A Toolkit For Reporting On Metagenome Assembly Quality

Final Report for CS39440 Major Project

Author: James Edward Euesden (jee22@aber.ac.uk)
Supervisor: Amanda Clare (afc@aber.ac.uk)

1st May 2016
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name ... James Edward Euesden ...

Date ... 4th May 2016 ...

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name ... James Edward Euesden ...

Date ... 4th May 2016 ...

Acknowledgements

I am grateful to...

(Unfinished) Friends, family, peers, lecturers, university

I'd like to thank...

specifically thank supervisor, close friends in and out of department

Abstract

The project topic outlines the issue that there is a need for toolkits to report on the quality of metagenome assembly data, and that one such tool could be created for this project as a software engineering issue. The report looks into creating the tool ‘Khimeta’ that attempted to meet this problem, the background research done to prepare for it, issues and interesting points during implementation, expectations, testing of the application, the lifecycle model of development and an evalutation of the process and development of the application after completion.

CONTENTS

1 Background & Objectives **1**
- 1.1 Introduction . . . 1
- 1.2 Background . . . 1
  - 1.2.1 Metagenomics . . . 1
  - 1.2.2 Understanding Quality . . . 2
  - 1.2.3 Existing Software . . . 3
- 1.3 Analysis . . . 3
  - 1.3.1 Quality assessment . . . 3
  - 1.3.2 Quality report and data input . . . 6
  - 1.3.3 Implementation - 3rd Party vs Self-Created . . . 8
- 1.4 Process . . . 9
  - 1.4.1 Scrum . . . 9
  - 1.4.2 Extreme Programming . . . 10
  - 1.4.3 Pomodoro Technique . . . 10
  - 1.4.4 Project blog . . . 10

2 Design **12**
- 2.1 Overall Architecture . . . 12
  - 2.1.1 Choice of technologies . . . 12
  - 2.1.2 MVC Framework . . . 14
  - 2.1.3 Naming Conventions . . . 19
  - 2.1.4 User Input . . . 19
  - 2.1.5 Directory Structure . . . 19
  - 2.1.6 QualitySummary and Results . . . 20
- 2.2 User Interface . . . 21
  - 2.2.1 Welcome . . . 21
  - 2.2.2 List . . . 22
  - 2.2.3 Results . . . 24
  - 2.2.4 Error . . . 28
- 2.3 Support Tools . . . 29
  - 2.3.1 Version Control and Continuous Integration . . . 30

3 Implementation **32**
- 3.1 Development Environment . . . 32
- 3.2 Features . . . 32
  - 3.2.1 Reading User Input . . . 32
  - 3.2.2 Counting GC Content & Percentage . . . 34
  - 3.2.3 Displaying GC Content percentage . . . 35
  - 3.2.4 Finding Open Reading Frames . . . 36
  - 3.2.5 Displaying ORF Locations . . . 40
  - 3.2.6 Superframe Comparisson . . . 42
  - 3.2.7 Implementation Review . . . 42

4 Testing **44**
- 4.1 Overall Approach to Testing . . . 44
- 4.2 Automated Testing . . . 44

4.2.1 Unit Tests 44
4.2.2 User Interface Testing 46
4.2.3 Manual Testing 48

5 Evaluation **50**

Appendices **51**

A Third-Party Code and Libraries **52**
1.1 Tabs for results 52
1.1.1 HTML 52
1.1.2 CSS 52
1.1.3 Jquery 54

B Ethics Submission **55**

C Examples and Extras **56**
3.1 Example User Story Breakdown 56
3.2 Artificial Test File 57

Annotated Bibliography **60**

LIST OF FIGURES

1.1 A very small example of a number of reads being taken into an assembler and the output of a contig where there may be a chimera. 2
1.2 A small example of the contents of a FASTA file, with very short contigs. The example is just to show the format of how multiple contiguous reads are stored in the same FASTA file. 6
1.3 A use case diagram demonstrating the expected functionality for a user based on the functional requirements laid out in this section. 8

2.1 Adding an object to the Model via the Controller to be accessed in the View. The View here is 'list', which is a Thymeleaf template that will dynamically build the page using the data from the Model added here. 13
2.2 Including a thymeleaf fragment in a page. You can see how the call is made from one 'th:replace' with the name of the html fragment file and then the fragment to be included from that file, in this case 'inspectbox'. 13
2.3 Including a thymeleaf fragment in a page. This is the declaration of the fragment being included in the previous figure, declared as such through 'th:fragment'. . . 14
2.4 The MVC framework the application is designed upon. The data flows between the Model and the View via the Controller, based on the user requests and interaction. 14
2.5 Class diagram for the 'Welcome Page', and what classes are used upon a Request for the page. 15
2.6 Class diagram for the 'List Page', and what classes are used upon a Request for the page. 16
2.7 Class diagram for the 'Results Page', and what classes are used upon a Request for the page. 17
2.8 The layout of the directory structure for the application. 20
2.9 The 'welcome' page that greets the user upon requesting the home page of the web service. 21
2.10 The 'list' page that is displayed after a user submits their (valid) assembly data in FASTA format. 22
2.11 When a user clicks to inspect a contiguous read, this menu appears giving them options to alter the parameters of the quality assessment inspection. 23
2.12 If the user clicks to see an explanation of the process, they can click the text that says to 'Show explanation', and this text appears. 23
2.13 The GC result tab shown to the user on inspection of a contig. The browser has been zoomed out in order to display the page in full. 25
2.14 The Open Reading Frame results view tab. 26
2.15 The Open Reading Frame results view tab, displaying that when a user clicks on a different ORF Location, the highlight on the frame chart and list changes to reflect this. 27
2.16 The 'Superframe', a comparison between the ORF Locations within the contig and the GC content windows that are above the threshold.. 27
2.17 The result of when a user clicks to view a window data, they can see the particular percentage of the GC content of that window, and where that window starts and finishes, if they wished to inspect the contig themselves using those numbers. . . 28

2.18 If an error occurs as the user is browsing the web service, i.e. they try to access a page when they do not have the required data submitted, they are presented with this page informing them of an issue. 29
2.19 The Jet Brains IntelliJ Integrated Developement Environment (IDE) I used for developing my application. 30
2.20 A countdown of the pomodoro tomato timer on `http://www.tomato.es/`, used for breaking up development time and keeping track of work done over time. 30
2.21 A few of the commits made to my major project repository. 31
2.22 Examples of the continuous integration on CodeShip running my tests every time I committed to my repository. 31

3.1 Part of the code for reading in a users data when they have pasted it into the text area of the web service. Deals with creating new ContiguousRead objects and adding them to a ContigResult every time it finds a new header for a contig (or reaches the end of the input). 33
3.2 The code for splitting a contiguous read into windows available for calculating the GC content window percentages. 34
3.3 The code for calculating the GC content percentages for each window passed into the method. 34
3.4 Extracting the data from the Model, provided by the Controller to the View, using Thymeleaf's inline tag to be able to use CDATA to convert the Thymeleaf extracts into JavaScript. 35
3.5 'GcContentViewData', the object provided to the View in order to display the GC content data to the user. 36
3.6 Finding all of the start and stop Codons from within the passed frame from the contig. 37
3.7 Extracting each frame and calling to run the ORF Finding process. 37
3.8 Getting the base pair characters of a reverse frame is as simple as a switch statement and building the reversed contig from back to front. 38
3.9 'Zipping' together ORF Locations from start and stop Codons within the frame. . 39
3.10 Finding if a click if within an ORF Location is as simple as going through the list of ORF Locations, only checking against those within the same frame as the click, then looking at whether the click is within the start and stop points of an ORF Location within the canvas, based on the size and location of where it was painted (for reverse frames, the start and stop points are swapped, as reverse frames are displayed in the same direction as forward frames, with the indexes in the right order for a reverse frame). 40
3.11 Formatting the characters from within an ORF Location into HTML tags to display the data in the way I wanted it to be designed, then inserting it into the div for displaying ORF Location information within the page. 41

4.1 An example of a unit test used in my application. 44
4.2 Results of running the set of unit tests developed for the application. 45
4.3 The test coverage of my unit tests over the developed application code. 45
4.4 The test coverage of my unit tests over the developed application code, broken down for each class. 46
4.5 The same page and results from the application shown in different modern browsers. 46
4.6 YSlow report after running on the Welcome page. 47

4.7 YSlow report after running on the List page. 47
4.8 YSlow report after running on the Results page. 47
4.9 Combining two species contiguous reads together at 50% of each of the file (first half one species, second half another species), we see a very obvious split in the GC chart. 48

LIST OF TABLES

Chapter 1

Background & Objectives

1.1 Introduction

The project aim was to create an application that could report on the quality of a metagenome assembly provided by the user, presenting them with feedback about the contiguous reads contained in their data. The requirements for the project topic were very open, as there are multiple techniques for attempting to report the quality of a single species genome assembly, and while some can be used for metagenome assemblies, it was believed that no one tool covered this area yet with numerous techniques. Likewise, the way in which the results could be presented to the user was not established and open to my own interpretation.

1.2 Background

Before the project began, my knowledge of metagenomics was very limited, close to none. However, I liked the project title and description and thought it would be an interesting and challenging topic to learn new domain knowledge, use different technologies and attempt to implement an application where I had to learn from the ground up. On top of this, I find DNA to be an intriguing topic even with my limited knowledge and I was curious to learn more as I worked on this project.

My first step was trying to understand what exactly it was I was expected to produce at the end of the project. As the requirements of the resulting application were so open, it was up to me to research what metagenomics is, what is meant by 'quality' within the subject, how this quality might be found and reported on, what technologies would be appropriate and what quality techniques could be used.

