Biocomputing II Reflective Essay

Wing-kin Chan

2022-04-22

I Approach to the Project

i Interaction With The Team

I created a WhatsApp group for our group to communicate, discuss ideas, issues, personal goals, and how our code may interact together as a whole. I firstly allowed others to decide which part of the project they most wanted to work on; Anthonia chose front-end HTML/CSS, and Python CGI scripting, Tiina worked on Python scripting for the business layer (blAPI), Kyan worked on an alternative front-end, and I was responsible for parsing the GenBank file, database creation, and creating database application program interface (dbAPI) scripts.

A few initial group meetings were held to set individual goals for each part of the project, however I left the members to be mostly very independent, and encouraged anyone to speak up if they disagreed with any of the requirements I laid out, if they had any problems, or required any help.

ii Overall Project Requirements

Project Outline Our main goal was to have a genome browser with various pages user could navigate to, and browse entries of genes stored in Chromosome 10. The pages include:

- A homepage
- A list-all page
- A search results page
- Pages for individual gene records

Homepage The homepage was to have ideally, a single search-bar, or some form of search function as the main feature to find various entries within the database via Accession, GeneID, Gene Product, and Locus. The search functions would use the search.cgi script that calls the blAPI which in-turn calls the dbAPI to query the database. Additionally, it would have a link for a list-all page.

List-All Page The list-all page would return a table, with clickable links to every record stored within the database. The list-all function would call the blAPI which in-turn calls the dbAPI. Clicking on links within the table will call the search.cgi and search by the feature that was clicked on. For example, clicking on 'seven transmembrane helix receptor' would return all entries in the database whose gene product was a 'seven-transmembrane helix receptor'. This page was to be generated by the listall.cgi.

Search Results Page The search results page, like the list-all page would return a table, with clickable links to every record stored within the database, whose Accession, GeneID, Gene Product, or Locus was similar to their respective search query. This page was to be generated by the search.cgi.

Pages For Individual Gene Records

The pages for individual gene records was to display information on the various annotations and features of the record. This includes:

- Accession number
- Date of the record
- Locus
- GeneID
- Gene Product (name of protein)
- Description (verbose describing function of protein/gene/locus)
- Source (organism record was isolated from)
- Sequence (genomic sequence including introns)
- Reading frame
- Coding regions and sequence
- Translation
- Complement sequence
- Restriction enzyme cutting sites

The code to display the sequence, coding sequence, and regions, aligned complement sequence, and aligned translation would rely on functions within the business layer to calculate the above, and returned a string with markers in a standardized format so that a function could insert HTML elements to display such features.

Moreover, some of these business layer functions such as computing the complement sequence, and coding regions can have long runtimes so once calculated, the blAPI can call the dbAPI to store these elements in the database.

iii Requirements For My Contribution

I was responsible for parsing the GenBank file, storing information within a MySQL database, and writing the dbAPI so other layers can retrieve data in a standardized format, and store data. The functionality of the dbAPI code was ultimately decided by the type of information we wished to be displayed on the webpage, what information the blAPI may want to store within the database, and how a user of our genome browser, may search for entries (Accession, GeneID, Gene Product, and Locus).

The parser was created to retrieve, and store information on records in a format similar to GenBank files so that referencing features and annotations would be similar in looking up the respective headers in a GenBank entry. This would aid in the development and understanding of how the code ran, essentially making the GBParser code self-documenting.

Also as an individual who felt more confident in coding, I openly offered any help to other development layers should they feel like they needed it. ## II Performance Of The Development Cycle

Our group was quick to come up with and agree upon a design. The development for the various layers was mixed. I feel that although there were a few setbacks, uncertainties, and small disagreements between the blAPI and dbAPI layers which interacted with each other for data retrieval and processing, this is only natural for a collaborative coding project, although minimizing these in future endeavors would be key. Initial stages of dbAPI, database creation, and parsing was rushed however, good communication between myself and Tiina ironed any bugs and missing requirements out very quickly.

Frontend development was slower than I would've liked, and I believe we are still yet to have a page that displays information on individual records working, nor do we have any CSS styling to make the webpage pleasing to look at and use.

