ElasticOSM: A reimagining of OpenStreetMap’s search user interface

In this project, I implement an alternative search user interface to the one’s that's currently available for use at the OpenStreetMap website.

View live project →
View source code →

Submitted by **Arogya Koirala**
Information Organization and Retrieval
FALL 2021

Background

With nearly 7.8 million users having contributed over 7 billion data points over the last fifteen years, the OpenStreetMap project has grown to become one of the world's largest sources of open geographical data today. Not surprisingly, a number of software development efforts, both from commercial stakeholders and from members in the open-source community, have taken place with the goal of making this data more accessible to everyone.

However, despite all the progress in making information retrieval for OSM faster and more efficient, work on user interfaces that serve as a portal between the user and this data leave much to be desired.

In the sections that follow, I provide an overview of the OpenStreetMap data model, some of the challenges posed by this model to the development of search user interfaces, some potential solutions to tackle identified problems, and finally a proof of concept implementation of an alternate search user interface that aims to implement the same.

Motivation

The purpose of this project is not to "solve" geospatial search, as doing so would be too complex of a challenge to take on for a class project. Geospatial search is highly situational, and its efficacy really depends on what the user's needs are. The focus of this endeavor is to examine how we can take advantage of presently existing metadata in OSM and recent advances in technology and database systems to provide a subjectively "better" search experience for a general user.

Through this project, the author therefore aims to develop a proof-of-concept that will help contribute to the conversation around:

1. What are some of the challenges and limitations posed by OpenStreetMap's data model on the design of search user interfaces for OSM data?
2. What are some characteristics of metadata available crowdsourced geospatial information, and what are some challenges that arise when we try to organize such information?
3. How can OSM metadata be repurposed to facilitate a better search experience for users of OSM data?
4. How can recent developments in information retrieval technology support a better search experience for users of OSM data?

The author notes that the very nature of OpenStreetMap being a volunteer driven movement imposes a lot of constraints on developers of active open source projects in the OSM ecosystem. And user interface design, although highly critical to the user experience, is often prone to taking the hit during development with constrained resources. The purpose of this project is not to criticize and dismiss the work of hundreds of selfless volunteers who have brought OpenStreetMap to where it is today, but lay out guidelines for similar work in the future.

The OpenStreetMap Data Model

The following section contains a short introduction to the data model in OpenStreetMap and the process of mapping. It also introduces some terminology which will be used time and again in subsequent sections that follow.

Geographical entities in OSM

OSM uses a data structure that is made up of geographical elements called nodes, ways and relations. Nodes represent points on the earth's surfaces, defined by their longitude and latitude. These nodes can be combined together (in a specified order) to form a way. Together, nodes and ways can be combined or used separately to denote physical geographical entities such as hills, roads or forests (see image below). Sometimes, when there is a need to define connections between two geographical elements (for instance a city being inside an administrative boundary), OSM supports the use of elements called relations to make these connections explicit.

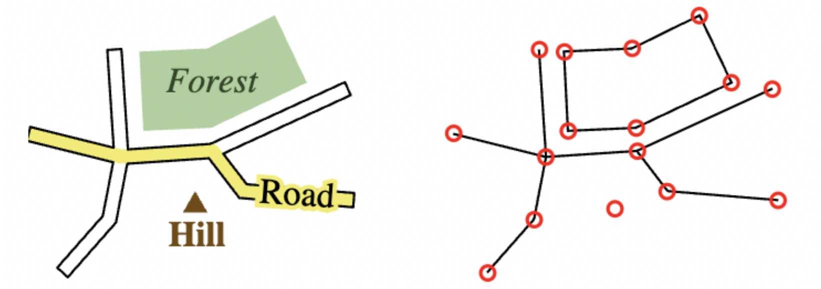


Image source: TU Eindhoven

Tags and Tagging

It is not sufficient to simply represent the map features as nodes, ways, or relations. To help describe the contextual characteristics of these geographic elements, OpenStreetMap provides users with the affordance of ***tagging***, wherein every map feature can be supplied with metadata in the form key-value pairs (e.g., amenity=hospital, opening_hours=Mo-Su 12:00-20:00). To connect back with our previous example on the above image, supplying the tags *highway=primary* to the way that represents a road in our above example, is what would help denote that the polyline represents a road. Similarly, attaching the tag *name=CA-1* to the same polyline would then denote that our road is a part of CA-1.