1.2.1 Metagenomics

Metagenomics is the study of environmental samples of genomic data where the contents of the data are potentially unknown and unclear. It has been described as 'Open-ended sequencing of nucleic acids recovered directly from samples without culture or target-specific enrichment or amplification; usually applies to the study of microbial communities.' [9] It can be used in the findings of what an animal gut may contain, what viruses are within a sample when looking into

outbreaks and finding what microbial communities exist in a sample area.

Metagenomics is a hot and interesting topic in the bioinformatics field, and its uses grow as more is learned, but there is the issue of quality, and how a metagenomic sample should be processed. To help me better understand the project task, I read articles that attempted to provided ways of analyzing metagenomic data to get the best quality results at the end. [22]

1.2.2 Understanding Quality

Considering the nature of metagenomics and the unknowns, it becomes clear quite quickly that when a sample you have is run through an assembler in an attempt to create a genome for sequencing, without the proper tools to quality assess your data you cannot be sure if what you are creating is an actual thing that could exist in nature. The process of taking a sample through to sequencing with metagenomic data can be very error prone, leading to misassemblies with duplicate or short reads, or combining reads together to make chimeric contiguous reads (contig). [7]

A chimieric contig is an instance where an assembler has put together reads from a sample that it believed were part of the same whole, and yet were in fact of different species/sub-species, and so creates a contig that does not actually exist in nature. It can be understood then that if a user were to sequence this, unless that is the result that they wanted, it won't be of any use to them. Without the proper tools, how would they know that their assembly data contains chimeric contigs and are not just wasting their time and money?

We can visualize this through the diagram below, where we take a number of reads, put them through an assembler and the output is a number of contigs that the assembler believes it has created correctly, but may in fact be chimeric.

Figure 1.1: A very small example of a number of reads being taken into an assembler and the output of a contig where there may be a chimera.

If we imagine each of the colours representing an individual species (though in reality, there would be no colours, only the characters of data, and potentially hundred to thousands of different species and sub species in a sample) we could, by eye, see how we could extract some interesting genome data, for example taking all of the green in the order they are presented and finding the 'green' genome. As naive as that is, it serves then to demonstrate that if we look at the output of the one contig shown in the figure, we can see how the colours are aligned in a way that they are mixed.

It is possible that the resulting contig is a valid genome that could be found in nature, where the

particular sequences in the different coloured sections are shared between each colour 'species' and so this contig could work. However, it is also likely that the different coloured sections are widly different than the natural connecting colours if instead this contig was made up of a single colour (in this figure, all yellow, all blue, etc). This would then be a chimera. This demonstrates that without some quality assessment tools to attempt to report on where these misaligns have occurred, a user may never know if what they have is good quality data or not.

1.2.3 Existing Software

There are a number of tools I discovered in my background research that attempt to aid in the quality assessment of metagenomic data, in particular MEGAN, a 'next-generation metagenomic data, called MEtaGenomeANalyzer', which attempts to do a taxonomical analysis comparison to known reference data [5] and PRINSEQ, a tool which provides 'summary statistics of FASTA (and QUAL) or FASTQ files in tabular and graphical form' [16]. When considering what my own application should do, I looked at the techniques used for PRINSEQ most, as these seemed to match up with what I thought would be useful to a user for my own application, and from discussing the topic with my supervisor I found techniques such as the GC Content distribution could be a good place to start.

It was not just tools for metagenomic quality assessment I looked into. I also found the NCBI database and their BLAST tool [4], and kept in mind these may be useful as I progressed through my applications development, and through reading an article that discussed the advantages of k-mer frequency analysis for quality assessment, I looked into the Jellyfish [11] and BFCounter [12] tools for just this role where I could potentially consume the output of their processing, although they took a step back in my mind while I considered what it was I actually wanted my application to be and worked out the requirements.

1.3 Analysis

After understanding the project topic and problem a little better, I decided upon a number of requirements of the application to begin with. Some of these were definite goals, some stretch goals and some future development tasks if I were to finish all else or were to continue the application after the project deadline. I broke the problem down into its two core components steps, the analysis and the report. I felt that the resulting list that can be seen below was enough to work with based on the knowledge I had gained from my background reading and what I thought would be appropriate for the time allotted for the project.

1.3.1 Quality assessment

The quality assessment had to use a number of methods suitable enough to produce some data or statistics that could indicate to the user a measure of quality of their assembly data. For this end, I decided upon a number of objectives:

1.3.1.1 Contig length

'Give the user control over the minimum length a contig should be to be considered.'

This measure would be good in displaying where an assembler could not find any reads to match with a single read and so didn't do anything more with it than output it as a contig of an individual read length. This would most likely indicate that it is of no use to a user, as a contig the size of a read length is unlikely to contain any useful genome data. Allowing a user to set the minimum length threshold lets them set their own size, be it the known read length size of their data, or a size they think would be appropriate to start seeing some usable data from contigs with length over a particular amount/number of read lengths.

1.3.1.2 Number of unknown characters

'The application should count the number of unknown (N) characters within a contig.'

When an assembler cannot understand what to do with a character, or a sequence of characters, it may insert an 'N' character. Indicating to a user how many of these exist in a contig, and what percentage of the whole they make up is a helpful indication of whether their data is of good quality or not, with the less or no unknowns the better the quality.

1.3.1.3 GC Content

'Conduct a GC Content percentage analysis in sliding windows of a size set by the user.'

The GC content is the percentage of 'G' (guanine) and 'C' (cytosine) characters within a sequence [18]. When the application takes a contig, it should break the contig up into windows of a size set by the user. If a contig was of length 30,000, they might break it into 100 windows of 300, for example. These window sizes should then have their GC content percentage calculated, and the mean of the entire contigs GC content worked out. Using these values it then becomes possible to detect potential anomalies in the percentages of individual windows that have a percentage value drastically different than the mean.

1.3.1.4 Open Reading Frame Locations

'To confirm if a GC window is outside of the mean by misalign, Open Reading Frame (ORF) Locations should be found where the GC content would naturally be higher in protein coding regions.'

This addition to the requirements came later in the projects development as I came to understand GC content and how a window that looks like an area of a misalign may actually not be. We can look at a contiguous read and find protein coding regions through the use of Open Reading Frames [20], where the GC content percentage is often naturally higher than outside of these regions. [23]

An ORF Location is found through looking for where a location starts with ATG and where it ends with TGA, TAA or TAG. There are 6 frames to be found in sequences, where three are 'forward' and 3 are 'backwards' with the opposite characters of the base pairs to the forward frames. For both forward and backwards, the first frame is the original sequence, the second

frame begins one character into the sequences and the third frame begins two characters into the sequence. A very small example just for basic understanding is provided below:

```
Sequence:
TTGATGGCGCATAG

Frame \#1 (fwd):
TTGATGGCGCATAG
Frame \#2 (fwd):
TGATGGCGCATAG
Frame \#3 (fwd):
GATGGCGCATAG
Frame \#4 (bck):
CTATGCGCCATCAA
Frame \#5 (bck):
TATGCGCCATCAA
Frame \#6 (bck):
ATGCGCCATCAA
```

By finding the ORF Locations, we can use these in the quality report to match up to windows that may at first seem like anomalies in the contig where it is instead actually a natural occurrence. There is a preexisting tool for this ORF finding functionality by NCBI [19]. While it might not be the case that an ORF Location matches with an out of threshold GC content window, or that they do match but it is still in fact an anomaly, it was a requirement of the application I wished to include in order to give the user another tool to use for inspecting their data.

1.3.1.5 K-mer frequency analysis

'The application should conduct a k-mer frequency analysis, or use output of k-mer frequency analysis tools.'

Conducting k-mer frequency analysis in window sizes has a similar output as GC content, except we look for the frequency of particular 'k-mer' (where 'k' is a number of how many characters to be considered, e.g. 3mer 'ATG', 4mer 'ATGA'). Through measuring the frequency in windows we could see if there was an even distribution of frequencies across a contig, and if any windows had large changes in the frequencies that could indicate a potentially bad quality contig. [8]

This was set as a stretch requirement, as the process of efficiently conducting k-mer frequency analysis and the research into the possibility of writing my own software to do it or consuming output of another application was expected to take over the time for the project, after my background reading and understanding was completed and the other features previously mentioned implemented.

1.3.1.6 NCBI reference data

'The application should compare the contiguous read data to known reference data in the NCBI database with BLAST.'

Using the known reference data, much like with MEGAN, it would be possible to see if any of the contigs in the user data match to known reference data and so can be considered good quality. This was a stretch requirement though, with the idea more in mind for if the application were to be continued to be developed after the project deadline.

1.3.2 Quality report and data input

1.3.2.1 Read user input in FASTA format

'A user should be able to paste or upload their data in FASTA format.'

