Biocomputing II – Reflective Essay

1 May 2022
Amber Hilton

Approach to the project

Interaction with the team

This group worked consistently well as a team from the initial assignment of the project through to the last days. We met each week on a Tuesday and Thursday evening to develop the code for each of the application layers and also spent several days together in Birkbeck library. We kept a log of our progress in a shared google doc, and had an additional shared google doc to keep any research, code snippets, or lecture content that would be relevant for our work.

As a team we worked in a highly collaborative manner reflecting the pairing methodology adopted by many software engineering companies. We worked through each layer together to problem solve and develop the code often following a unit testing approach, where we alternated the lead developer role for each layer (Denzel - DB, Amber - BL, Hirushi/Farah - FE). This was the intended approach in order to facilitate the varying levels of programming knowledge in the team, as well as to build an enjoyable environment in which we could learn from each other.

Overall project requirements

In the initial phase of the project we spent time together as a team to understand the requirements of the application from the brief provided. We leveraged a virtual whiteboard software to mock up a rough wireframe of what we expect the front end to look like, to map its functionality, and the data that would be required.

Once we had the initial mock up of what we might expect the application to perform, we studied the genbank file in its original format and the data that was available to us, mapping this to the data that we were looking for. This analysis provided extra context to the requirements from which we itteratied on our wireframe model for the front end.

The final step in the requirement gathering phase was to detail the non-functional requirements which would enable the data from the genbank file to satisfy the requirements of the front end web application. This included extraction, transformation and loading functions, as well as data enrichment for the restriction enzymes and codon patterns. These functions were declared in the Database API dummy file and the Business Logic API dummy file.

Requirements for my contribution

The requirements for the Business Logic layer were defined as part of the non-functional requirements and captured in the Business Logic API dummy file in the early stages of the project.

To define the detailed requirements for the functions in the BL API layer it was necessary to fully understand the data available, as well as the required end format of the data in the web front end and perform a delta analysis. It was obvious from this analysis that the data needed to undergo cleaning, various transformations, and data enrichment. The definitions of the functions continuously evolved as we came to understand more about the data in the genbank file. Each of these functions were documented in the dummy API code and would be used later on in the project to act as a foundation from which to build the production API.

Performance of the development cycle

As was suggested, we allocated a lead developer role to each member of the group corresponding to different layers of the web application, however we chose to work through each layer together as a group of 4. During the development of each layer the lead developer would share their screen during the virtual working sessions and was responsible for driving progress, writing and committing the code to github. In some instances another member of the team submitted code to github on the lead developers behalf as we were ramping up with our knowledge of Github. Towards the end of the project we had the opportunity to work together in person, during these times we worked independently on the development of our application layers constantly communicating progress and problem solving blockers together.

We started with the data layer which took a significant amount of time, but we found it was incredibly valuable to each have an indepth understanding of the raw data. Problem solving as a team in this section was efficient, we had a diverse set of knowledge in the team from biological understanding to database knowledge which meant that we made good progress early on in the project to establish the database. With everyone in the team having a good understanding of the raw data and how it was extracted meant that trouble shooting at layers higher in the stack was much easier.

The Business Logic layer and the Front End layer were developed in parallel using a unit testing approach. We chose to adopt this agile method rather than a waterfall development method as it enabled us to troubleshoot in much smaller increments. It also enabled the Front End lead developers to fully understand the inputs required from the front end to call the BL API, and the outputs from the BL API that would need to be presented.

It may be considered inefficient and unnecessary that we worked together on every layer of the application, however we found that the approach we took made up for this inefficiency by avoiding any conflict or incompatibilies in code and data between each layer of the web application stack. More importantly I believe we all learnt a lot from each other's skills and took significantly more away from the overall activity than if we had worked individually.

The development process

In the first week we established and agreed on a way of working, and I mapped out what a high level project plan might look like, and the timelines we expected to follow. We had originally allocated 2 weeks for each layer, which later proved difficult to meet. *Figure 1* represents a true reflection of the progress of the application development.

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
Requirement Gathering	░	▓						
Front End Wire Frame Design	░	▓						
Data Analysis (Genbank File)	░	▒	▓					
Setup GitHub Repo	░	▓						
DB dummy API	░	▓						
BL dummy API	░	▓						
Data extraction code			░	▓				
DB population Code (MariaDB)				░	▓			
DB API					░	▓		
BL API				░	░	▒	▓	
Front End CGI scripts				░	░	░	▒	▓

First Draft	░
Taking shape	▒
Completion	▓

Figure 1 - Project Timeline

The initial phase of the development cycle focused on design, we spent time in the first week reading through the project brief, understanding the data in the genbank file and mapping out the user journeys to define the requirements. We built a rough wireframe (*Figure 2,3*) for the front end views to visualise the user journeys, and the non-functional requirements and data that the user journeys would need.