Mapping = feature tracing + tagging

To summarize, the act of mapping in OSM is a two-stage process that involves a) tracing out geographic elements over publicly accessible satellite imagery, and b) tagging traced elements with key-value pairs that appropriately describe properties of the element being mapped. There are a number of publicly available tools that allow users to map in OSM such as JOSM, iD editor.

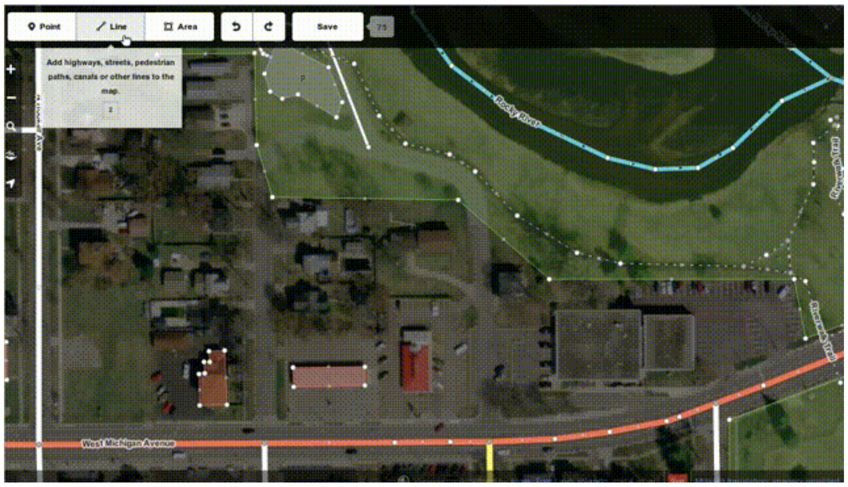

Point
Line
Area
Save
Add highways, streets, pedestrian paths, canals or other lines to the map.
Rocky River
West Michigan Avenue

Image Source: GIS Lounge

For a more detailed description of mapping, please refer to this article by TU Eindhoven

OSM's data model and the challenges it poses to the development of search user interfaces.

As we can see, there is really no upper limit in the amount of metadata that any given feature in OSM can hold. One could argue that this choice is what makes the design extremely powerful. Theoretically speaking, any geographical element in OSM can be associated with an infinite number of secondary attributes. This flexibility has allowed researchers to come up with highly specific and creative approaches to capture and represent ground information (see OpenHazardMap). Additionally, the adoption of a key-value structure has also freed OpenStreetMap from the limitations of using a relational approach for metadata organization, where all attributes have to be predetermined. This flexibility has allowed data to more accurately capture our constantly changing world - the addition of a simple *shop=shoe* tag can denote that a building changed to a shoe shop, and the modification in the *opening_hours*=* tag can allow anyone to quickly reflect this change.

However, this lack of an upper ceiling can also be problematic. A free-for-all metadata entry scheme, combined with the adoption of a crowdsourced model for data collection has introduced a high amount of heterogeneity in the data. Although there has been significant progress around coming up with standard metadata conventions for OSM over the years, arriving at a consensus over changing geographical and cultural contexts continues to remain a challenge for the OSM community. A lack of a standard convention for metadata means filtering capabilities in search user interfaces for OpenStreetMap data would be limited. This is especially true for search user interfaces that used ruled based mechanisms for generating filters (e.g. creating a filter for all values of “amenity” category) without knowing what the available values of amenity are. This has translated into lower availability of post-search interaction features (narrowing down of results through filters and faceted browsing) in popular OSM search user interfaces (Nominatim, OverPass Turbo, Photon), with most search engines limited to simply showing matches for a given query. The author also did not find any implementation of faceted navigation which could positively augment the information seeking process by supporting orienteering.

The high specificity with which geographical features are tagged is another challenge in the development of search user interfaces for OSM data. While the specificity of primary tags in OSM data (e.g., highway=residential or highway=tertiary for distinguishing roads based on their use) for entities are certainly helpful when displaying these geographical entities in a map, such a granular differentiation is not always necessary, and might even be detrimental during the purposes of search. For example, how would the user feel if their favorite bar doesn't show up when they use the keyword "bar", simply because the entity they were looking for is tagged as a "pub".

Furthermore, it also doesn’t help that for the uninitiated, OSM data can be highly technical and inaccessible, both for reading and writing. Consider the simple case of specifying a business’ operating hours, which is represented by the tag opening_hours=*, which involves relying on users entering information following a complex, and specific syntax.

