2018-metabrainz.md

Google Summer of Code Proposal

Name. Sumit Ghosh
MB Username + IRC Nickname. SkullTech
Github. SkullTech
Website + Blog. https://sumit-ghosh.com
E-mail. sumit@sumit-ghosh.com
Time-zone. UTC +5:30

Importing data into BookBrainz from third-party sources

Project Overview

As of now, BookBrainz only supports adding data to the database manually through the web interface. Building a database of considerable size only by this method would be highly inefficient. To solve this problem, we could develop a complete system for importing third-party data into the BookBrainz database easily.

The following text taken from here sheds light on the plan developed my *Leftmost* and *Sputnik* regarding this matter.

> At last year's summit, the two BookBrainz lead developers, Leftmost and LordSputnik worked on a plan for importing third-party data into BookBrainz. This plan has several stages. First, data sources need to be identified, including mass import sources with freely available data, such as libraries, and manual import sources, such as online bookstores and other user-contributed databases. The next stage is to update the database to introduce an "Import" object, which can be used to distinguish mass imported data from (usually better quality) user contributions. Then, actual import bots for mass import and userscripts for manual import will need to be written. Finally, it would desirable (but not necessary if time is short) to introduce an interface to the BookBrainz site to allow users to review automatically imported data, and approve it.

As mentioned in the text above, we can work on this project dividing it into a few distinct stages.

1. Updating the database schema.
2. Designing and creating the pipeline for importing and inserting data into the DB.
3. Identifying data sources and writing the importer for those sources.
4. (if time permits) Making a web interface for users to review conflicts.

Updating the database schema

On the database side, a new table for storing the sources will be created. It will store an identifying name of the source, along with corresponding URL and description. A rough schema of the table is outlined below.

```
CREATE TABLE bookbrainz.source (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) NOT NULL UNIQUE CHECK (name <> ''),
        url TEXT NOT NULL DEFAULT '',
        description TEXT NOT NULL DEFAULT '',
);


CREATE TABLE bookbrainz.import (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) NOT NULL UNIQUE CHECK (name <> ''),
        url TEXT NOT NULL DEFAULT '',
        description TEXT NOT NULL DEFAULT '',
);
```

As we will be creating new entities from the imported data, we will need to specify the source in the stored data. For that, we will be modifying the entity table, adding two additional fields to it. One of the fields will indicate whether the entity is an imported one or user-contributed one. The other will point to the source in the case it is an imported one. The corresponding schema is outlined below.

```
CREATE TABLE bookbrainz.entity (
        bbid UUID PRIMARY KEY DEFAULT public.uuid_generate_v4(),
        type bookbrainz.entity_type NOT NULL,
        imported BIT NOT NULL DEFAULT 0,
        source_id INT,
        CHECK ((source_id IS NOT NULL OR OR imported IS 0) AND (source_id IS NULL OR imported IS 1))
);
ALTER TABLE bookbrainz.entity ADD FOREIGN KEY (source_id) REFERENCES bookbrainz.source (id);
```

Here we added a check statement in the definition of entity, which takes care of the interdependent constraints between `source_id` and `imported`.

Designing the *data import pipeline*

We will be creating a modular structure for the importer, so that new sources and the importer for that can be added later quite easily. The basic project structure will be somewhat like below.

```
.
├── dumps
│   └── librarything-2018-03-15T20:34:17+00:00.json
├── importers
│   ├── librarything.py
│   └── openlibrary.py
└── insert.py
```

Here, the scripts inside the `importers` directory import data from the corresponding source and then dump it in a specified format in a JSON file inside the folder `dumps`. After that, on running the `insert.py` script, it inserts the data dumped into the database. An example command to run it would be like the following

```
$ python insert.py -d librarything-2018-03-15T20:34:17+00:00.json
```

Things to note here.

