Project:INTERMINE DATA MODELLER TOOL

Name: *Sanat Mishra*
Email: sanatmishra1@hotmail.com
InterMine Discord username: SM
GitHub profile: Sanat-Mishra

Overview

Intermine is a data warehouse comprising over thirty different mines which together contain a variety of information about different model organisms with a core InterMine data model based on based on sequence ontology (SO). This makes the data in Intermine take shape of a directed acyclic graph, which develops into a highly interrelated data model, which can make it complex for those exploring Intermine.

A data modeller tool will thus help in understanding the relationships among each component of the data model and its hierarchy. It can graphically display the relationships among the various data types in their respective mines and act as a search tool to query through the data model and return the model as it is.

Intermine administrators and developers will find this tool helpful to quickly visualise the data model and compare it across various mines. It will save time and energy, focussing on the data itself rather than engaging in how the data exists and complying with its protocols.

PROBLEM STATEMENT:

- The Intermine data model for a particular mine is an amalgamation of three different layers of XML data. Each of these contribute certain data types to a mine. Each data type, or class in the model further contains information, classified as **attribute, reference or collection**. References and collections explain how one data type references to another object (Data type) in the database. While the attribute contains the nature of data such as integer, string, Boolean value and so on.
- Additionally, the data types exhibit inheritance, that is they can extend a class, which in turn extends another one and so on and so forth. However, there are standalone ones too, that is these do not exhibit any inheritance!
- Hence we see that the data types are interrelated in more than one way and it is necessary to understand their relationships to build newer data types or export them to other mines. Considering that there are more than 30 mines, it is very cumbersome to manually understand the various ways in which a data type may exist in a particular mine.
- One would want to know the relations that exist between various data objects **to understand which side of the relation to populate in the mines**, as those from the 'reference' side saves time and effort as against that from the 'collection' side. Additionally, each referenced object should be included for **its attributes and references to function correctly.**
- This justifies the need for a data model search tool, one that renders the model graphically for visual clarity as well as returns the data model as it is.
- The former can be conceived as a **network graph**, which is most suited for this purpose due to the hierarchical nature of the model. I shall be using the **NetworkX** module to solve this problem. So, the output shall be a network graph consisting of nodes as the related data types and edges between them to indicate inheritance. Rendering it with Bokeh will make it interactive and more convenient to the user.
- To solve the latter problem, I choose to use a full text search library, **Elasticsearch**. It is an incredible library supporting fuzzy searches, phrase searches, partial searches (automatically graded according to relevance) among a host of other features. The data types and values need to be only indexed beforehand, for it to render results.

IMPLEMENTATION PLAN-

Step 1:

Objective: Scraping data from each mine's website and ingesting it into Elasticsearch.

Firstly, the model data is scraped from each mine's website in XML format. The website details are accessed through the 'requests' library and for web scraping we use the etree module. The data corresponding to each class tag, along with the attribute, extends, collection and reference tag is then added to a new dictionary. We then assemble all data and prepare the payload for Elasticsearch. Elasticsearch easily recognizes data in JSON format, hence the need for this effort.

Here I have extracted the data 'attributes' contains. It can be **easily extended for other headings (collections, classes, references)-**

```
import requests
from elasticsearch import Elasticsearch
import requests
import xml.etree.ElementTree as ET
from urllib.request import urlopen
from intermine.webservice import Service

es = Elasticsearch('http://localhost:9200')
drop_index = es.indices.create(index='worm', ignore=400)

def search(url):
    final=[]
    xml = urlopen(url)
    tree = ET.parse(xml)
    root = tree.getroot()

    for neighbor in root.iter('class'):
        attribdata=neighbor.attrib #New dictionary that would contain the model data
        allattribs=[]
        for attr in neighbor.iter('attribute'):
            attrib_val=attr.attrib
            attrib_val_final=attrib_val['name']
            allattribs.append(attrib_val_final)
        newval={'attribute':allattribs}  #Contains all the attribute data
        attribdata.update(newval)  #Added to the dictionary we created
```

To utilise the services of Elasticsearch, we need to index the data beforehand. Elasticsearch is a tool utilizing the index found at the back of a book to look for the word and for the page number it occurs on, similarly it indexes documents to later retrieve them by the index rather than going through the whole document. Thus for the data from each mine, I have assigned one index name. Since Elasticsearch allows us to query across different data indices, it becomes quite convenient.
Here is the code to include the data from the ‘worm’ mine-

Once the data has been ingested into Elasticsearch, it can be **easily verified via the command line.**

```
Underground-Revolution:~ sanatmishra27$ curl http://localhost:9200/_cat/indices/worm?v
```

And the response-

```
Underground-Revolution:~ sanatmishra27$ curl http://localhost:9200/_cat/indices/worm?v
health status index uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   worm  qxXW2e3cRECDwvKq9PxKEA   5   1        102            7    151.4kb        151.4kb
```

Here, I have added the data from Worm Mine and indexed it under the name ‘worm’. As you can see, the ‘docs.count’ and ‘store.size’ give relevant information about the indexed document.
Thus this is a permanent storage on the Intermine servers, but would occupy just few megabytes of space!

Finally we can iterate the above code to cover all mines. Again, as an example I have done so only for the wormmine.

```
minelist=['wormmine']
links=[]
for i in range(0,len(minelist)):
    link="http://registry.intermine.org/service/instances/"+l[i]
    get_req=request.get(link)
    dict=json.loads(get_req.text)
    links.append((dict['instance']['url'])+ '/service/model')
for i in range(0,len(links)):
    search(links[i])
```

PLAN:

1) To help set-up Elasticsearch on Intermine servers.
2) To ingest data from all other mines by proceeding as shown.
3) **To include Ontology Term details**, as and when it is part of the data model.

Step 2:

Objective: Writing code for the front end in HTML, with Flask as the web framework.

While designing the front end, I prefer using Flask which comes bundled with 'Jinja2 templates'. The 'Templates' feature of Flask is pretty handy, the most kickass thing about this being, that **it keeps the application logic and layout separate**. Hence in future we can migrate to any other web development package while preserving the logic it runs on.
Once a template has been designed, it is rendered to a HTML page. Every subsequent function in the Python code will have an output as described in the template files.