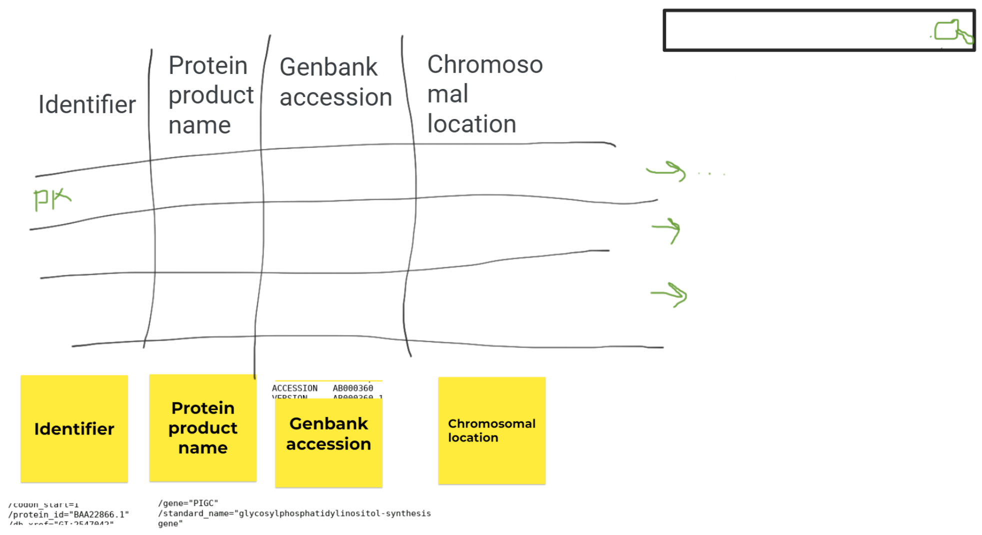

Figure 2 - User Journey 1, view all genbank records with minimum set of variables

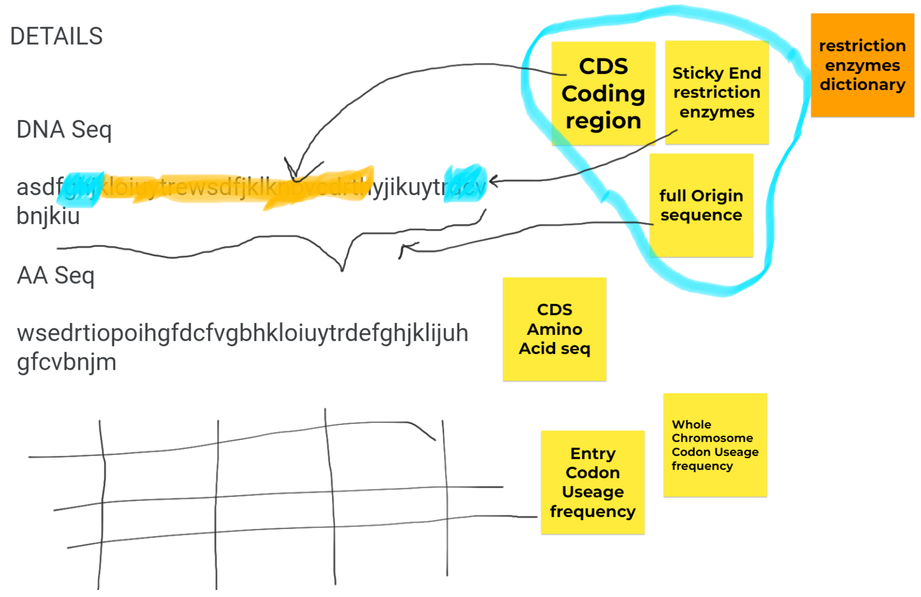
DETAILS
DNA Seq
asdfghjkloiuytrewsdfjklkjhgvcdrtfhyjikuytrdcv bnjkiu
AA Seq
wsedrtiopoihgfdcfvgbhkloiuytrdefghjklijuh gfcvbnjm
CDS Coding region
Sticky End restriction enzymes
restriction enzymes dictionary
full Origin sequence
CDS Amino Acid seq
Entry Codon Useage frequency
Whole Chromosome Codon Useage frequency

Figure 3 - User Journey 1, view single genbank record with more detailed variables

We then ensured that we were set up and all confident with using GitHub as a software repository to develop our code and collaborate effectively. Establishing a method for code management was key to having a strong foundation for getting started quickly.