The FASTA format is a commonly used format for sequence data. It is 'a text-based format for representing either nucleotide sequences or peptide sequences' [24]. It is formatted in the way that can be seen in the figure below. I set myself the task for allowing the upload and paste of data, beginning with pasting the data into a form, as uploading data could add a larger realm of security and data size issues and so I wanted to have the simplest thing possible first.

```
>NODE_1_length_180_cov_7.511111
CTCCCTCTTTTTTCGGATATGCTGGTGATTTGGCGACGAACAGTATCTGGGCAGCTCTAT
TTATCTCGAAGGCAGTTTATTTAATAAAATATTGTGGCGAAAATATTCACTCACTCCGTC
GGTTTGCCAGTTTCCTATCTGGTCGGATAAACTAGTGATTTGGCGACAGGCGGTATCGGG
TAAGCTTGTTTTATCCCGACGATCAGTGCCTGAAATACTGACAGATACTG
>NODE_3_length_74_cov_9.391891
ATTAAAAAGGGTGTAAATTGAATATTATAATATGTATAGGAGTCGGATAGTCGTGACCTA
GTTTGTGTCGTCTGGTCAACTGTTGACGAGAAGCGTGTAAAGAGATCGCGTGCTACACAA
TCGA
>NODE_9_length_70_cov_41.885715
ACGTGACGTCATGGAAGAATAAACACACAATAATATAATAATATTGAATTAGATATAGTA
TAATATTGTAATTATTTTGAAATGTTTAATTTGGGTTTAAATTTGTCAATAAAATATTGT
>NODE_16_length_154_cov_11.662337
TGCCTATTACAGGTCTTTCTTTAGGCCTGAAATTTTATCTAAATTATTAAAGATACATTA
ATTAACATATCCAGACCATTAATGAACTCTCGATGTCTACACTAGCCTTTGATATTGACT
GATAATTAAATCAAAAAATGGATAATCGAAACTATGCTTGTTATTATACTGACTTTATTA
ATTTCTATTCCTTAATTCTTGGTt
>NODE_23_length_138_cov_4.391304
GTACCATCCAAAGTACTAGACATTGTTTCTTGTCAGACCCAATTTTACATCACTGATATG
GGGAAAGCAGATTAGAAAATTGGTGAGGACCTTTTCTCTGCCCATAGAGGAAAGGCAACC
GGGACCTTCTGGTAACTGTGCACCCCACCCTCCTCCTTTACTTATTAaacaggcatctac
taagttga
>NODE_32_length_78_cov_4.871795
ACAATTTATCATAGGTCCGAGTAGTCATATTAAAACAAACACAAGTTTTGGTCCAATCTT
TCCGAAATTCGAACTCATCCTTCCTTAGGGTGTAACGGTGATATAATCAAACGATTTTTT
AAATCGGa
```

Figure 1.2: A small example of the contents of a FASTA file, with very short contigs. The example is just to show the format of how multiple contiguous reads are stored in the same FASTA file.

For the FASTA format my application accepted, I planned for it to begin only taking into account ATGC & N as characters to deal with, where N was 'unknown/unclear' and the other characters were valid. Uppercase and lowercase characters would be preserved in any output display of the application, but capitalization would not taken into account for any of the processes as they did not make a difference to the quality assessment techniques.

1.3.2.2 Display a list of the contigs

'A user should be able to see the contiguous reads listed in their FASTA data.'

I decided it would be a good idea to display to the user the list of contigs that their file con-

tained, with small bits of relevant information, such as the size of the contig, the header of it and the previously mentioned 'N' count and percentage. From here a user would be able to select a contig they wish to further inspect for quality issues.

1.3.2.3 User control over GC Content, ORF Location and k-mer frequency analysis parameters

'When conducting a quality inspection, a user should be able to adjust the parameters of the methods based upon their expectations, data input and choice.'

Since the data uploaded by a user would have been assembled from reads of a particular length, the appropriate size of GC content and k-mer frequency windows, minimum size of ORF Location lengths and minimum length of contigs should be set by the user. It could have been a possibility to guess what a good size would be for the user (i.e. find the shorted contig in the data and consider that that might be the size of a single read, and suggest length parameters based on that).

1.3.2.4 Visual reports of the quality inspection methods

'A user should be able to see a visual representation of a contigs GC content and any anomalies.'

In order for a user to be shown whether their data is of good quality or not, visualizations would be required that highlight any problem areas, or if areas are not highlighted, at least give them a broad sense of the layout of the results of the quality inspection to allow them to analyze the results from themselves. This should include displays for GC content in windows, ORF Locations broken into individual frames, k-mer frequency analysis in windows and a comparison shown between the GC content and ORF Locations.

I did consider how I could potentially provided a quality confidence factor to a user, or how I might demonstrate to them in a statistical way that their data was of good or bad quality. However, with the amount of methods that could be used, the unclarity of the data itself and the users expectations, I felt that it would be better to instead provide the user with visuals to analyze the results of my application themselves, and I would instead try to highlight problem areas without necessarily saying clearly 'This is/is not a problem area'.

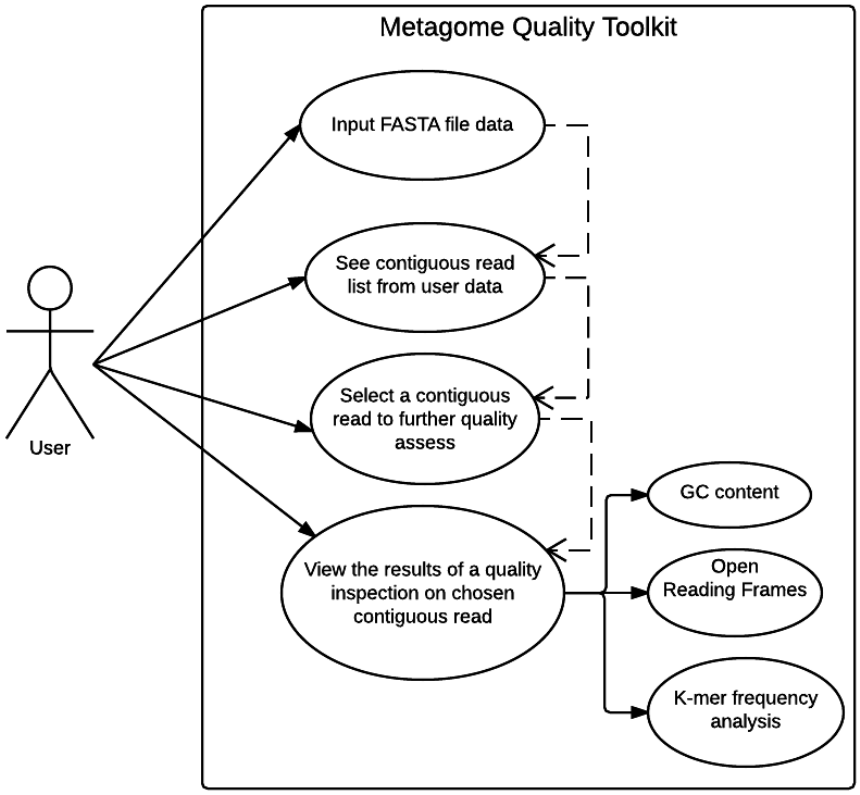


Figure 1.3: A use case diagram demonstrating the expected functionality for a user based on the functional requirements laid out in this section.

1.3.3 Implementation - 3rd Party vs Self-Created

Based on the requirements I had selected, I decided that I wanted to write my own software for each functional component. While solutions for each component exists individually, I felt there would be more benefit to a user if I could produce my own software to support the functionality, over requiring them to use third part software in order to use my application too.

Additionally to this, I wanted the technical challenge of writing my own software, and enabling the application to be maintainable and expanded upon in the future rather than relying on third party support applications in order to process a users input. The process of developing the application from scratch would also give me the opportunity to learn more about the domain and technologies for development than just consuming output of other existing applications, even if achieving the end result would take more time.

While I knew the project would come to a close on development at the end of the semester, I also wanted to envision this as a project that could be taken forward in the future to be further developed and maintained. For this reason it also made sense to develop my own application without consuming output from third party software, excluding the FASTA files from an assembler. If one of these third party applications was no longer available or their output changed, it would render my application unusable or require more future modification to adapt to changes by outside sources.

1.4 Process

An agile methodology seemed most appropriate for this project development, due to the unclear requirements (based on my lack of understanding and the open tasks at the start of the project) and would allow me to break down each requirements into stories, and then into tasks and get a thin slice of the application done quickly, and build outwards from there, always ensuring I had some finished product at each functional completion. While my project could well follow a plan driven approach, such as waterfall or the spiral model, I believed that an agile approach would be more suitable for developing it.

I chose to use Scrum for my framework, and used aspect of Extreme Programming for my daily development cycle, adapted to a one person project. Considering the nature of my projects requirements, I considered if I should instead use Feature Driven Development, but decided upon Scrum due to my experience with it through my industrial year and in that I could see how I could break the requirements down in to user stories and tasks.

1.4.1 Scrum

Scrum is an agile framework for developing complex products [17]. In brief, it works in iterations (Sprints) where work is broken into user stories and tasks for those stories, there are key members of the team who have designated roles (Engineer, QA, Scrum Master, Product Owner, etc) and where each Sprint has meetings involved in it to help the flow of work: Sprint Planning & Sprint Retrospectives, Sprint Review and Daily Stand-ups. I decided that adopting this framework would help me structure my weekly work and aid me in planning and development for the project.

As a one-person project, a number of the components of Scrum either had to be dropped or adapted, for example, the roles of team members were all adopted by myself rather than having actual team members. I worked on a weekly Sprint cycle, where at the beginning of every Tuesday a new Sprint would begin, and I would review the work done and have a Retrospective with myself and plan out the next weeks tasks, often through talking with my supervisor. As for Daily Stand-ups, I was able to carry these out through holding them with my peers, Alex Jollands and Sion Griffiths. Between the three of us we would meet in-person or using Internet calls on weekday mornings at 10am and discuss our work from the day before, the planned work for the current day and anything we wanted to soundboard off one another.

The Scrum framework had me taking my initial planned out requirements and turn them into Stories with their own tasks. Stories are structured in the way of:

‘As a <*type of user*>, I want <*some goal*>so that <*some reason*>.’ [14]

Once a user story has been created, I would then consider it and see if I could break it down into a ‘narrow slice’; the simplest, thinnest possible thing I could do for a feature implementation that would produce results. There is an example of this in Appendix 3 ‘Examples and Extras’ under ‘Example User Story Breakdown’. Breaking down the requirements into user stories and then tasks helped me focus on what was important, and gauge what needed to be done and how much effort vs time it would take.