The main application file looks like this:

```
from flask import Flask, render_template, request
from elasticsearch import Elasticsearch

app = Flask(__name__)
es = Elasticsearch('localhost', port=9200)

@app.route('/')
def home():
    return render_template('search.html')

@app.route('/search/results', methods=['GET', 'POST'])
def search_request():
    search_term = request.form["input"]
    res = es.search(
        index="fly,legume,indigo,worm",
        size=3000,
        body={
            "query": {
                "multi_match" : {
                    "query": search_term,
                    "type": "best_fields",
                    "fields": [
                        "Mine",
                        "class",
                        "extends",
                        "attribute",
                        "collection",
                        "reference name",
                        "referenced-type"]
                }
            }
        })
    return render_template('results.html', res=res )

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

‘search.html’ and ‘results.html’ are the templates I have created separately, these will only contain the final layout in the search (home) and results page. Proceeding as planned, it’s also time to write the code for the Boolean filters through Elasticsearch. This will help **narrow results to include only specific mines-**

```
@app.route('/filter/<string:id>/<string:search_term>', methods=['GET', 'POST'])
def filter_request(id, search_term):
    res = es.search(
        index="fly,legume,indigo,worm",
        size=3000,
        body={
        "query" : {
         "constant_score" : {
         "filter" : {
            "bool" : {
              "must" : [{"term": {"Mine" : id}},
              {"bool":{
              "should":
                        [{"term": {"class" : search_term}},
                        {"term":{"extends": search_term}},
                        {"term":{"attribute": search_term}},
                        {"term":{"collection": search_term}},
                        {"term":{"reference name": search_term}},
                        {"term":{"referenced-type": search_term}}]