To approach the DB layer we first set out to gain a full understanding of the raw data, and how we could best extract the data needed. We built a data model and defined the data types, and relationship between different attributes (eg. identifying accession number as the primary key, and which data points could be null). From here we extracted most variables using a regex itterator, and the origin was extracted using BioPython. Extracting the data took the longest, following this we quickly populated the database and wrote the DB API.

The development of the data layer, outlined in red in figure 1, was complex and overran beyond the project plan. To adapt to this we began to focus on the BL API and Front End in parallel leveraging the dummy DB API. Running this development work in parallel gave us a good foundation for when we began work on these layers later on, and it was possible to accelerate development in the last couple of weeks before the deadline without any significant pressure.

Once the database was populated and the DB API was complete we could focus on the BL API and the front end code. It was useful at this point to go back to our wireframe and check against the requirements. I developed each feature of the user journey as a function in the BL API using notebook as a workbench for testing functions and building up the code. I also wrote a basic HTML script to test the feature against the requirement.

Once each requirement had been met to satisfy the user journeys of the web application we could focus on the aesthetics of the front end, converting code from basic HTML to CSS.

Code testing

As a team we followed a unit testing approach, testing the smallest testable features of each requirement as they were developed. We deployed this approach for both the extraction of variables from the genbank file, for each ETL function written in the BL API, and each piece of information displayed on the Front End web page. This approach was particularly useful for troubleshooting and allowed us to avoid wasting time developing a function or extracting a variable which was incompatible with layers either side in the stack.

A specific example would be that of the accession number, once populated in the database we write a short code in the DB API layer to extract the accession number, another function in the BL API layer to simply call the DB API function, and then a short piece of HTML code to display the accession number on the Front End Web Page. Once this was working we repeated this process for each variable until we were able to display a full det of detailed information for each gene on a web page.

We found this approach very easy to adopt and digest what initially seemed a large task into actionable steps.

The most challenging aspect of testing we found in the project was the inconsistencies in the genbank file. It took a considerable amount of time to clean the data, but given the very large data set it often surprised us with a new unexpected result in the data which would impact code dependent on data types. Additional time to write testing and debugging code to filter for unexpected data would help ensure the quality of the web application.

Known issues

The variability found in the genbank file, and the rigidity of the code that parses the file to populate the database means that in some situations we may find anomalies presented in the front end (eg. where an entry has overlapping CDS regions). To address these issues a more sophisticated parser would need to be developed and a deeper understanding of all common issues in the genbank raw file.

What worked and what didn't - problems and solutions

We used two different methods for extracting data from the genbank file, Regex and Biopython. We had to do this because using Regex to extract the origin data was using considerable processing power and would often cause server issues. BioPython easily extracted the origin data and so we created a separate list for this data. This worked as a method however I suspect there are more elegant ways to approach this problem, the major flaw we found with this method was that BioPython was quite sensitive and if the exact format of the raw genbank file was not preserved we would quickly see errors. The potential fragility of this code gives me doubt with the method we took.

One of the bigger obstacles we faced as a team was the use of GitHub, the majority of the team were new to using the software management tool and as such we experienced several instances of overwriting each other's code and losing code all together.

Alternative strategies

As a group we prioritised working on the data layer first and building up our design from here. An alternative strategy would be to have a more design heavy approach and define the requirements to a greater depth - this would in turn inform the requirements from the BL layer and would give us direction when building the genbank parser. This alternative approach would possibly lead to other challenges with regards to designing features that are impossible to fufill with the available data.

Personal insights

I am very aware that we were lucky to be on a collaborative team where everyone was willing to share their time and knowledge to help everyone else. This was the main factor which meant that this project was enjoyable and challenging without being overwhelming.

From this my major take away will be, never to underestimate the importance of taking time at the beginning to build trust and establish common goals and ways of working.

The benefit of being placed in randomly allocated teams is that we had a very diverse set of experience and knowledge, this is something I believe we are often not exposed to in our daily jobs. It was interesting to see how differently we each approached the same problem and how this often was the reason we so quickly foudn the cause of errors in our code, or found alternative methods to writing a function.

However working as a team of 4 on each task felt inefficient at times, and possibly breaking into two sub teams could have been a more effective use of time. Although this would have required greater project management and coordination.

Working together in person in the Birkbeck library enabled us to progress much faster, it was interesting to observe this and on reflection doing more of these sessions earlier on in the project would have given us more time at the end to work on known issues and add extra features.

Due to the varied and generally limited programming skills in the team I found that we generally took a trial and error approach, which was often cumbersome and demotivating. I am incredibly grateful for the team and the attitude to genuine collaboration which enabled us to complete the task despite these challenges.