1.4.2 Extreme Programming

For the day to day development outside of the Scrum framework, I decided to adopt a modified Extreme Programming (XP) approach [13]. While I couldn't do all of the XP techniques (pair programming, for example), I strongly took the elements of Test Driven Development (TDD), refactoring, simple design (coding only what needs to be done based on the TDD and refactoring) and continuous integration. I felt that using a combination of Scrum and XP techniques was a suitable process for my application, and would aid me in my design, implementation, testing, meeting the requirements and focusing my motivation.

The motivation for taking the aspects of TDD and Refactoring were that I could build and design my application iteratively, as I understood and refined the requirements and functionality of the application as I went, while still having a usable application from an early stage that could be developed upon in each Sprint.

TDD ensured that the application would be developed in the simplest way possible, and always meet the requirements currently tested for by doing the unit test development up front. Refactoring the work as I went and where needed meant that I would keep my code maintainable while developing in this way. The end result through selecting this methodology was to aim for a robust, maintainable, well-design application with test coverage for all functionality and programmed efficiently through proper time management. Using continuous integration meant that I could be sure that whenever I completed a new bit of code, it would be run against all existing tests to check that nothing had broken and was impacting previously working functionality.

1.4.3 Pomodoro Technique

On the matter of time management during development time, I found it useful to break my work into small cycles using the Pomodoro technique [2] to increase my focus and productivity. The aim of the pomodoro technique is to spend 25 minutes with full focus on the work, and then spend 5 minutes to stop and have a break. They are also useful for keeping track of work done and visually seeing how much time has been invested into the project for the day. To record my pomodoros I used `http://www.tomato.es/`.

Due to the short length of each pomodoro, it was also seen as an effective method for squeezing in work where there was time. By quantifying time into small slots, it made it possible to consider how much time I might have between other events in the day and work out where it could be possible to put in a single pomodoros worth of work. This is instead of perhaps not working because I felt I did not have the time, or working and eating into other daily activities if I didn't block out the time and be aware of when to stop.

1.4.4 Project blog

While developing my application I kept a blog about my project, any milestones I reached or interesting sections. While I did not designate a particular time to blog, I believed it would be a useful tool in reflecting on the work done when writing this report, something for my supervisor to be able to check upon my progress between meetings and give me the time to reflect on the work done and any upcoming tasks to help me mentally process where the project was in its life cycle against the planned objects. The blog may be seen at `http://users.aber.ac.uk/`

`jee22/wordpress/`.

Chapter 2

Design

2.1 Overall Architecture

The design of my application was evolutionary as I was developing it using XP in an agile way. This meant that each step had an initial design and was built upon every time a new feature was added, being refactored along the way. There were a number of prototypes made during the course of the project's development, and this chapter will discuss the design of the end product, and go into detail where the design may have been previously different.

The resulting application evolved into a web service, with a Model View Controller (MVC) framework and resources structured for maintainability. Following XPs guidelines, at each step of the way the applications design was made as such to be the simplest yet most maintainable it could be through refactoring, cutting down on duplicate code, structured logically and choosing smart data structures and Objects to represent aspects of the application and quality results.

2.1.1 Choice of technologies

At the beginning of the application, I felt that the application should be programmed in a language that would be able to be ported across multiple platforms for used by anybody who wished to use it. Additionally to this, I wanted to have ways of representing the components of my application as Objects and would need some way of presenting a UI to the user, even if my initial application would only output command line results, in order to follow the XP value of Simplicity (YAGNI - "You Ain't Gonna Need It", until you do).

For this reason, I selected to develop my application using Java to begin with as it filled the criteria of being Object Oriented and was portable through using the JVM on different operating systems. I resolved that I would select a UI package to present the report of my results once some of the core functionality had been implemented. I considered instead using Ruby or C++, but due to my familiarity with Java I believed it would be a better choice to stick with what I knew. At this point, I did not consider having the application as a web service, and so using Ruby with Rails was not something I had considered.

As the application developed and I started reading in files and outputting results from the GC Content process, I found that I was having a hard time finding a quickly usable GUI I could work with for Java to display the results in a way that I wanted. It was at this point I started considering

a technology change to Ruby on Rails until Sion Griffiths, a peer of mine, suggested I could keep my application in Java and turn it into a web service using Spring Boot [15]. This seemed like the perfect solution to my problem, allowing me to generate my charts and results using HTML5, JavaScript and HTML5 Canvas, along with JavaScript libraries such as Plotly.js.

The conversion to Java Spring Boot was not a painful one, and only involved setting up a new Maven project and declaring it to run as a Spring Boot application in the pom.xml file, then copying over my previous code and structure into the new project, from there I could deploy the application as a web server using Tomcat and was back to my previous position of working out a UI. Thankfully, there is a package called Thymeleaf [21] that works with Spring Boot to allow access to Objects from within the Model of the Java code (placed there by the Controller) using the View dynamically as the View is generated through HTML templates and fragments.

```
model.addAttribute("contiglist", contigList);
model.addAttribute("discardedcontigcount", contigResult.getDiscardedContigCount());
model.addAttribute("contiguousread", new ContiguousRead());
return "list";
```

Figure 2.1: Adding an object to the Model via the Controller to be accessed in the View. The View here is 'list', which is a Thymeleaf template that will dynamically build the page using the data from the Model added here.

On top of using Thymeleaf for accessing data put into the Model, it has the additional benefit of using fragments that can be imported into different HTML templates. This design choice made it so I was able to cut down on writing duplicates of code. For example, I wrote the header and footer of the UI design in a Thymeleaf HTML fragment, and then on each page just need to call one line in order to import it into that page, rather than built the entire thing again. An extra benefit to this is that it allowed me to keep the HTML templates clean and easier to maintain by reducing their size and separating out different aspects of the UI. Having the header, footer and a number of forms in Thymeleaf HTML templates meant that these sections could be edited without impacting any page they are imported into.

```
<tr>
    <td>
        <div th:replace="fragments/qualityparameters :: inspectbox"></div>
    </td>
</tr>
```

Figure 2.2: Including a thymeleaf fragment in a page. You can see how the call is made from one 'th:replace' with the name of the html fragment file and then the fragment to be included from that file, in this case 'inspectbox'.

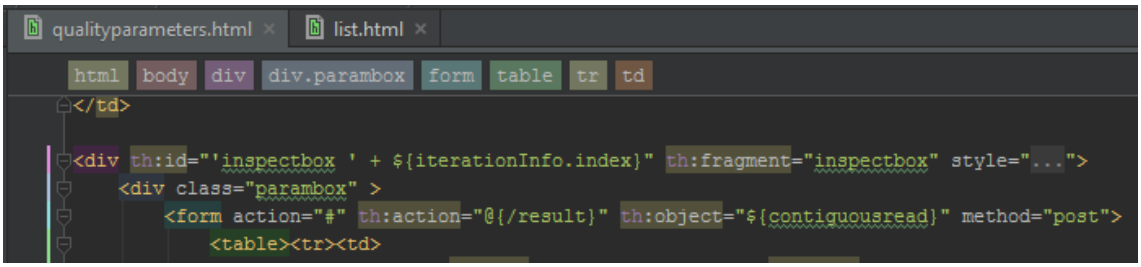


Figure 2.3: Including a thymeleaf fragment in a page. This is the declaration of the fragment being included in the previous figure, declared as such through 'th:fragment'.

For the GUI itself, I elected to use Plotly.js [6] for representing my GC content charts for the level of detail and control it gives a user over inspecting charts, large and small, that worked no matter what GC window size the user selected. For the ORF Locations frames charts I wrote my own code using JavaScript and displayed it with HTML5 Canvas, as it allowed me control over what should be shown depending what a user clicks upon and I could work with the data from the Model in ways I saw fit.

2.1.2 MVC Framework

As I was building an application with a GUI, the application was built using a Model View Controller framework design, separating out the components into their different types. This helped the separation of code and responsibilities. The class diagrams presented in this chapter are based on the final product design and based on a page per page view from the UI and what classes are used on the HTML request of that page.

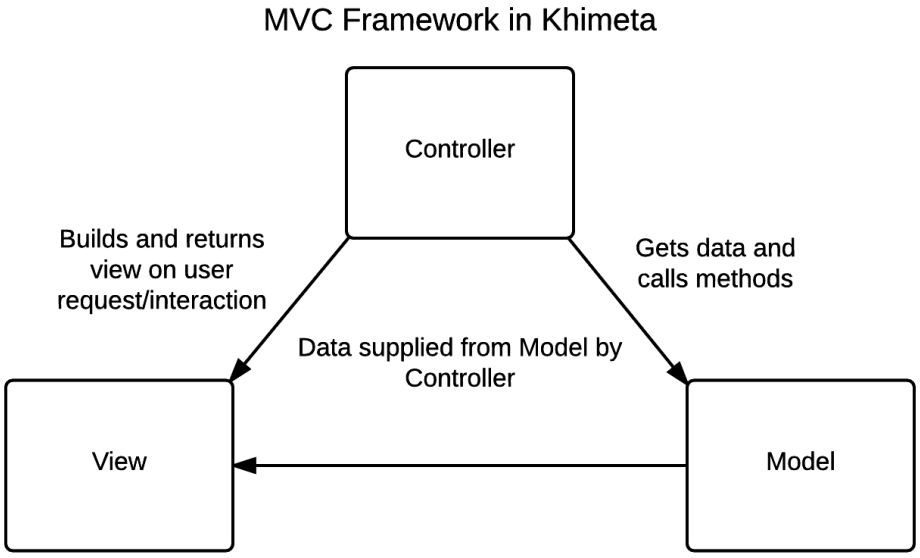


Figure 2.4: The MVC framework the application is designed upon. The data flows between the Model and the View via the Controller, based on the user requests and interaction.

2.1.2.1 Model

The Model was designed to contain the data and methods for processing a users input. This contains the data structures and objects for handling a users data as they traverse from page to page of the view. I designed the Model in a way that certain objects would be Bean objects (classes with getters and setters for their properties) that could be accessed by the View in the way that Thymeleaf required.