                    }
                    }
                ]
              }
            }
          }
       }
    }
    )
    return render_template('results.html', res=res )
```

The ‘must’ feature contains the filter you want to apply, i.e. the name of the mine- which the user will click whereas the ‘should’ feature contains the ‘OR’ criteria, i.e results should contain if any one of the following terms is a search hit. This functionality will be used in the following section, as it becomes clear through the UI.

PLAN:

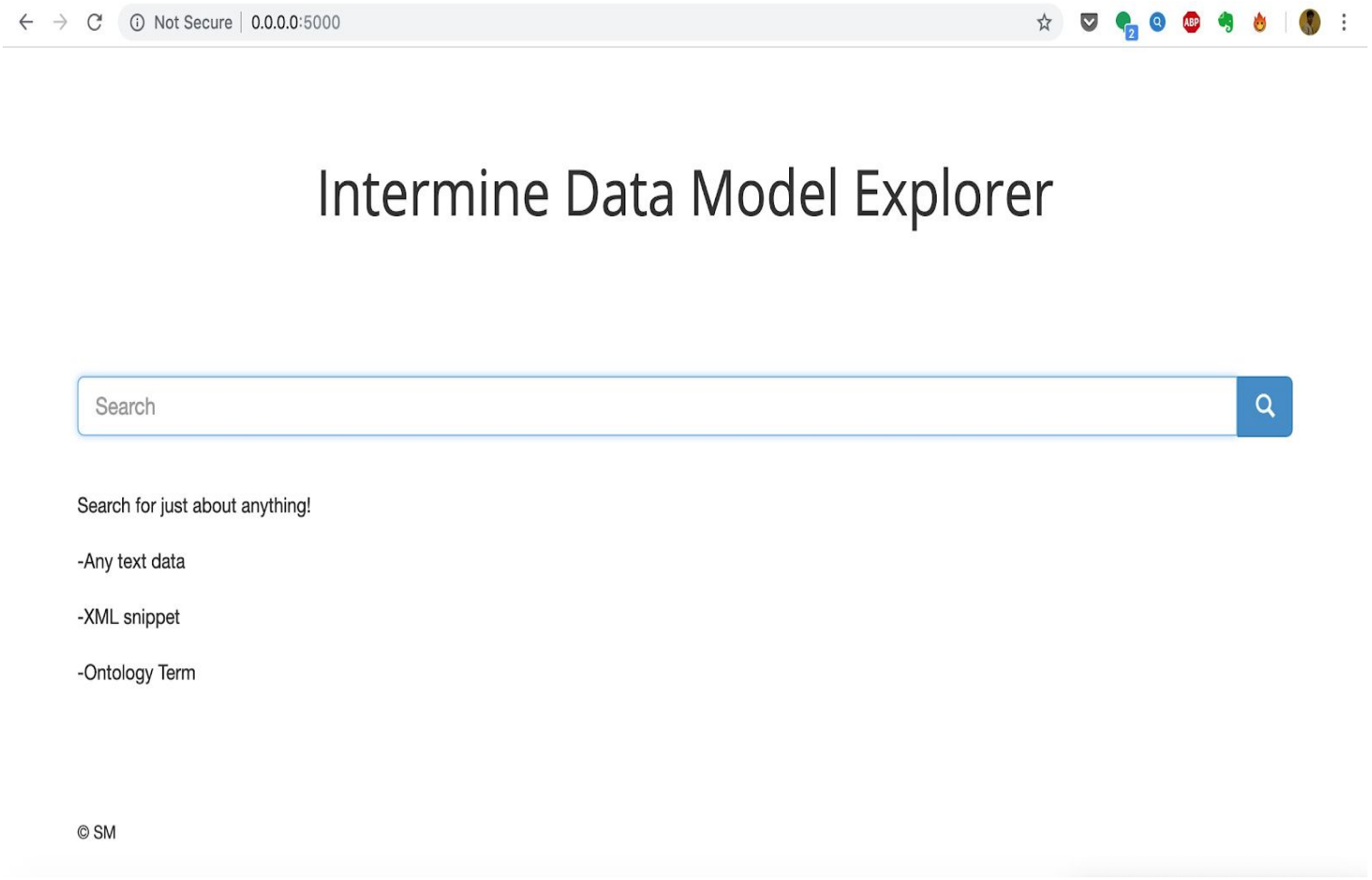
During the project, I shall be adding code to implement additional features, **some of which are**:

1) Fuzzy search
2) Partial match results
3) Pagination

Step 3:

Objective: Implementing the front end UI with Bootstrap.

I picture the homepage to be as **minimalistic as possible**, with just a search bar, this helps keep things simple.



To help out the user, I think we can add some text like as shown on the homepage to get an idea as to what all can be searched through this.
The code for this is written in HTML using Bootstrap and/or CSS wherever required.

PLAN:

