A chromosome browser; middle layer implementation in Python.

Ioannis Valasakis
Birkbeck, University of London

May 7, 2018

1. Approach to the project

As was suggested we started looking individually at the data files, their documentation and the project requirements. Use the MAC OS X terminal, I used a few command line tools like head, less, grep to have a quick look at the provided human genome file.

```
zcat chrom12.gz | head -n 40 | less
```

This identified the file as a fasta with NCBI identifiers. Looking at the online specification, I found out it described as following(*FASTA format - Wikipedia*, n.d.):

Database	Format
GenBank	gb, accession, locus
EMBL Data Library	emb, accession, locus
DDBJ, DNA Database of Japan	dbj, accession, locus
NBRF PIR	pir,,entry
Protein Research Foundation	prf,, name
SWISS-PROT	sp, accession, entry name
Brookhaven Protein Data Bank	pdb, entry, chain
Patents	pat, country, number
GenInfo Backbone Id	bbs, number
General database identifier	gnl, database, identifier
NCBI Reference Sequence	ref, accession, locus
Local Sequence identifier	lcl, identifier

This gave really valuable insight into what is described in the fasta database format and how it will be later processed and analysed

Interaction with the team

We had a few team meetings that took place in Birkbeck Library and Computer rooms but the rest of the communication was done through Code Reviews in our GitHub repository, the Slack channel of our team (#group12) as well as personal messages when we had specific issues. Overall, the interaction was pretty effective although some minor issues (like software incompatibility, different versions of system tools) blocked some of our efforts.

Requirements for my contribution

The middle layer which includes the business logic was worked through a TDD approach (described in more detail here **??**). As the business logic part of the Chromosome Browser, it was really important to have all the information from the database to be able to communicate the results further to the front-end layer. I used a top - bottom implementation in Python and I tested using a custom json file, unless I could retrieve the real data from the database.

2. Performance of the development and process

Code testing

The development (as described above) was approached in a TDD aspect. Even though the time didn't allow for a full implementation and finalise of the test module, this can be found in the folder middlelayer/tests. The tests described there, were designed to fail in an initial approach and then each module was programmed and tested individually with a few stub styled data (as the db layer wasn't ready, a json format of a protein implementation was created).

This proved to be extremely useful, as a few edge cases were discovered. Specifically, that was in the implementation of the sequence alignment, where the algorithm returns an undefined value when it wasn't initialised as expected.

Overall, it was a pretty successful approach although sometimes it slowed down the creative process and was more time consuming than a direct specifications implementation.

Known issues

As the integration of the team project didn't work as well as expected (few delays in the business logic code, troubles with the hope server and the database initialisation there). As a result, we don't have a final working version of the browser but with a minor integration effort, this could result in a fully working browser.

What worked and what didn't

The individual elements of local creation and database retrieval, the API and algorithms of the middle layer and the requests of the AJAX front-end implementation. A difficulty was that the different Python environments didn't work so well together, especially considering the $PYTHON environment variables as we well as the Python virtual environments.

Problems and solutions

The problems described above should be further investigated to decide what needs to be fixed. One important issue was that we were unable to get the database working on the hope server, which resulted in a breaking point on our development workflow. A second problem was that the front-end layer didn't implement all the described API functions, as well as a few issues on the middle layer that affected the generation of code for the fron-end.

6. Alternative strategies

It would be really useful if we were able to invest on a more conceptual solution using already developed libraries and single page REST API approach using one of the modern Javascript frameworks.

7. Personal insights Code documentation

See the documentation of the API included with the source code in the folder middlelayer/doc/docs.md{pdf}.

References

FASTA format - Wikipedia. (n.d.). https://en.wikipedia.org/wiki/FASTA$_format.Retrievedfrom$
(Accessed on Mon, May 07, 2018)