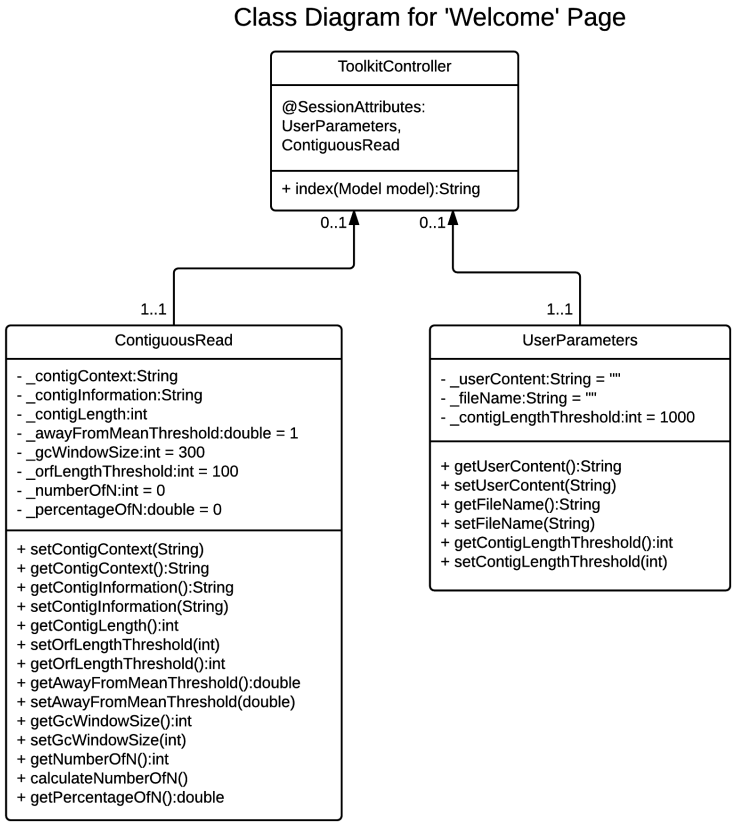


Figure 2.5: Class diagram for the ‘Welcome Page’, and what classes are used upon a Request for the page.

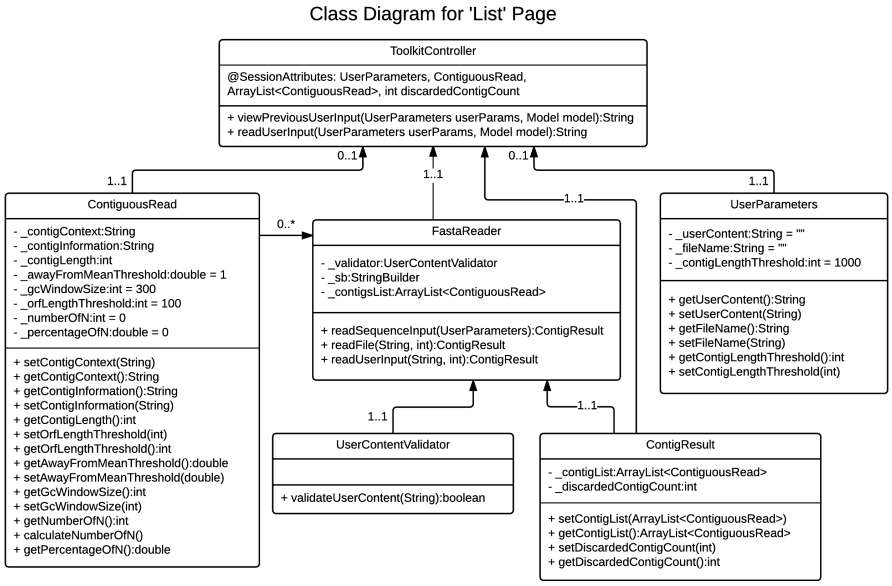


Figure 2.6: Class diagram for the ‘List Page’, and what classes are used upon a Request for the page.

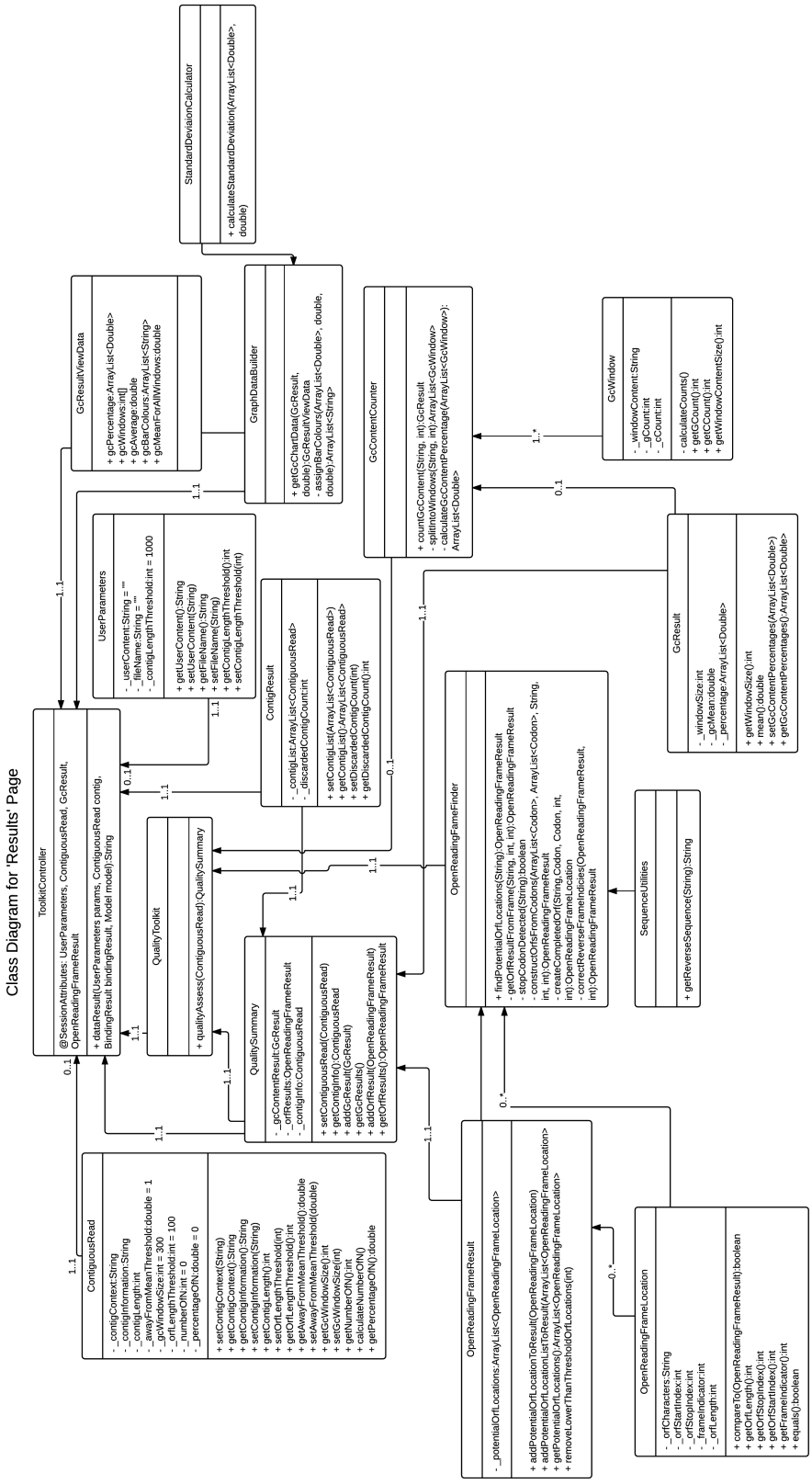


Figure 2.7: Class diagram for the 'Results Page', and what classes are used upon a Request for the page.

The structure evolved over time due to the nature of XP and using an agile methodology. Even so, the result was that there was a logical sense to the structure of classes, methods belong in the class they live in, some utility classes exist for carrying out operations that don't necessarily need to be within a class that needs that utility and this also left the code easier to maintain and separated. By using utility classes and methods, they can be called by other classes in the future, and the class that uses them doesn't need to know about the implementation internals of the methods they use, only the results.

2.1.2.2 View

The View of the application is where the GUI is presented to the user to be seen in their browser. Designing the application to use the browser was chosen to allow anyone to be able to use the application as long as they are using a modern browser. It is using HTML, HTML5 Canvas and JavaScript, dynamically built using Thymeleaf templates. If any of the template does not process or there is an issue with the users data they are presented with an error page and told to return to the menu and try their task again.

The data for the View comes from the Model, passed by the Controller. In line with keeping responsibilities separated, the Model does not care about how the View displays the data, and the View does not care about the content of the Model data it is passed as long as the data types it expects are present in the Model from the Controller. For example, the ORF Location data is sent to the View as an OpenReadingFrameResult Object, that has a list of OpenReadingFrameLocations and other data about the process. It is then the responsibility of the View to use this data in displaying it in HTML5 Canvas with JavaScript. No additional processing or modifications are carried out on the data at this point, the View just picks up the data and places it on the Canvas element where it expects it should go, based on the content of the data, e.g. an OpenReadingFrameLocation with the 'frameIndicator' set as 0 would indicate to the View that it should be placed on Frame 1.

This type of responsibility separation is consistent throughout the application, with one except. There is a GcResultDataView, that is the GcResult data processed into an easy to handle class. This is to make the process for the View far faster than having to get the View to carry out some processing to determine the GcWindows that are out of the mean threshold and additional results to display on the bar chart for the GcResult. While it would be possible to alter the application in order to leave this up to the View, I felt that it would be better to instead serve the View with the window data it needed to put on the chart and the colour codes for each of those window data bars.