1) Write relevant code in HTML through the Bootstrap/CSS framework to implement UI ideas/modify the above design.
2) Include relevant links to the Intermine website, if needed.

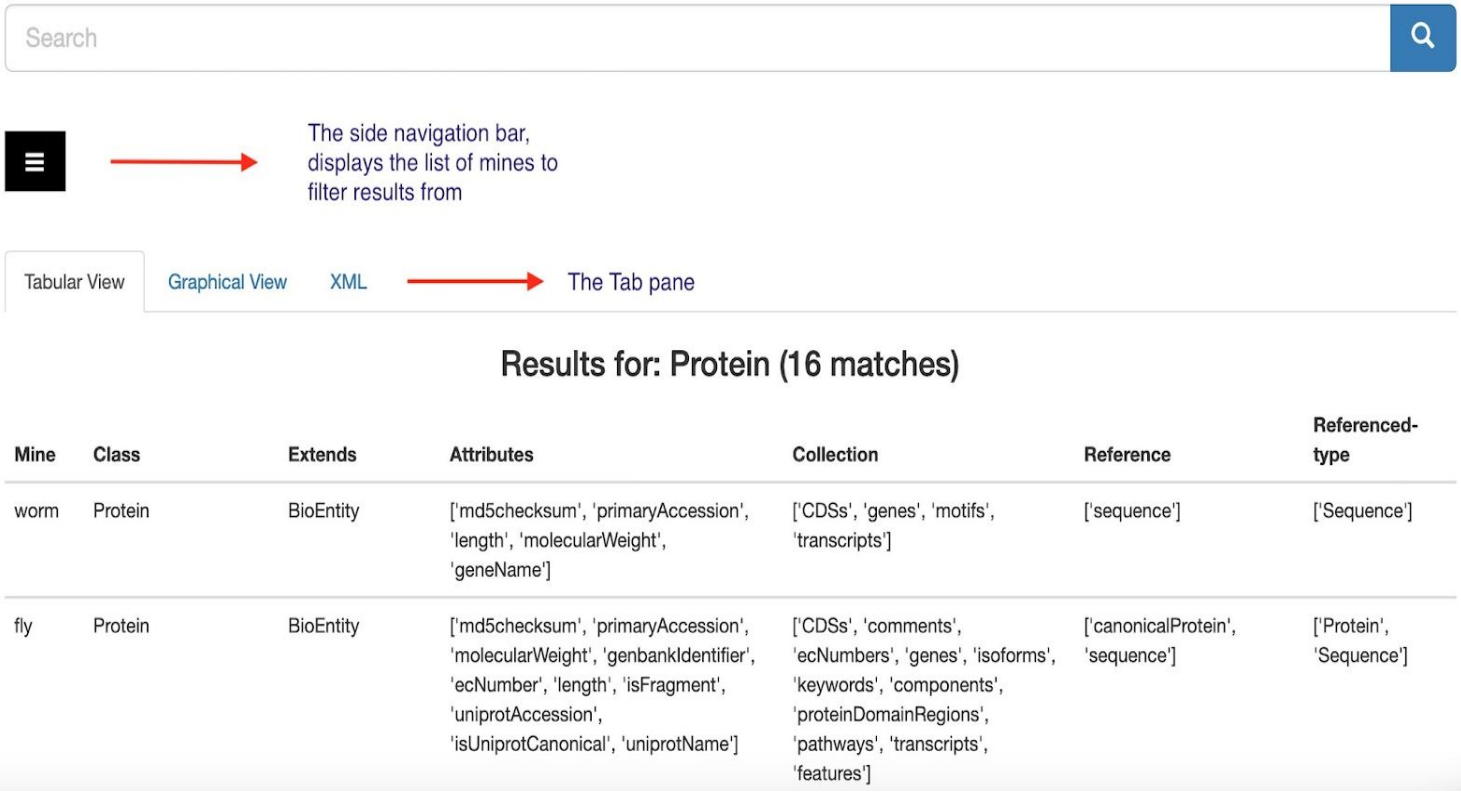
Step 4:

Objective: Implementing the 'results' page.

As an example, I will run a search for 'Protein'.

The 'results' page will **retain the search bar** to be available to the user in case he/she wants **to perform further searches**.

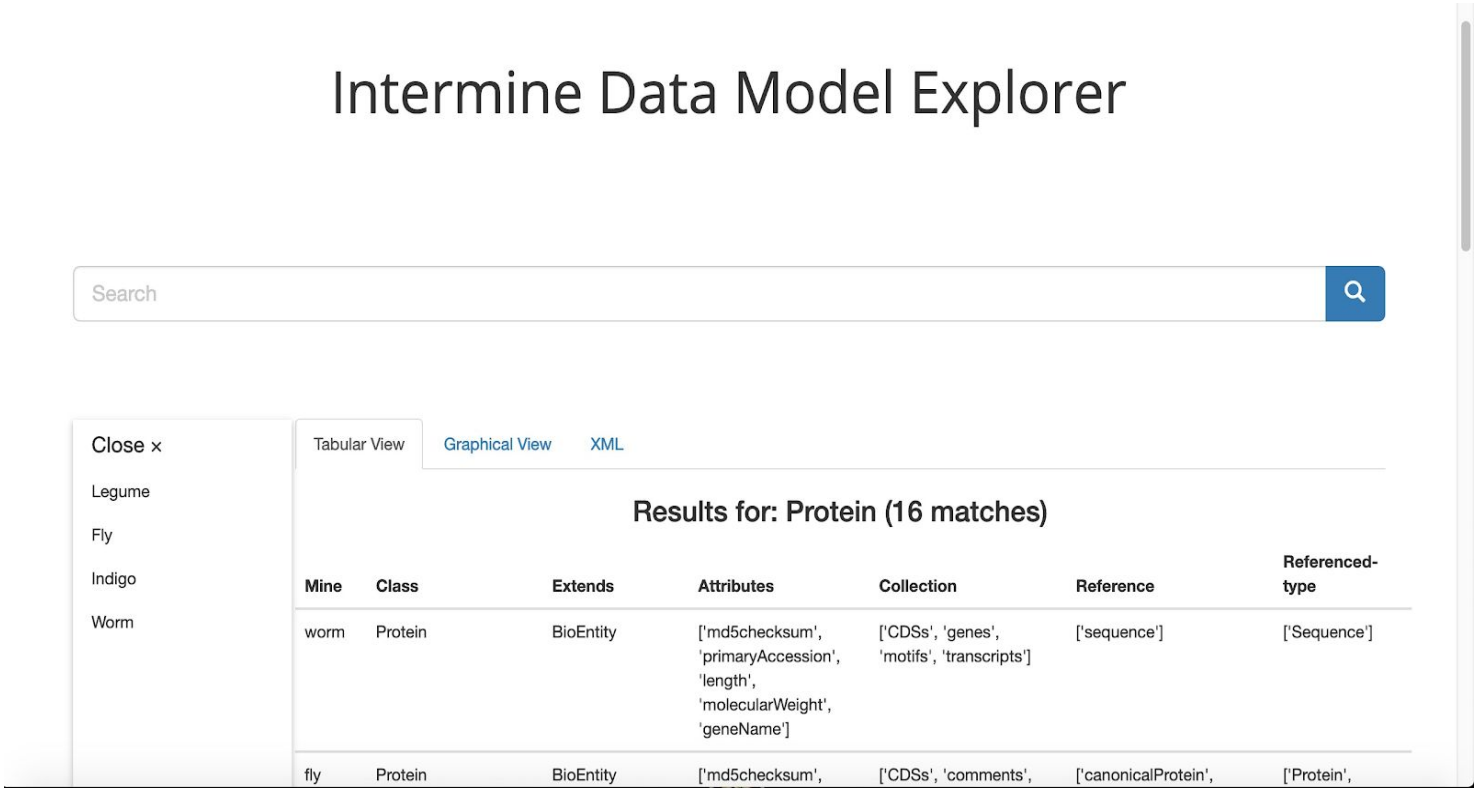
The rest of the design looks like this-



The search returns results from **any tag within which it finds the term** the user inputs, thus on scrolling we can see results where the reference tag contained the term protein and has hence showed up on the results page- (Note- that the Referenced type terms are corresponding to those in the Reference)

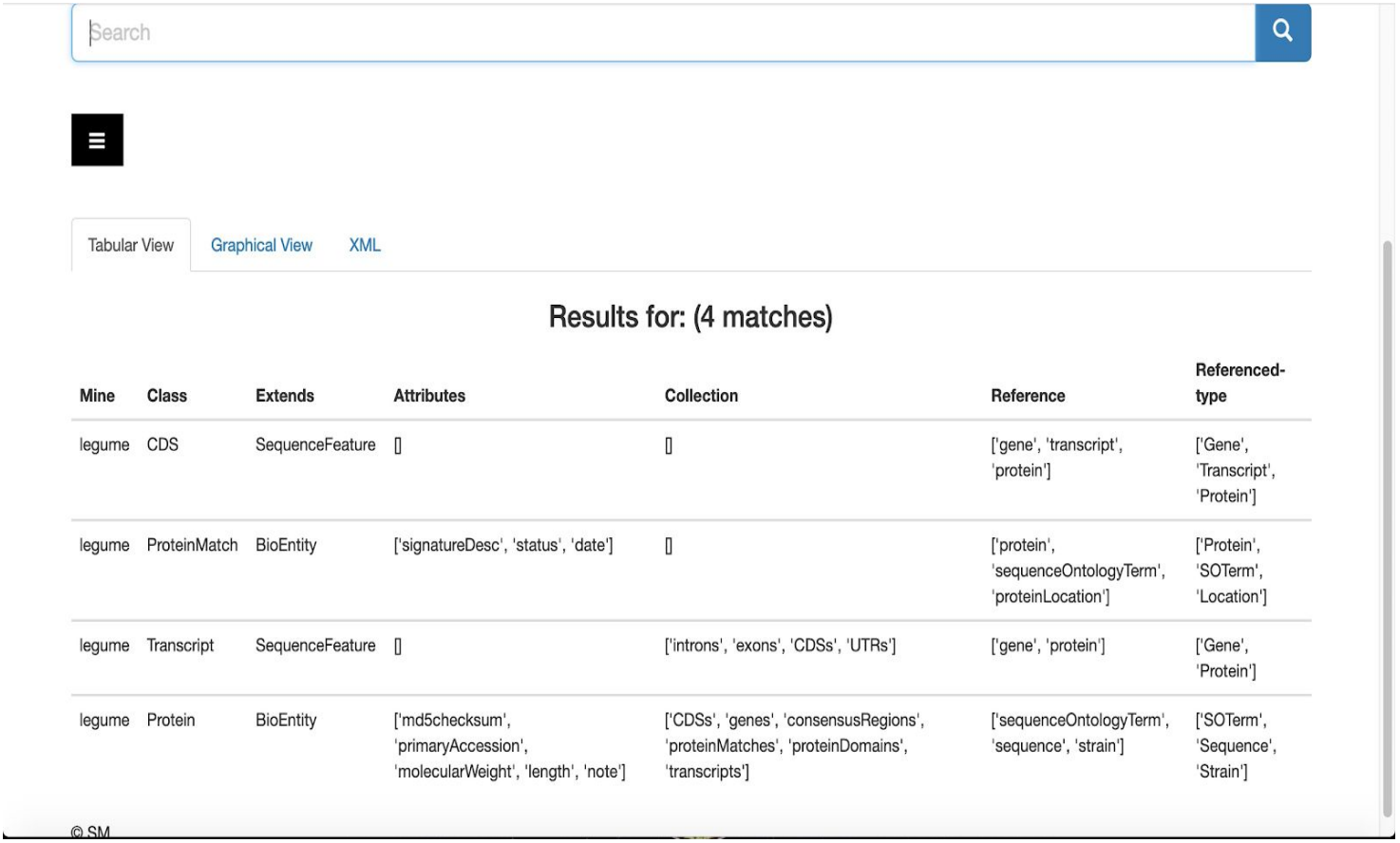
fly	Component		['name']	[]	['protein']	['Protein']
legume	CDS	SequenceFeature	[]	[]	['gene', 'transcript', 'protein']	['Gene', 'Transcript', 'Protein']
fly	UniProtFeature		['begin', 'description', 'type', 'end']	[]	['feature', 'protein']	['OntologyTerm', 'Protein']