ElasticOSM

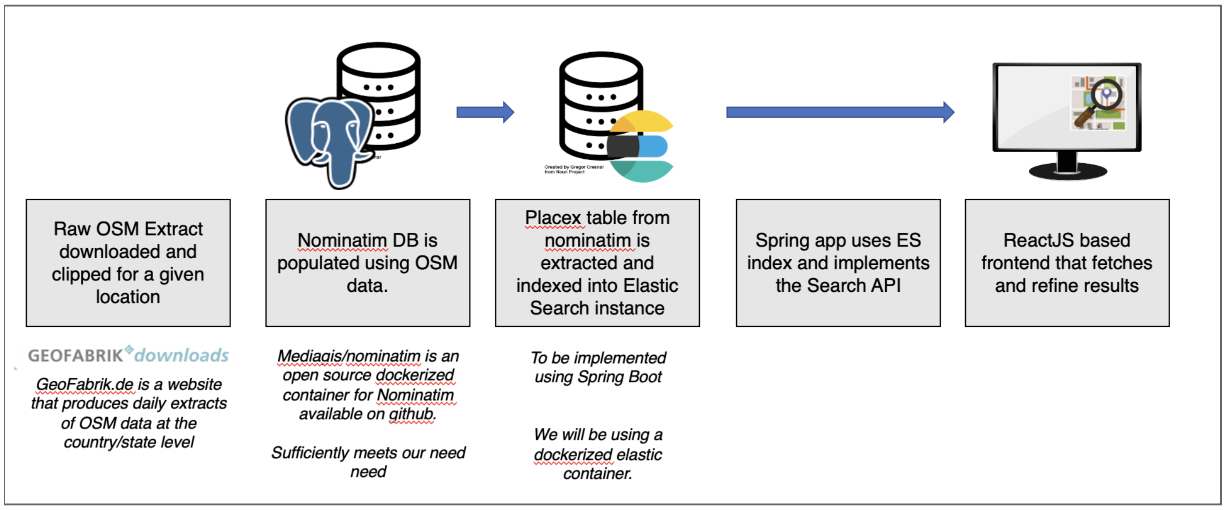
This section contains notes around the implementation of ElasticOSM, an OSM search user interface for the city and neighbouring communities of Berkeley, which seeks to tackle some of the problems identified above. It consists of two sections. First, is a detailed description of how the project was technically implemented. This is followed by a description of how different concepts of information organization and retrieval were applied in the execution project.

Technical Implementation

This section describes in detail the process taken to produce the ElasticOSM project.

High level architecture

The following table outlines the high-level architecture of the ElasticOSM project.



Raw OSM Extract downloaded and clipped for a given location

GeoFabrik.de is a website that produces daily extracts of OSM data at the country/state level

Nominatim DB is populated using OSM data.

Mediagis/nominatim is an open source dockerized container for Nominatim available on github.

Sufficiently meets our need need

Placex table from nominatim is extracted and indexed into Elastic Search instance

To be implemented using Spring Boot

We will be using a dockerized elastic container.

Spring app uses ES index and implements the Search API

ReactJS based frontend that fetches and refine results

In summary, the entire project comprises of the following five stages:

1. Raw OSM data is downloaded and loaded into an instance of Nominatim, an open source search engine that uses PostgreSQL as a backend.

2. OSM feature metadata from the placex table in the nominatim database is analysed for identifying reorganization strategy.
3. A Java application that can connect to an elasticsearch database instance and the Nominatim DB indexes all of the available data in the placex table in the Nominatim database to an index in elastic search.
4. Another java application serves data from an elastic index in the form of a REST Api.
5. A ReactJS app that contains the search user interface interacts with the application in step 4 to facilitate the search.

Stage-wise breakdown

This section provides a stage wise breakdown of all of the activities listed in the high-level overview above.

Step 1: Download OSM data

OSM Data for the city and surrounding area of Berkeley was downloaded from the BBBikes download server. The data was in a highly compressed PBF binary format.