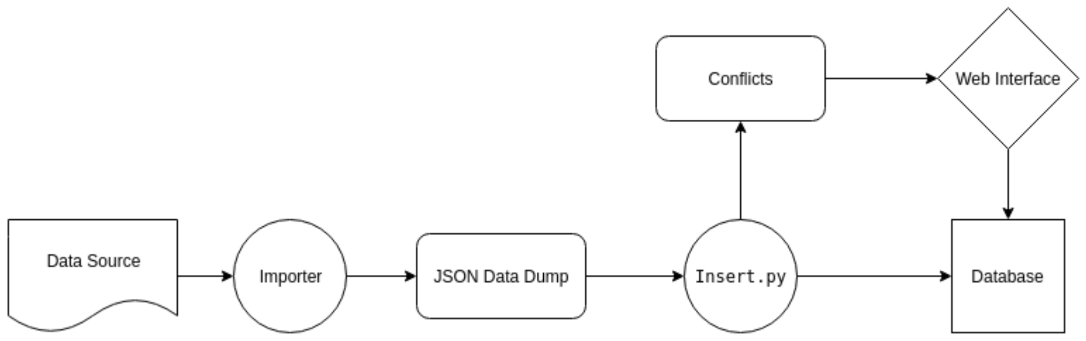
- The JSON files inside the dumps folder are of very well-specified format and are machine-generated.
- The importer scripts are free to import the data in whatever way they prefer. Possibilities include
  - Downloading and then reading a data dump file, such as the ones provided here by OpenLibrary.
  - Accessing a web API.
  - Scraping the content off web-pages. (Very unreliable, would avoid).
- The modularity of this approach lies in the fact that the insert.py script does not care how the data was imported, it just reads it from the dump JSON file, and inserts it into the database.

Handling duplicates and conflicts

While inserting entries into the database from the data dumps, we will have to take care of the following

- Avoid duplicate entries
- Resolve conflicting entries

For this, the `insert.py` script can do the following. Before inserting, check if the database already has an entry for the same item. If it has, and if the data is conflicting (i.e the values stored in the database and the values imported do not match), save it separately for manual confirmation by a user. It can either be stored in a separate table in the database, or in a JSON file (something like *conflicts.json*).



Identifying data sources and writing importers

This step would involve researching about available sources for data, having a permissive license. Writing the importer scripts for those sources will have to done parallelly with this. Also, this stage can continue indefinitely, i.e new developers joining the project later on, can keep extending this part of the pipeline.

Infrastructure to be used.

- We will be writing all the scripts (importers and *insert.py*) in Python.
- For CLI facilities of the scripts, the `argparse` library will be used.
- For interaction with the database, we can use something like peewee. It's a simple and clean ORM for Python. If it turns out to be inadequate, we can fall back to SQLAlchemy.
- For writing test suites, pytest could be used.
- For documentation, Sphinx along with Read the Docs is an excellent solution.
- For any potential scraping and crawling, the Scrapy framework can be used.

Timeline breakdown

The total coding time for SoC is 12 weeks, divided into 3 phases.

Timeline

1. **Week 1.** Updating the database schema.
2. **Weeks 2 to 3.** Designing and creating the data import pipeline.
3. **Week 4.** Writing test cases and documentation for the above.

Phase 1 evaluation.

4. **Weeks 5 to 8.** Identifying data sources and writing importers. Also writing the test cases and documentation for these parallely. We would try to cover at least a data source per week, so that we can write 4 code for 4 data sources at least by the end of this phase.

Phase 2 evaluation.

5. **Weeks 9 to 10.** In case there is some backlog, complete the work. Otherwise, make a web interface for users to review conflicts.
6. **Weeks 11 to 12.** Finishing up the project. Making sure the codebase is complete with test-cases and documentation. Making sure the code is deployable in production.

Final evaluation.

7. **Beyond.** Keep writing code for various MetaBrainz projects!

About Me

I am a Computer Science major undergrad studying at IIT Delhi. Check out my resume here.

Question: Do you have Github, Twitter or any other public profiles you think we might find interesting?
Answer: Yes, I do have some public profiles I think you might find interesting.

- Github: I put up most of the projects I work on here. I built many of these for small freelancing gigs that I did through Upwork.
- Personal Website: My blog, double-serving as my personal website.
- Twitter: Disclaimer, I am not an avid Twitter user.
- LinkedIn: For jobs and internships and stuff. The chore.
- Goodreads: I wanted to put this because I'm an avid reader, and that's one of the primary motivation I want to contribute to BookBrainz. I share all of my reading activities here on my Goodreads account, so you might get to know my taste from there.

Question: Have you contributed to other Open Source projects? If so, which projects and can we see some of your code?
Answer: Yes I have contributed to some open source projects. Listing them below.