worm	CDS	MRNARegion	[]	['transcripts', 'affectedByAlleles', 'RNAi']	['gene', 'protein']	['Gene', 'Protein']
indigo	CDS	SequenceFeature	[]	[]	['gene', 'transcript', 'translation', 'protein']	['Gene', 'Transcript', 'Translation', 'Protein']
legume	Transcript	SequenceFeature	[]	['introns', 'exons', 'CDSs', 'UTRs']	['gene', 'protein']	['Gene', 'Protein']
fly	CDS	SequenceFeature	[]	[]	['gene', 'transcript', 'protein']	['Gene', 'Transcript', 'Protein']
worm	Transcript	SequenceFeature	['method']	['introns', 'exons', 'affectedByAlleles', 'UTRs', 'CDSs', 'RNAis', 'preMiRNAs']	['gene', 'protein']	['Gene', 'Protein']
fly	ProteinDomainRegion		['identifier', 'start', 'database', 'end']	[]	['protein', 'proteinDomain']	['Protein', 'ProteinDomain']

On clicking on the side navigation bar button, it **displays the mines to filter** your results from-



Since I had added data from four mines into Elasticsearch, I have added only those corresponding options into the side nav bar to filter results from. (Legume, Fly, Indigo and Worm)

On clicking Legume- **the results are filtered** to display only those from the Legume mine.