Step 2: Populate data into Nominatim

This data was populated into a containerized instance of Nominatim, an open source postgreSQL / postGIS based search engine used in the official OpenStreetMap website. In addition to transferring contents from the raw OSM binary file into a postgreSQL database, Nominatim also provides a host of other functionalities which would serve handy in the later stages of our project. These include:

1. **Address Generation through identification of parent geographical entities:** Nominatim takes advantage of spatial capabilities afforded by PostGIS to identify nearest features to an element that can help in address formation. For instance, for an entity tagged with amenity=book_shop and name="Mike's Books", Nominatim will query nearby elements with address tags (place=neighbourhood, highway=street, place=city, postbox=*, etc) and use the information available there to generate an address for Mike's Books (Mike's Books,

Solano Avenue, Berkeley CA, 94712). The address generation logic can also be customized through a JSON file that can be supplied during build time. However, this is out of the scope of our project.

2. **Identification of primary tags for a feature:** Not all tags in OSM are equal. Primary tags are key-value pairs in a geographical element that serve to identify the nature of the element. Examples of this include amenity=hospital, shop=books, tourism=hotel for hospitals, book shops and hotels respectively. For a given geographic entity, nominatim automatically goes through all of its tags to identify the primary key-value pair. This information is then stored for a feature as its "class" and "type".
3. **Calculation of feature centroids:** Nominatim takes advantage of postGIS to generate centroids for all ways in the dataset. This is useful when results of our search have to be represented for viewing in a map based interface.

For the purposes of this project, I used a containerized open-source implementation of Nominatim provided by mediagis. This implementation was relevant because it allows easily specifying an external dataset for populating the Nominatim instance through environment variables.

Relevant output:
The ***placex*** table in the Nominatim postgres database contains all of the raw information we need to build the search functionality. The following image shows a description of the table's fields.

Step 3: Analysis of class and type data from the placex table
As discussed in section 4, given the high specificity of OSM data, a remapping and reorganization of primary tags would facilitate more user-friendly interaction over the interface. To support this, a table that contains frequencies of different primary tags was generated and exported from the *placex* table in Nominatim for analysis. The following query was used in this process:

```
SELECT
   class, type, count(1) as total,
   count(name) as total_with_name,
   count(1) - count(name) as total_without_name,
   100.0 * count(name) / count(1) as percent_with_name
FROM
   placex group by 1,2 order by 3 desc, 6;
```

This yielded a total of 858 unique class + type combinations. Recognizing that coming up with unique labels for all 858 key-value pairs would be both time consuming and *not* all-encompassing, we focused only on relabeling the top 56 categories, which represented 95% of all the data. The following table shows examples of labels resulting from the exercise.

class	type	label
amenity	restaurant	Restaurant
highway	motorway	Road
amenity	bench	Bench
leisure	swimming_pool	Swimming Pool
highway	service	Road
man_made	pier	Pier

It must be pointed out that there are some limitations of this approach, which need to be dealt with:

1. The process is non-exhaustive, although we've managed to identify labels for geographical elements that occur most frequently, the heterogeneity of OSM data means that it is impossible to use this approach in accounting for all class-type combinations, specially in larger datasets.
2. Frequency is not always a good measure of importance, especially when it comes to geographical entities. There may be less museums in a city, but museums are important.

This was dealt with at a later stage through iterative testing of the client UI and addition of new labels.

Relevant output:

The file categories.csv, which mapped primary tags into their relevant newly identified labels.

Step 4: Transfer placex data into elastic search

A containerized single-node instance of elasticsearch is made available for use. The Spring Boot application pg2es is launched which is responsible for:

1. Preparing an elasticsearch index for storing place data
2. Specifying analysers and filters different fields in the place index (see section 4 for details around this)
3. Connecting to postgres database in the containerized Nominatim instance.
4. Transferring contents from Nominatim DB (postgres) into ElasticSearch

Step 5: Creating the ElasticOSM Search API

The Spring Boot application api uses the Spring Boot framework to create a simple REST API that connects with the elasticsearch container and retrieves search results. Specifications of the API are provided below.

GET **/search**

Following are the parameters accepted by the endpoint

- **q** (required): the query string against which to search the database
- **top** (required): Latitude corresponding to the north-east end of the map view.
- **left** (required): Longitude corresponding to the north-east end of the map view.
- **bottom** (required): Latitude corresponding to the south-west end of the map view.
- **right** (required): Longitude corresponding to the south-west end of the map view.

Step 6: Bringing it all together through the ElasticOSM client
Finally, **client** a single page web application built using React JS provides a mechanism for users to interact with the search functionality. This is the final search user interface for the project. Please keep in mind that the application requires location permissions to function.

Key IR concepts that were used in the implementation of this project.

This section discusses how the project applies relevant concepts and principles of information organization and retrieval to tackle some of the problems discussed in section 4.