2.1.2.3 Controller

The Controller serves as the master for how data flows through the application, making calls to methods from the Model to process the users data, receiving requests from the user via the View and HTML Requests and handling what data is for use in the View. The user may send HTML Requests to the different pages of the application and will receive a response based on the '@RequestMappings' of the Controller.

For example, a GET on the 'list' page will return a page expecting the user to have already submitted some data that has been turned into a list of contigs. A POST on the 'list' page however will be expecting that this is a new set of data and process the data submitted into a list of contigs and put them into the users session attributes.

Through every page, a user carries a Session with them and particular session attributes are expected at certain points in @SessionAttributes. For example, a user must have UserParameters and a ContiguousRead in their session parameters in order to be able to view the List and Results page. If they have not visited the Welcome page and submitted data, however, it is not possible for them to have these attributes, and so they are locked out of those pages. This is good behavior as the user should not be able to try and view results for data they have not submitted.

2.1.3 Naming Conventions

I followed a naming convention of attempting to name classes, methods, fields and variables in such a way that a reader could understand what the function of that particular thing was just from the name alone. Private field names all begin with an underscore, e.g. '_privateField' and everything is written in camel case, starting with a lowercase character, e.g. 'thisWouldBeAMethodName()'.

2.1.4 User Input

Based on the background reading, the decided file format to accept was FASTA format. The design allows for user uploading and user pasting of files in the Model methods. However, in the end I decided to keep the design just to handling pasting of data. This was based on a time constraint and issues with file upload limits, security and the need to test. While the functionality has been left in the code, it has been left as a 'if I were to continue' functionality to be expanded upon if I had more time to work on file uploading as a priority over different techniques for the quality assessment.

User data is also not kept by the application, it is stored in a user session that expires once they leave the page. This is handled by Java Spring, and set as one of the @SessionAttributes. If the user does not have their session variables, they are presented with the Error page, so they cannot try and access areas of the application/web service where they currently do not have access or the data to do so. This means that the data they have is their own, and not retained by the application or possible for other users to access.

2.1.5 Directory Structure

The directory of the applications structure was designed such that it would reflect the MVC aspect of the application, and make finding resources and particular Classes easier for a reader. The design makes the application easier to maintain in the future and is another way of helping to enforce separation of responsibilities between classes and resource types. In the figure below you can see the way that the directory structure is designed to back this up.

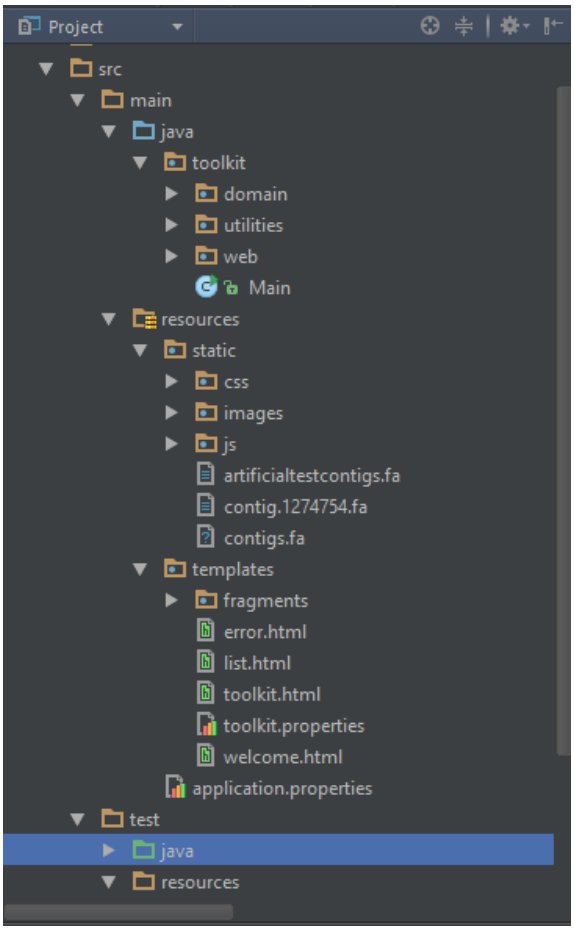


Figure 2.8: The layout of the directory structure for the application.

Within the 'src/main/java' folder, there are the sub folders 'domain', 'utilities' and 'web'. Separating the classes out this way helps with the previously mentioned separation of responsibilities. Domain contains the classes for the domain objects themselves and the data and processing (the Model, so GcResult, OpenReadingFrameLocation, etc), Utilities contains utility classes for use by any of the other classes and to allow their functionality to be independent of any domain class and Web contains the Controller classes (ToolkitController, ErrorController).

2.1.6 QualitySummary and Results

In the application, the quality assessment results are returned in a QualitySummary object. This QualitySummary contains references to GcResult and OpenReadingFrameResult. The is the option to have these two implement an interface that might be called QualityResults, and have just an array of QualityResults. This would allow us to add any type of result without needing to know what is in there. However, I chose not to do this as considering the way in which we serve data to the View it seemed okay to be able to have the View called results from the model directly, and through XP developing this any further would go against YAGNI.

I am aware that in the future if we allowed a user to select what type of processing they wish to use and as more techniques for quality assessment are added, this technique would be very useful

to implement. It would be possible to do this and in the View have it check for the existence of the expected results, or check if they were empty/null, and then only dynamically include fragments of results that are present.

2.2 User Interface

Below is the finalized designs of the user interface, shown through screen shots, along with an explanation for the design choices in how the pages are laid out for the user to navigate through. Each page has a header, containing the title of the application and the logo I made.

The logo is based on the mythical creature 'chimera', a creature made from multiple parts of other creatures (often a lion, snake and a ram). The inspiration for the name and design comes from the application hoping to highlight potential Chimeric regions of contigs to a user, hence 'Chimera - Metagenome - Chimeta'. The 'K' in Chimera is simply because 'Chimeta' sounds too much like it would be pronounced 'chai-meta' not 'kai-meta' as in 'kai-meera' for Chimera. It isn't an important aspect of the design.

2.2.1 Welcome

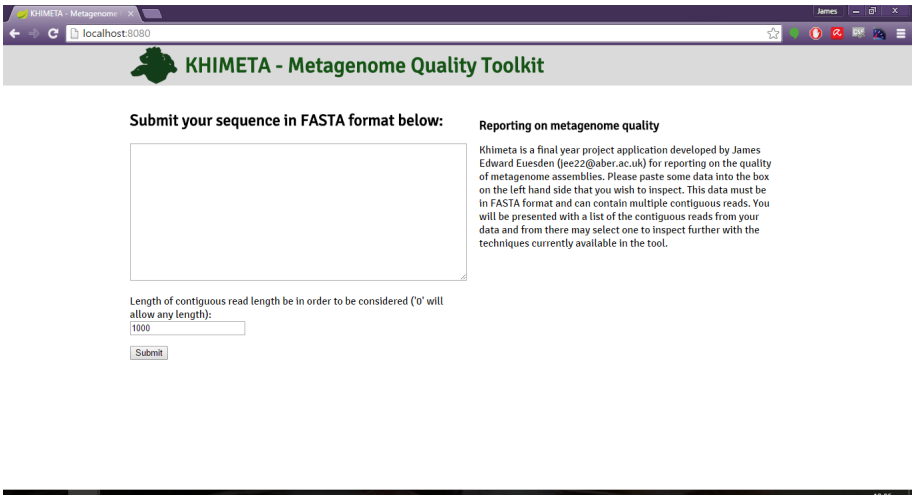


Figure 2.9: The 'welcome' page that greets the user upon requesting the home page of the web service.

The welcome page is structured so that there is an explanation for the user of what the tool is, and an area for them to paste their FASTA data in. There wasn't much else needed for this page, but were the project to be continued this is where there would also be the option to upload user files instead of pasting in data. The user also specifies the minimum length a contiguous read must be in order to be considered here.

The original design for this page also included the parameters for modifying the quality assessment, as this tied in with when the application dealt with only one contiguous read at a time. As the application evolved, the design changed and was refactored to shift the parameters for quality

inspection to the next page where they could be tied to contiguous reads and not to the assembly data submission as a whole.

2.2.2 List

The below images show the 'List' page, where a user can see all of the contiguous reads within their uploaded data file that are above the minimum length threshold they specified. The page also displays the number of discarded contigs due to the length, as I felt this would be useful information to give to a user.

For each contig, I also believed it would be useful for the user to see the contigs header (name), the length (how many characters) and the number of unknown 'N' characters, along with the percentage of those that made up the contig. This could be used for a user to consider whether a contig has too many of these characters to the point that they feel it is a bad quality contiguous read.

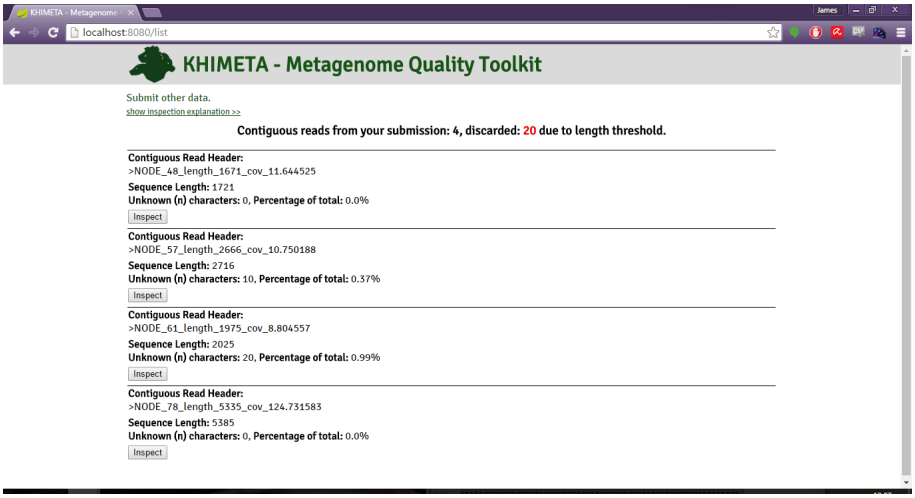


Figure 2.10: The 'list' page that is displayed after a user submits their (valid) assembly data in FASTA format.