One of the feedbacks I received was to be able to view the data types and mines in a perhaps lesser elaborate way. I can easily include a table to be able to quickly filter through the mines.

Here is an example of what I'm referring to, picture it is just representative of what I'm thinking of doing-

	FlyMine	modMine	metabolicMine
Genome Annotation			
sequence	✗	✗	✗
genes	✗	✗	✗
transcripts	✗	✗	✗
exons	✗	✗	✗
regulatory regions	✗	✗	
microarray probes (expression/tiling)	✗	✗	
ESTs	✗	✗	
insertions / deletions	✗		✗

(Taken from- https://www.researchgate.net/figure/Data-types-present-in-different-instances-of-InterMine_fig1_238314673)

PLAN:

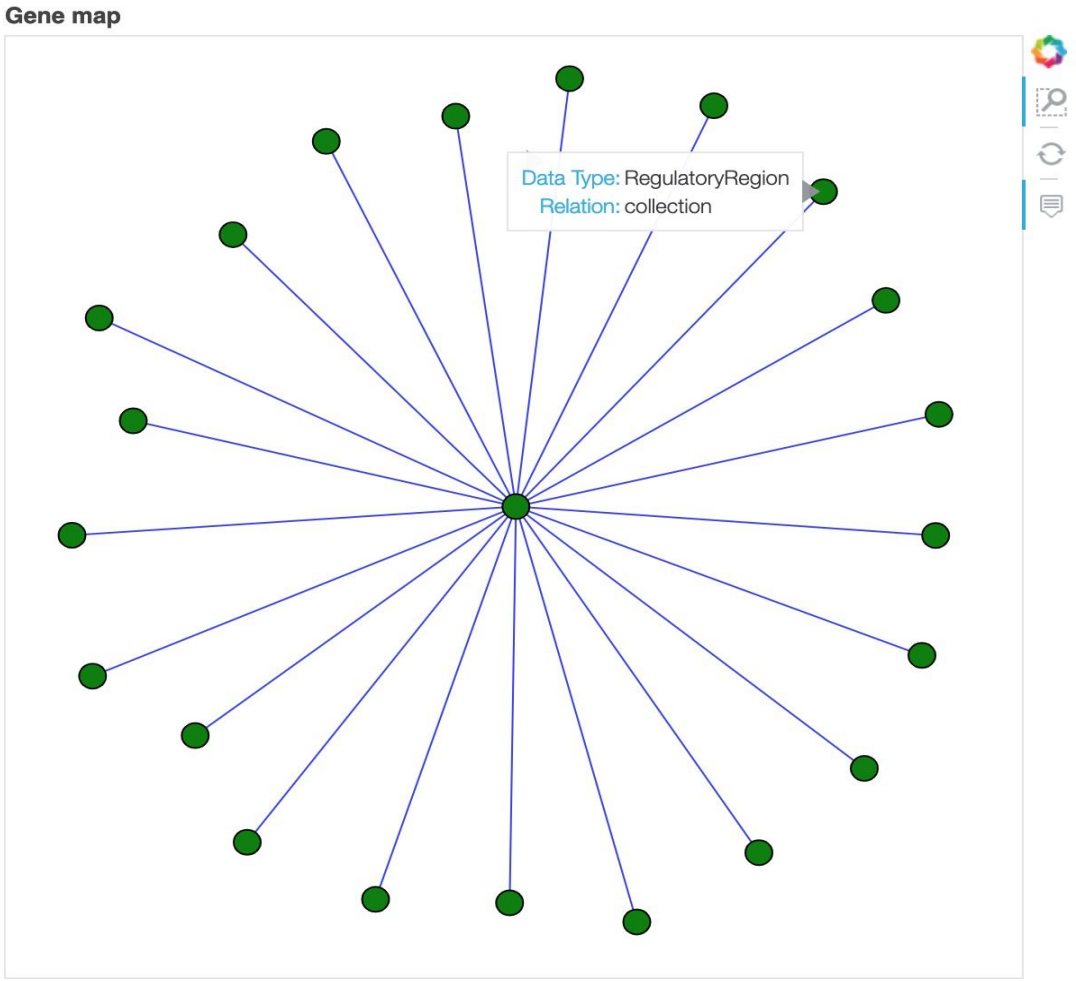
1) Include all mines in filter section.
2) Enable application of certain filters together, so one can see the results for 'Gene', in say both Worm and Fly mines.

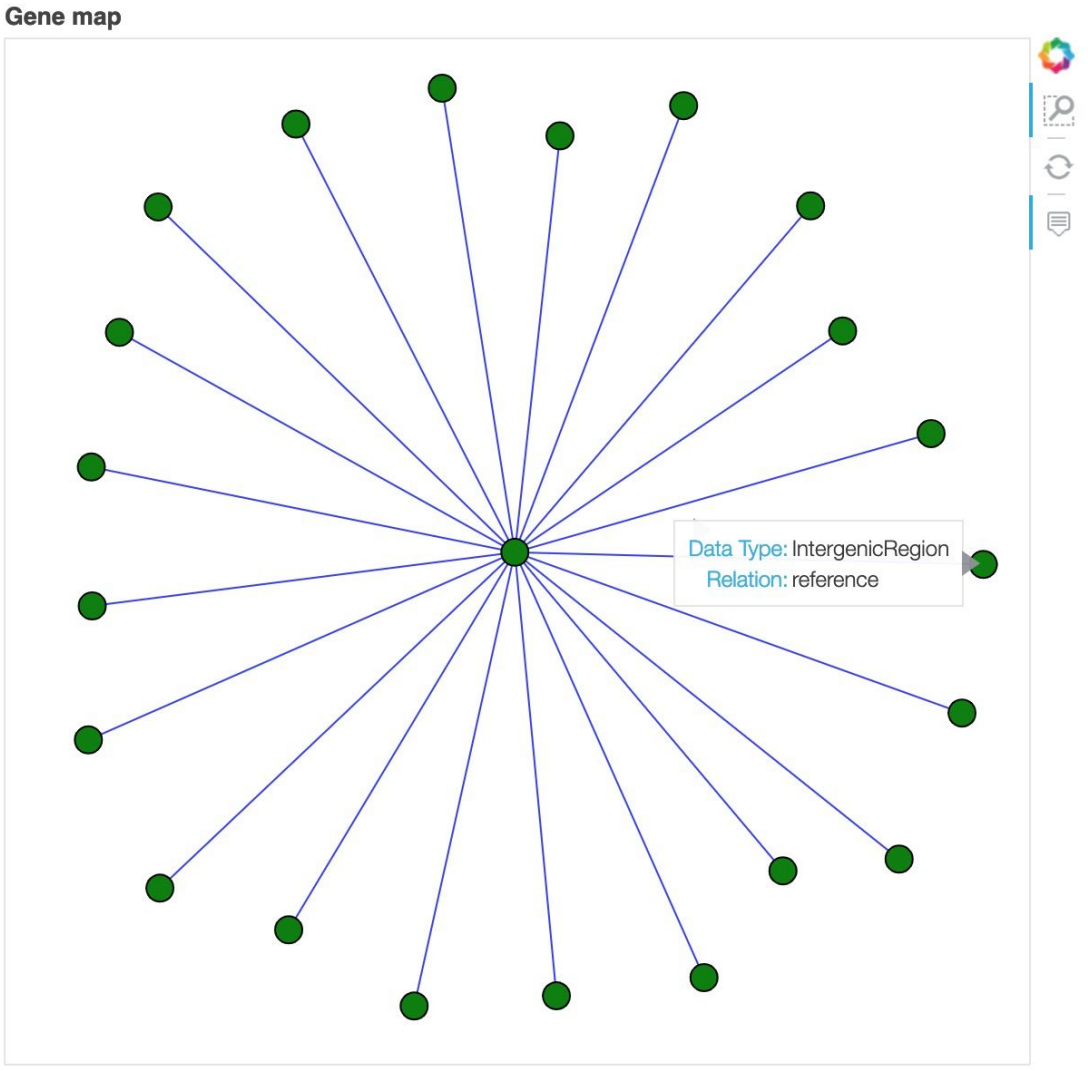
3) Review column format in Tabular Form with mentors.

Step 5:

Configuring ‘Graphical View’ and ‘XML’ Tabs.

I shall be using the Networkx module to graph the relations between the data types. The most appropriate plots for this purpose are network graphs, which can be produced quite easily using Networkx. Then I shall