III The Development Process

I aimed to have real data stored in the database, and returned to the business layer as quickly as possible. My reasoning behind this was so that the blAPI could have varied data to discern, if any, any bugs with their code. I admittedly did not read the coursework specification fully and used BioPython: SeqIO in my initial implementation of code to create the database. If I was to do this project again in the future, dummy data should've been stored in a mock-up database to check if dbAPI functionality, and dbAPI calling by the blAPI worked, whilst a parser was coded separately. In hindsight, although using BioPython SeqIO in the initial implementation of createdb.py sped up the start of the project, it only slowed me down afterwards, having to adapt the code to create the database to use my own parser.

The design and implementation of the dbAPI itself was relatively quick and simple. I knew how records would be searched for, and which data should be returned in what format after the end of the first week and writing basic dummy dbAPI code. The dbAPI code did not return much data to the blAPI but it gave both layers a good idea of the structure of the data to be returned. On top of this, additional functions were added on-top of the base querying functions as needed by the blAPI.

Developing my own parser function library took around 2 weeks. In designing the parser library, I asked myself; how does oneself go about reading, understanding, and extracting information from a GenBank file as a human? To identify individual records, we look for the 'LOCUS' header, and the '//' header that shows us the start and end of a record respectively. Individual annotations have their own headers for example 'ACCESSION' and if it continues across multiple lines, these lines are prefixed by a blank space. In this sense, the development of the parser library was very linear; I had a clear goal and visualization of how it should work laid out and I stuck to it. Within the library, there are various subroutines that went through several iterations and various implementation strategies which could be seen as branches off the main direction however each subroutine had clear end goals in terms of functionality.

IV Code Testing

Development and testing of the parser library (GBparser.py) was performed locally in VSCode. As the parser library was modular with individual subroutines, these blocks could be tested singly to ensure functionality and make debugging easier. Small objects were created in Python to imitate parts of GenBank files were created during initial subroutine testing. Individual records, or a small number of records from the original multi GenBank file were copied into a separate text file to test real world functionality of the parser library. Features such as class **repr**, **str** methods, and docstring tests were used to aid in testing and debugging.

Development and testing of parsing the GenBank file to objects that PyMySQL could call and dump into a database was also performed locally. This ensured data quality and integrity and did not require constant bandwidth usage by separating data handling from database creation. The first 95 lines of code in createdb.py was to parse the GenBank file. The latter part was to actually upload the data.

The dbAPI was mainly developed locally, and Jupyter Notebooks was used to test the code as this had a stable connection to the database.

V Known Issues

The dbAPI, and createdb/GBparser codes all function and have successfully parsed data, created a database, queried the database, and returned data to the blAPI. It is worth noting that the parser can not parse verbose miscellaneous feature qualifiers due to the feature regex searching for '/[a-zA-Z]' to denote the start of a qualifier and "‘' and whitespace denoting the end of a qualifier. However the '/[a-zA-Z]' pattern can come up in verbose paragraphs with '/' hence these qualifiers are not parsed. This is not an issue as we did not desire to store or display this information.

VI What Worked And What Didn't

I am painfully aware that development of the front-end HTML did not go smoothly although I did manage to get basic list-all, and search functions to work, the data displayed for individual records does not in anyway comprehensively meet our project requirements. This may have been due to the webpage being hosted on my Pandora account (individual responsible for the database side) so one could argue previewing, testing, and debugging the HTML/CSS and CGI might've been more difficult this way. On the contrary, no issues were communicated, and VSCode offers a LivePreview plugin to run HTML code and view changes on the go as one codes. The pages could've been developed using this with dummy data which was implemented very well by the blAPI (I cannot address how thoughtful the blAPI developer has been).

The blAPI and database/dbAPI is fully functioning in that they can retrieve, correctly process and return data for the front-end to utilize, however it is a shame to be unable to see this in a working webpage.

VII Alternative Strategies

In hindsight I should have suggested that the demo implementation of the website be used as a platform to develop the front-end rather than the front-end be developed separately to the updating, and implementation of the user-facing product.

Additionally, everyone worked on their individual aspects of their project with ruthless independence. It would've been better that if anyone needed help, others could have stepped in and worked on issues faced by other members in different project areas. This may lead to branch/merge conflicts but I believe this would've helped us get a fully functioning front-end.

VIII Personal Insights

- I recommend the front-end developer hosting the final user-facing implementation of the product.
- I recommend that should an individual become stuck or is having issues with their code, voice their concerns, ideas, and discuss with the group; it is a collaborative project after all.