If a user wishes to further inspect a contiguous read, I designed it such that when they click to 'Inspect' on a contig, a box appears underneath the contig information for them to modify the parameters for the quality assessment process. I had previously considered a design where these parameters were on the right side of the list, and would stay static there as the user scrolled down the page. However, if a contigs name was very long it would run 'underneath' the box and be hidden. I also found it just did not look very appealing, and so I opted for the design you can see below instead. When the user clicks to run the process the 'Inspect' button changes to 'Loading...', to inform the user that the process is working.

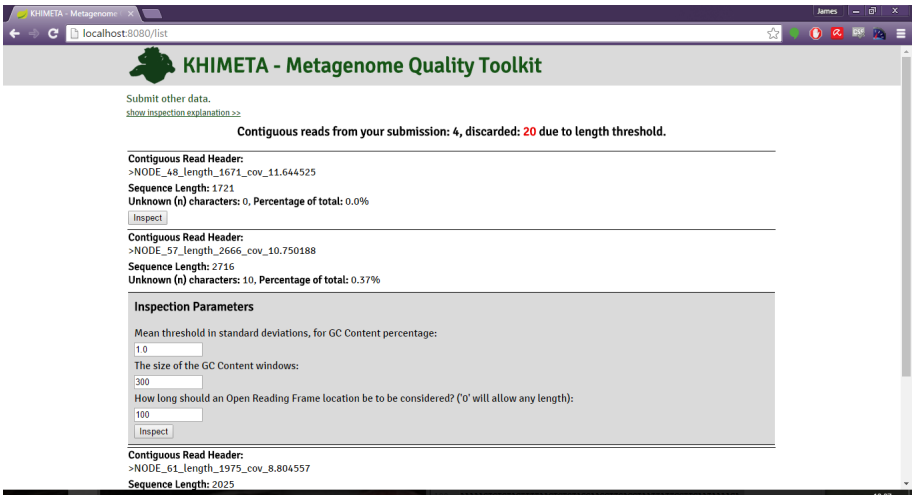


Figure 2.11: When a user clicks to inspect a contiguous read, this menu appears giving them options to alter the parameters of the quality assessment inspection.

In order for a user to understand what the inspection process is, I included a box that details the process, which appears when they click the words for 'Show inspection explanation'. This box can be hidden by clicking the 'Hide inspection explanation'. Currently, the link to display the explanation is at the top of the page, and a user must be at the top of the page to click and view it. It could instead be possible to have the list of contigs scroll in their own box, such that the inspection explanation is always available to be seen and viewed when clicked. However, it did not fall into scope of my Sprints while working on the functionality.

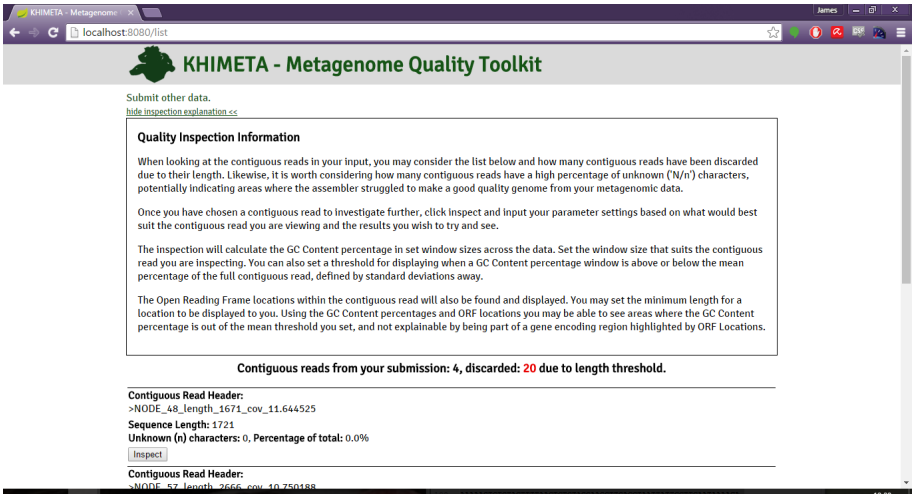


Figure 2.12: If the user clicks to see an explanation of the process, they can click the text that says to 'Show explanation', and this text appears.

2.2.2.1 Parameter choices

The choice of parameters that the user can modify were selected based upon what could be altered about the inspection processes. For example, allowing a user to change the minimum length of

an Open Reading Frame location gives them control over what size they think is important, based upon what their assembly data is. It could be that they care about any sized potential protein coding region, or that they only wish to see ones of size 1000 or above, as they know that anything less is unlikely to be an actual protein coding region. Giving them this parameter choice allows them to decide this for themselves, as the application won't know what is the best size for the user.

Similarly, the GC content window size should be set by the user, again based on what they think is a reasonable size. If a user were to think that they might want windows of size 100, that probably wouldn't return them great results on a large contig, as the GC content could vary so much between windows where the data is just too small to be useful. Likewise if the window size is too big, this data may also be useless, as they cannot properly compare their GC window sizes to the expected size of what they want to see from protein coding regions.

A reasonable window and minimum ORF Location size can only be selected by the user, and so the choice was made to allow the user to select their own choices, even allowing them to select values to high or too low, it is their decision to make based upon the data they have provided. In future versions of the application, it would be interested to infer a decent window size for the user and suggest it to them, but for the purpose of this project it was out of scope for the time allotted with the other functionality to be implemented.

The final parameter is how many standard deviations away from the mean of all GC content windows the user would like to consider is enough to warrant highlighting a window as being a potential issue. I selected standard deviation as it is a standard way of determining a value away from the mean that could be useful to inspect. Whether the user cares how much or how little this value actually is is irrelevant. The user may select what number of these they feel is useful to consider just for highlighting bars in the results chart.

2.2.3 Results

For the display of the inspection results, the user is shown a set of labeled tabs, that correspond to the techniques used. It is designed this way so that more tabs can be added in the future as more techniques are set in the inspection process, and to keep the page from being too crowded with results that would require a large amount of scrolling. The code for making the tabs themselves work is credited to Catalin Rosu [1] and their original code can be seen in the appendices.

2.2.3.1 GC Content

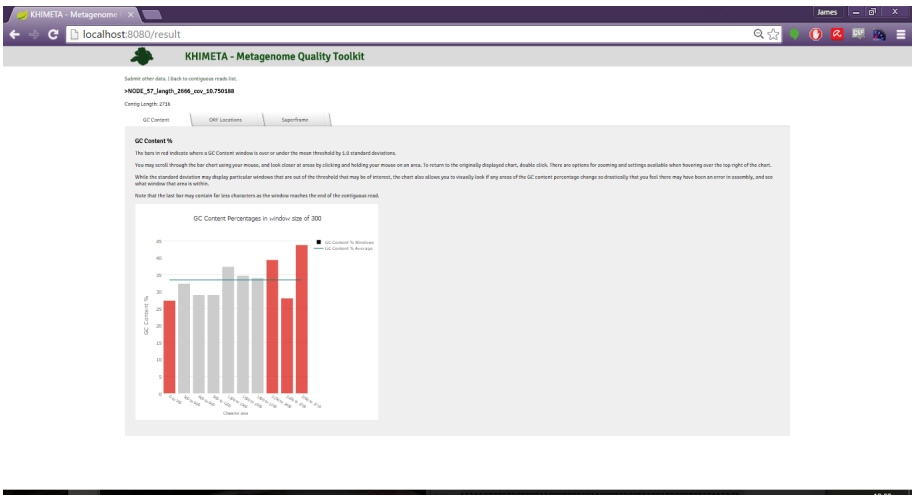


Figure 2.13: The GC result tab shown to the user on inspection of a contig. The browser has been zoomed out in order to display the page in full.

The GC Chart result is displayed to the user in order to give them a visual reference of how the GC content in their contig is. Through this they may detect single windows that may have chimeric properties through a difference in GC content above or below the mean, which can be highlighted in red and are set by being a number of standard deviations away from the mean. It also allows them to look and see if there is a naturally large difference at points between the GC content windows that are not necessarily picked up by the threshold.

The chart is built using Plotly.js [6], as it allowed me to give the user more control over how they view the GC content results in a chart than if I spent the time implementing my own solution to give them the same amount of fine-grained control. A number of the options offered to a user by the Plotly.js chart are probably far more than they need to be and may be confusing to a user. If the project were to be continued it may be worth developing a chart system for displaying the GC content without using Plotly.js, which could also help with developing a way of showing when there are drastic changes in the GC content not caught by the threshold.

Above the chart is a small explanation of what the chart is displaying, and what the user might look for in order to understand the chart and whether there could be potential issues in their contig displayed by these results.

2.2.3.2 Open Reading Frames

The tab for the Open Reading Frame displays a graphic made using JavaScript and HTML5 canvas, that gives the user a visual representation of 6 frames of the contig and any protein coding regions within those frames. The design was inspired by NCBI‘s orffinder [19].