export it as an **interactive visualisation in HTML using Bokeh**. Bokeh has an **inbuilt library** to render Networkx plots and is an amazing library to render interactive visualisations as against static plots with a vast number of creative options. Here is an example I have made in Bokeh, where I have plotted the objects Gene references to in FlyMine. **On hovering the cursor over one of the nodes,** we can see it reads the ‘Data Type’ as RegulatoryRegion and the ‘Relation’ as Collection. This displays the objects that are under ‘referenced type’ for ‘Gene’ in Flymine.



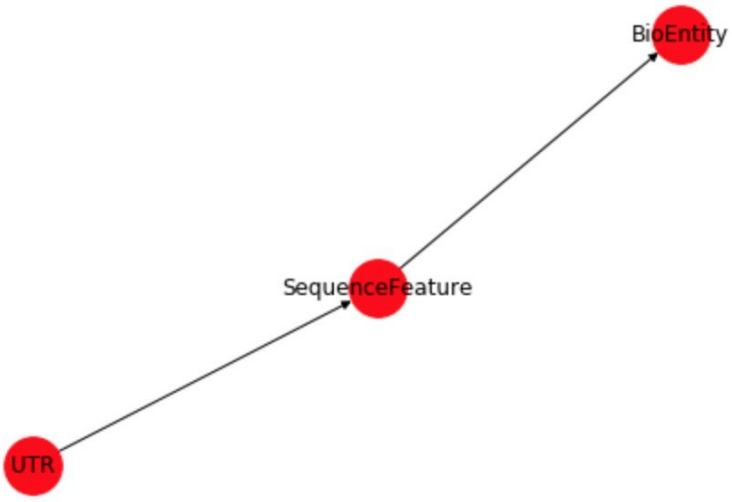


This is another example, which is **hovered over another node** and reads the information it contains.

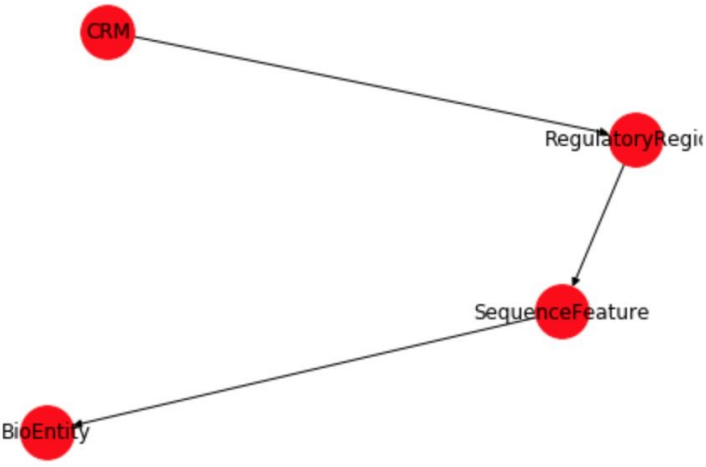
Bokeh has many **features to make plots interactive**, one can ‘Scroll’, ‘Pinch/Zoom’, ‘Selective neighbourhood highlight’- highlights only those edges which are related to one node on placing the cursor over it.

Here are a few static plots from Flymine displaying inheritance among data types through the ‘extends’ feature, only the data type is inputted and it generates the following-