Metadata reorganization: relabeling of primary tags to reduce differentiation of similar geographic entities

In OSM data, every geographical feature is associated with a primary tag, i.e., a key-value pair that serves to identify the nature of the feature, e.g. shop=books for book shops, amenity=hospital for hospitals, and tourism=hotel for hotels. Given the heterogeneity and high granularity of, and the technically complex way in which OSM metadata is specified, it became clear that engaging in a manual process of labeling commonly occurring primary tags into more simple, easy to read labels would significantly improve the search experience. Additionally, we realized in hindsight that the activity was also helpful in making this data amenable to creating user-friendly facets and support the generation of synonyms rings. The following table illustrates the outcome of the process, with the column **label** being end result for this exercise.

class	type	label
amenity	restaurant	Restaurant
highway	motorway	Road
amenity	bench	Bench
leisure	swimming_pool	Swimming Pool
highway	service	Road
man_made	pier	Pier

Faceted browsing: identification of relevant tags for faceting

By making use of TagInfo, a popular tool to explore metadata in OSM, and wiki pages of commonly used OSM tags (see example), a few key OSM tags were identified as being suitable for supporting faceted browsing of search results. These include:

1. opening_hours (when): Opening hours of a commercial facility
2. wheelchair (what): Is the facility accessible via wheelchair?
3. internet_access (what): Does the facility have internet access?
4. delivery (what): Does the facility deliver?
5. drive_through (how): Does the facility allow drive-throughs?
6. payment (what): What sort of payment options can I use in the facility (cash, card, etc.)
7. takeaway (how): Does the facility provide takeaway?
8. cuisine (what): What kind of cuisine does the facility serve?
9. diet (what): What kind of meals are available (vegan, vegetarian) in this establishment?
10. centroid (where): The latitude/longitude of the facility, this can be used to determine how far a location is from the user's position.

It is important to note that some metadata (like centroid) are more universal, i.e., applicable to a wide range of geographical elements as opposed to others (like cuisine).

The 'opening_hours' tag

It was particularly challenging to generate facets using the opening_hours tag, as the values for these tags are not directly amenable to the formation of facets. However, using the open source javascript library **opening_hours**, we were able to repurpose information in these tags to create faces based on whether a facility is "open", "closed" (or "unknown")

Geographically bounded querying

The use of elasticsearch in the backend enabled support for geo bounding box queries. By returning only results based on where the user positions their maps, it helps retrieve results that are more contextual and relevant to their needs.

Weights and querying across multiple attributes

Elastic’s ability to support boolean querying and assigning different weights to independent results meant that the interface was not only able to match keywords to names and addresses (“Walmart”, “Target”), but also to category labels (“shops”, “department stores”)

Tokenization strategies

The following tokenization strategies were also employed to improve the performance of search functionality

Generation of edge-ngrams to support autocomplete functionality.

Edge-ngram filters are variants of n-gram filters that allow elasticsearch to store ordered combinations of letters that make up a given word (wash = w, wa, was, wash). Storing edge-ngrams of place names during the indexing process enabled the introduction of autocomplete functionality during query time.

Porter-Stemmer to capture keyword variations

A stemming filter was also used to capture variations in category labels (bars vs. bar).

Synonym rings to return thematically related search results

The use of elasticsearch in the backend made it possible to use synonym rings through the use of a synonym token filter. This provided a way to make the search user interface return also “bars” when the user queried for “pubs”. The metadata labeling exercise was very helpful in identifying relevant synonyms for use with the filter.

Acknowledgments

This project would not have been possible without the efforts of people contributing to the following open source projects.

- osm-search/Nominatim is what makes the initial indexing of OSM data into a postgres database possible.
- mediagis/nominatim-docker is a docker container for Nominatim.
- BBBike's OSM data extract for Berkeley
- tonytw1/nominatim-ac and their work on indexing the Nominatim DB to Elastic is the starting point for the code in the `pg2es` project.
- @turf/turf is what is used for the generation of geo-distance facets
- elastic/elasticsearch-docker the containerized version of Elastic search which hosts all of our data
- opening_hours is what is used for the generation of opening hours facets
- SteveTLN/https-portal is what seamlessly enables SSL encryption in the client website
- reduxjs/cra-template-redux the boilerplate code for our client app.

Finally, the author would like to thank all of the volunteer data contributors to the OpenStreetMap project, without whose sustained contributions, none of this would have been possible in the first place.