Java based IntelliJ framework which has all required plugins to query the fuseki servers as well as android support. For setting up servers a Macbook was used. To talk to servers and set up a connection, SSH was used which comes by default since it's a UNIX based system. Laptop Specs:

- Laptop: Dell XPS 13
- Processor: Intel i7-8550 quad core processor,
- RAM : 16GB
- OS: Windows 10

Server Specs: Amazon AWS, EC2 instance with Linux Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type.

G. Challenges Faced

There were several challenges faced during the entire process which as are follows:

1) During the dataset generation phase, the original dataset was constructed using JSON links. So, extracting that required going through each row, performing HTTP requests, checking for empty responses and placing them into different rows. This increases the time required for generating the dataset as a loop is required over 10000 instances to fetch HTTP response.
2) The GEO SPARQL queries were difficult to implement with the constructed ontology which made it necessary to neglect Earth's curvature during the calculation of nearby places.
3) Fuseki Servers were difficult to maintain as they were constructed in a way to shut down automatically after 2 hours with a broken pipe issue. This had to be compensated using a "Screen" session.

H. Future Work

The application was an entry point to the tourism industry by showing nearby places for a particular place. There are several additions that could be made in the future which are as follows:

1) A more robust methodology could be utilized to make the nearby places more accurate using Haversine formula.
2) All the filters currently do not work in parallel but are independent of each other. So, in the future, this could be incorporated into the app to allow users more options.
3) Automatic location could be picked from the phone to allow users a more easy transition into the application.
4) The mobile application could be ported onto a web based platform to allow easier visualization.
5) Restrict access to the server as currently the dataset hosted on them could be manipulated by anyone.

I. Conclusion

Tourism data comprises a wide variety of data which needs to be extracted from various different sources. And it's difficult to model these data due to the very same reason of the data being located in different sources. Our application aims to solve this problem using semantic web technologies like OWL, RDF, and XML. In addition, it makes use of federated SPARQL queries on different attributes to pull out relevant information and thus show nearby places using GPS coordinates. This provides a one stop destination for tourists to find everything related to one place. However, data storage is quite cumbersome and could be improved using different big data technologies. The query performance could also be improved using different table partitioning and indexing methods.

References

[1] N. Shadbolt, T. Berners-Lee, W. Hall (2006). The Semantic Web RevisitedIEEE Intelligent Systems, 21(3), 96-101.

[2] Candan, K., Liu, H., Suvarna, R. (2001). Resource Description Framework: Metadata and Its Applications.SIGKDD Explorations, 3, 6-19.

[3] Khusro, S.,Jabeen, F.,Mashwani, S., Alam, I. (2014). Linked Open Data: Towards the Realization of Semantic Web-A ReviewIndian Journal of Science and Technology, 7, 745-764.

[4] Quilitz, U. (2008). Querying Distributed RDF Data Sources with SPARQL. In The Semantic Web: Research and Applications (pp. 524–538). Springer Berlin Heidelberg.

[5] Cresci, S., D'Errico, A., Gazzè, D., Lo Duca, A., Marchetti, A., Tesconi, M. (2014). Towards a DBpedia of tourism: The case of TourpediaCEUR Workshop Proceedings, 1272, 129-132.

[6] Soualah-Alila, F., Faucher, C., Bertrand, F., Coustaty, M., Doucet, A. (2015). Applying Semantic Web Technologies for Improving the Visibility of Tourism Data.

[7] Garcia, Ander Linaza, Maria Franco, Javier Juaristi, Miriam. (2015). Methodology for the Publication of Linked Open Data from Small and Medium Size DMOs. 10.1007/978-3-319-14343-914.

[8] Amsterdam Museum of Linked Open Data by B Victor et al in Journal of Semantic Web (2013).Boer, V., Wielemaker, J., Gent, J., Oosterbroek, M., Hildebrand, M., Isaac, A., Van Ossenbruggen, J., Schreiber, G. (2013). Amsterdam Museum Linked Open DataSemantic Web, 4, 237-243.

[9] Cresci, Stefano D'Errico, Andrea Gazzè, Davide Lo Duca, Angelica Marchetti, Andrea Tesconi, Maurizio. (2014). Towards a DBpedia of tourism: The case of Tourpedia. CEUR Workshop Proceedings. 1272. 129-132.

[10] Mahmud, S., Hossin, M., Jahan, H., Noori, S., Hossain, M. (2018). Csv2rdf: Generating rdf data from csv file using semantic web technologiesJournal of Theoretical and Applied Information Technology, 96, 6889-6902.

[11] M. d'Aquin, and A. Nikolov 2012. Building SPARQL-Enabled Applications with Android devices Conference Item.

[12] C. Liu, H. Wang, Y. Yu and L. Xu, "Towards efficient SPARQL query processing on RDF data," in Tsinghua Science and Technology, vol. 15, no. 6, pp. 613-622, Dec. 2010, doi: 10.1016/S1007-0214(10)70108-5.