```
The graph for UTR is:
```




```
The graph for CRM is:
```



Once this is complete, I shall work on the XML tab.

Generating the XML snippet for a given data type can be done in two possible ways:

1) Using the etree module to write a XML document given an input dictionary.

2) Using the ‘dicttoxml’ library.

Both of these shall be considered to see which one is most suited for the task. Similarly, a similar option will be explored for JSON format.
Besides these, the links to **the complete data model of a chosen mine shall also be made available.**

PLAN:

1) To plot interactive directed graphs displaying inheritance among various data types.
2) To make graphs based on the type of relationships (many-to-many or one-to-many) that exist between different objects.
3) To implement plotting of a single data type showing the objects it references to.
4) Showing the entire data model as a network graph.
5) Make the ‘XML’ tab active, by providing export features as described above.

To display the single graph, the user will have to indicate **which mine he/she wants to view graphically, and only after that choice has been made, will the ‘Graphical View’ tab become active.**
A further option to plot multiple graphs (from different mines, and different data types from a single mine) which are from different shall also be made available.

DOCUMENTATION:

I personally feel that a search tool should need no documentation as that defeats its very own purpose! That being said, It should be as simple as possible, the user must intuitively know how to operate it and it must not be a radical departure from identical tools on the web.

All relevant and necessary softwares, **the code base with proper documentation and guidelines shall be added to the GitHub repo** to help other developers, who might want to maintain and improve this tool's functionality.

This completes the implementation plan, now let's look at the various technologies/libraries I plan on using and the reasons for choosing them.

SOFTWARE PACKAGES USED-

- Elasticsearch

  License: Apache 2.0

  Elasticsearch is a highly scalable full text search library. Since it has been around 2010, there are plenty of use-cases, most well known being Wikipedia, The Guardian, GitHub among others, which speaks of its **stability and reliability**. Besides there's plenty of help available on the internet in implementing it, which certainly helps.

  ES works by creating an inverted index for its documents, it's just like looking for a word in a book by the 'Index' provided at the end of it rather than manually going through each page! ES is a natural choice in case the data is in JSON. As a major plus point, the indexed data is available **for search in just about a second making the system almost real-time**!

  Besides the documentation is well maintained and it can be set up with ease very quickly.

- Flask

  License: BSD

Flask is a lightweight web framework in Python. The reason it beats its competitors is because of its **easy-to-use, lucid core structure**. It is amazingly flexible and can be extended wherever needed. One of the primary reason to choose Flask is also because hardly **very little of its functionality remains unexplored and unutilised** as against heavier frameworks like Django, which also contain significantly more boilerplate code.
Since it is **based on Python**, it makes it my first choice and as it is, is easy to configure and implement.

- HTML5
  HTML is the most classic web development language out there. Besides being incredibly simple and something I am acquainted with from school days, it provides the blueprint of almost all websites with nearly an **identical structure across versions**. Combine this with **CSS for styling and Bootstrap for responsiveness**, and we get a complete package for styling and deployment.

- Networkx
  License: BSD-new
  Since the **data model of Intermine can be conceived as a network**, with data types attributed as nodes and the references attributed to edges, my first choice was Networkx. It is a simple module **with clear syntax and rich documentation.** It has expressive features and a strong online community. It is the number one choice of the scientific community. A big plus is its feature to import and export graphs in many standard and non standard file formats.

- Bokeh
  License: BSD 3-clause New

The natural consequence of choosing Networkx was to include Bokeh. Networkx is very good at generating static plots, since it is based on Matplotlib, hence **to include dynamic data visualizations** I chose Bokeh. This has an inbuilt feature to include graphs generated with Networkx and hence **applying Bokeh's functionality to a Networkx graph** is smooth and no work at all. This was a major factor why I chose Networkx in the first place, and Bokeh over the likes of HoloView.
These visualisations can then be **easily rendered on the browser** with a few lines of HTML code.

Other modules that have been used as well are-

- Etree- For parsing XML documents. Its C-extension is the fastest and lightest XML parser among others.
- Urrlib.requests
- unittest
- IO

Timelines: Milestones

Community Bonding Period:

(May 6-May 27)

This time is critical for me to explain my vision of the tool to my mentors, at the same time familiarise myself with their ideas keeping in mind the community's necessities. I will revisit my understanding of the data model and clarify my doubts, if any, regarding it with my mentors as quickly as I can.
Since I will need practice in writing tests, I shall devote most of my time learning and implementing tests. **I will set up Travis CI during this phase.**
Besides that, I will run my project through my mentors, discussing the timeline and prioritising tasks, including preparing for any glitches and bottlenecks I might encounter.
I also hope to learn more about the Intermine community and my peers who will be working on different projects during this timeframe by interacting and keeping in touch over Discord/email.

Coding Period.

(May 28- August 19)

Week 1 and 2:

(May 28- June 10)

Indexing documents for Elasticsearch and implementing its features.

Data from all thirty mines will have to be ingested in the format explained above. Once done, different features of Elasticsearch can be exploited.

The following features will be implemented:

- **Best match**- To ensure that their are search results in case the input term is contained in any of the fields. (class, attribute, collection etc.)
- **Partial matches**- Some terms are long and a user might not remember the entire word, enabling partial matches ensures that the user can quickly find the desired term and its relationship.
- **Fuzzy Search** - to include cases involving typos, incorrectly spelled words, US/UK spellings.
- **Filters-** To include search options within one particular field alone.
- **Pagination-** Search result might number to hundreds of queries from many different mines, which cannot be rendered on a single page. Thus the pages will have to be numbered and displayed in batches.

OPTIONAL FEATURES:

The following features can be included (if there is time leftover within these two weeks and if mentors find them relevant)

- **Highlighting-** Terms that are found by running a search can be highlighted as well. This increases visibility.
- **Autocomplete-** A search-as-you-type feature and an autocomplete feature can be enabled based on most searched terms.

Week 3 and 4:

(June 11- June 24)

Configuring 'search', designing 'graphs' and developing export options.

- So far only data types have been included in the search functionality,

however XML input and Ontology Term need to be incorporated as well. Thus a user can now input three types of information, a data type, an ontology term or a XML code snippet.

- The graphing tool can as of now, implement the relationships between classes within a particular mine. However, it should be able to do the following as well-
  1) **Display directed graphs simultaneously for more than one selected mine** for a particular data type.
  2) Contain **information on the edges,** regarding the two nodes it connects.
  3) Display the **mines that contain a particular data type,** as a directed graph.

  The code for generating the plots can be reviewed as well to optimize size and execution time.
- Developing export features inside the 'XML' tab- to include downloadable links to both JSON and XML formats.

Week 5 and 6:

(June 25- July 8)

Configuring the 'results' page and making the graphing tool available to the user.

- The 'results' page will be configured. The 'Tabular View' tab shall be designed fully. This will complete the design and implementation of the three tabs on the 'results' page.

  The 'results' page will thus contain-
  1) Three navigable tabs- **'Tabular View', 'Graphical View' and**

'XML'.

2) A side navigation bar **to apply filters**- Will enable filters to choose among mines to activate 'Graphical View'.

- Since the graphing tool is fully equipped, it can be deployed. The corresponding HTML code and the routes for Flask will be written.

Week 7 and 8:

(July 9- July 22)

Writing tests and doing a code clean-up.

Since the backend would be fully complete by this time, running tests now will make sense.

- During this phase, I shall be writing tests with the unittest module in Python. Once complete, Travis CI can be used for continuous testing. (This is setup during the Community Bonding Period.)
- **Interface testing**- To ensure that the data flow is as expected.
- **Compatibility testing**- To ensure that the tool is accessible from all popular web browsers and screen sizes. (Phones/Tablets) and that the **pages scale** accordingly.

I will finalise the code and go through the work that has been done so far.

Week 9 and 10:

(July 23- August 5)

Conclude the UI design and implementation.

- I will discuss the UI design with my mentors to see if there is any issue with it, I will also seek feedback to improve the design and aesthetics of the tool.
- If there is time left in this phase, I will begin writing documentation that covers the tests I have coded, installation documents and explaining the features of the search tool to ensure it can be easily updated with new data.

Week 11 and 12:

(August 6- August 19)

General feedback, bonus section and documentation.

- Complete any pending work.
- If I have not begun with the code documentation, I will do so , also receive and implement feedback during this period.
- Prepare my presentation for the Intermine community call on 15th August.
- If time permits, I shall try to work on **at least one bonus feature**.

DELIVERABLES:

REQUIRED DELIVERABLES:

- Designing and implementing a web based search-cum-graphing tool for the Intermine Data model. This includes pushing relevant code to fork as soon as possible, sending complete pull requests once the code cleanup is complete.
- Testing the code with unit tests running on Travis CI.
- Detailed documentation covering code and for extending the software in future to include changes to the data model, it will be made available on GitHub.
- Presentation at the Intermine community call detailing all my work on August 15,2019.
- Writing blog posts biweekly on Medium covering my work and general experience with Intermine and Google Summer of Code.

OPTIONAL DELIVERABLES:

- Adding features such as: graphical view of the data model of one selected mine.
- Analytics:
  1. Percentage of a data model annotated with an ontology term.
  2. Similarity finder between models- implemented through Elasticsearch
- Benchmark for speed of search tool in returning results

PERSONAL BACKGROUND

Hello, my name is Sanat Mishra. I am a second year undergraduate student at IISER, Mohali, India. I am pursuing a BS-MS integrated course, and I will be majoring in Biology. I would love to work at the intersection of computer science and biology.
I spend most of my spare time quizzing or catching up on football!

My skill set includes Python, with brief experience in HTML, C and Sage.
I use a Macbook Pro that is running on macOS Mojave (v 10.14.4), while I split my work between Jupyter Notebook running on Anaconda and Terminal. Both of which are super convenient to work on!
The open source community is a platform for useful work which happens through collaborations and community interactions. It is an inspiration for me and others alike who get motivated looking at other people's ideas and efforts. The work we do has the potential to improve the experiences of a large number of people and impact their work and lives positively.
I really look forward to make a meaningful contribution to Intermine to help the Life Sciences and Genetics community out there.
In the past I have built interactive Arduino-based projects (based on C), am passionate about solving problems on Project Euler (code solutions to Math problems), taken part in Hacktoberfest and have reviewed code for my college peers.
One of the first courses I took in Python, was by Dr Charles Severance from University of Michigan, Ann Arbor, and I have rigorously built on the basics I picked up from there.
So far, in college I have taken two relevant courses, the first one was a programming course (Sage and Python) and the other is a theoretical course covering the theory of computation.

These are my contributions to Intermine so far-

- Wrote a tutorial for the Intermine Python client to show how the data model can be retrieved and manipulated. (PR has been submitted)
- Extended 'Tutorial 7' of the Python client by adding another example, to show how to modify constraints.
- Removed small typographical error in the tutorials.
- Currently, upgrading the documentation page to Python 3.x.

Presently, I am documenting my previous work on Github. I am confident that, given a chance, I can undertake the work I have begun here and see it it to fruition.