- **MusicBrainz Picard Website**. This is the project through which I got exposed to the MetaBrainz developer community for the first time, almost a year ago. Here are my pull requests.
- **Mozilla's Webcompat Project**. Below are the Mozilla repos related to their *webcompat* project to which I've made contributions recently.
  - **Webcompat Issue Parser**. This is a tool for downloading and parsing the webcompat issues. Here are my pull requests.
  - **Autowebcompat**. This project aims to automatically detect web compatibility issues using Selenium and Neural Networks. Here are my pull requests.
- **ProxyMan**. This is a handy tool to configure proxy setting on Linux easily. Here are my pull requests.

Question: What sorts of programming projects have you done on your own time?
Answer: I like to make small but useful projects using the concepts I learn to turn the newly acquired knowledge into something useful. Also along the way I usually end up learning a bunch of new things, so that's a bonus! Here are some of the projects I recently done on my own time.

- **PearSend**. A simple CLI client for peer-to-peer file or message sending, written in Python. Implements protection against transmission error using CRC32 checksum.
- **Goodgives** (WIP). A CLI tool + web-app for auto entering giveaways in Goodreads. I had always been a fan of books, and who wouldn't love some free books!
- **Journalist**. An app to write journal digitally. It lets you write your journal using Markdown in your favorite text-editor and view the journals (Markdown rendered in HTML) in browser. It stores written journals in a comprehensive directory structure.

Other than these, I've made a lot of small CLI apps and web-apps, for freelance gigs or just simply learning. Check out my Github for the full list.

Question: Tell us about the computer(s) you have available for working on your SoC project!
Answer: I have an ASUS Laptop (ZenBook UX430UA). It's having an Intel Core i7 processor, 8 GB RAM and 256 GB SSD. I dual-boot Windows 10 and Ubuntu 17.10 on it, doing all the programming related work on Ubuntu. I also use DigitalOcean VPS's for running long-running scripts and testing purposes.

Question: When did you first start programming?
Answer: I was introduced to programming at the age of 12, through QBASIC, in my secondary school. From my first exposure, I was hooked, but not having a personal computer was a inhibiting factor. Later, when I was around 15, I got a computer, and have been programming passionately ever since.

Question: What type of music do you listen to?
Answer: I love to listen all kinds of music, but in particular I have a weakness for alternative and psychedelic rock. Some of my favorite rock bands are Pink Floyd, Muse and Coldplay. I also like listening to Hip-hop and R&B once in a while, for example artists like Kendrick Lamar, Eminem, and The Weeknd are some of my favorites too.

Question: What type of books do you read?
Answer: Recently I've been reading a lot of non-fiction, especially those related to philosophy, psychology, self-help, business, finance, etc, some of them being: Man's Search for Meaning, The Mythical Man-Month, Homo Deus: A Brief History of Tomorrow, etc. Among fiction, I recently read The Stranger by Albert Camus and 1984 by George Orwell, I loved both of them!

I read a lot of fiction when I was younger, especially high-fantasy works such as the Harry Potter series, the Narnia series, and The Hobbit. I also read and loved all the books by Michael Crichton, Dan Brown, and Indian author Satyajit Roy.

Question: What aspects of BookBrainz interest you the most?
Answer: Most of the already available book databases don't link to other databases. But BookBrainz does, through the different identifiers, such as Wikidata ID, ISBN, ISNI, VINF, Libraything ID, etc. I think this aspect of having a singular database which will link to all the different already existing database is exciting. Also, giving the users power to add and edit data makes it a community driven project, that is a interesting concept, different than something like Librarything. (although similar to Wikipedia and Wikidata).

Question: Have you ever used MusicBrainz to tag your files?
Answer: Yes, I have a large collection of MP3 files. And I use MusicBrainz Picard to properly tag them.

Question: How much time do you have available, and how would you plan to use it?
Answer: The summer vacation at my college starts from May 18, which almost exactly coincides with the starting of the official coding period, i.e May 18. So I will be available full-time throughout the SoC period. I plan to work at least 8 hours every weekday, and at the weekends I plan to work about 4 hours per day. Possibly I will give many more hours into this project, as I won't be having anything else to do.

Question: Do you plan to have a job or study during the summer in conjunction with Summer of Code?
Answer: No, if I get selected for SoC I won't be doing any other job or studying this summer.