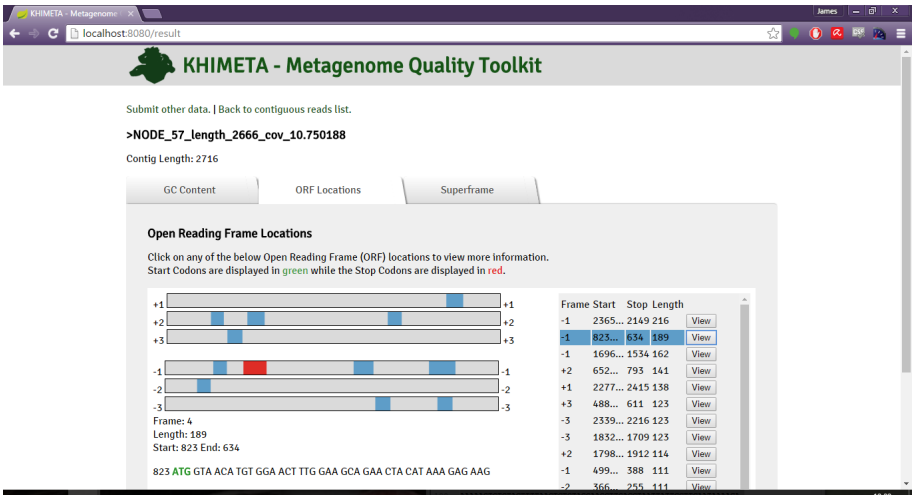


Figure 2.14: The Open Reading Frame results view tab.

I designed it in a way that allows the user to view the full list of ORF Locations (protein coding region) on one side, including their character length, start and stop point and which frame they are in, organized by largest to smallest. This is reflected in the frame chart where a user can see where the frames physically lie within the contig. If a user clicks an ORF Location, either on the chart or in the list, it highlights that ORF in the list in blue, and red in the chart. This also displays more information about the ORF Location underneath the chart.

The choice to display the ORF Location information was out of interest to the user if they wish to use the tool to look at the protein coding regions. The information displayed about an ORF Location includes the length, start and stop point and the characters within the ORF Location. While these might not be useful to the user for determining quality, the information is used in the comparison for GC content to find if the GC content is explained by an ORF Location, and so I believed the information might be useful to display to a user either way.

Each frame has its own canvas element, and they are aligned using a HTML table. I tried a design where each of the 6 frames was made up in a single canvas, and while this worked, it restricted the amount of formatting that could be done around each frame and had much more code for detecting whether a user has clicked within a frame, delaying the response a user got when clicking.

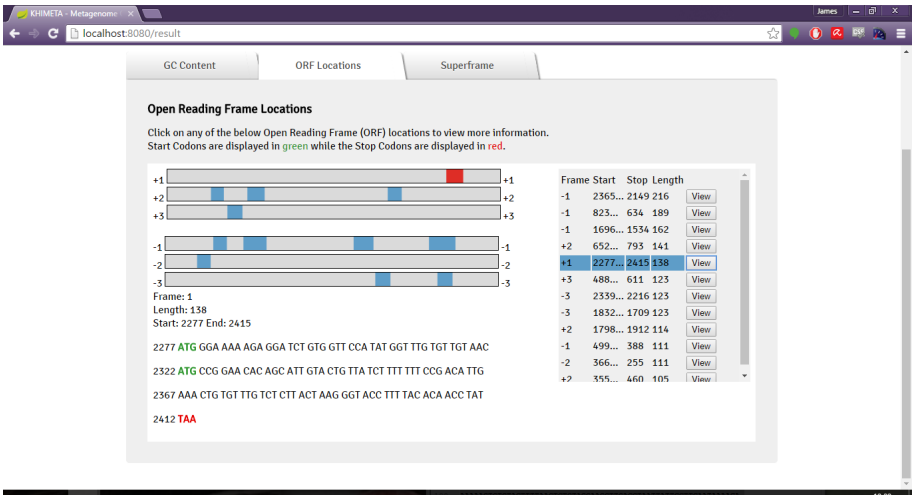


Figure 2.15: The Open Reading Frame results view tab, displaying that when a user clicks on a different ORF Location, the highlight on the frame chart and list changes to reflect this.

2.2.3.3 Superframe

Superframe is where the GC content and ORF Location charts are compared in order to help a user see if there are any areas of GC content difference that might be explained by the ORF Location. It is simply an overlaying of the 6 frames into one chart (all in the same sequence direction) and overlaying the GC content chart results for each window.

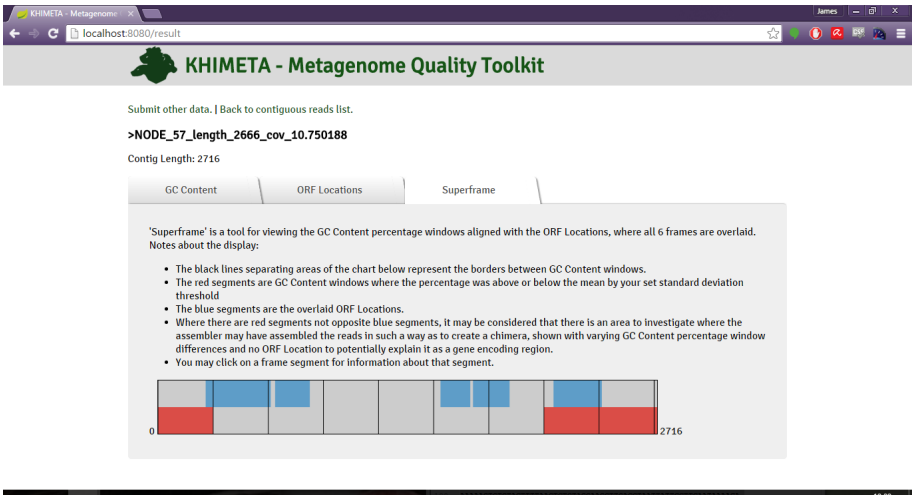


Figure 2.16: The ‘Superframe’, a comparison between the ORF Locations within the contig and the GC content windows that are above the threshold..

If the user clicks on a particular window, they can see information about what window that is (where it starts and ends) and the GC content percentage. I felt this tool could be useful for a user to be able to determine themselves if they felt there were enough individual windows of GC content that were out of the mean threshold and not explainable by ORF Locations and so felt the quality of the contig was or was not good. I attempt to explain this with the description above the

chart.

It is worth noting, the GC content areas highlighted for the Superframe are only those that are above the threshold of the mean, not below it. This is because it is those areas that are above could be explained by an ORF Location, as the GC content tends to be higher in protein coding regions, and so there is no use in displaying the GC content windows out of threshold by being below, which cannot be explained by the ORF Locations.

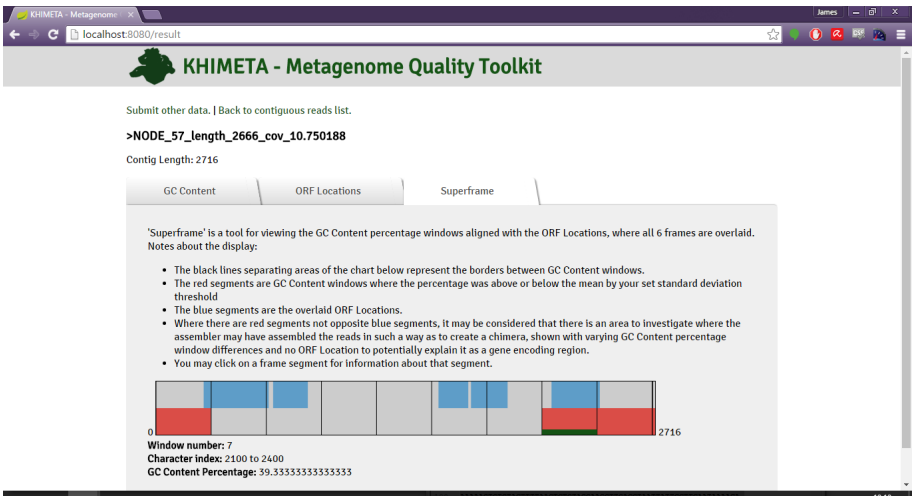


Figure 2.17: The result of when a user clicks to view a window data, they can see the particular percentage of the GC content of that window, and where that window starts and finishes, if they wished to inspect the contig themselves using those numbers.

2.2.4 Error

If a user encounters an error on the website, such as attempting to access pages without first submitting data) the page in the figure below will be shown.

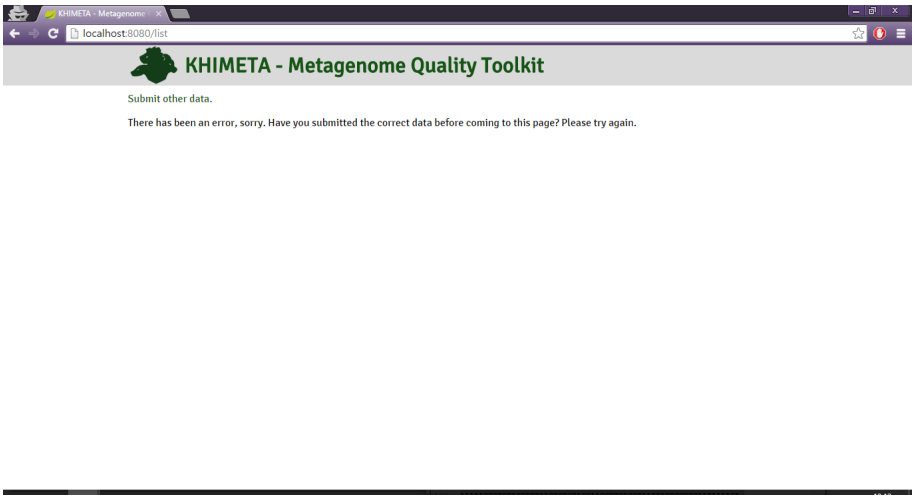


Figure 2.18: If an error occurs as the user is browsing the web service, i.e. they try to access a page when they do not have the required data submitted, they are presented with this page informing them of an issue.

While not very descriptive, it serves to return the user back to data submission, which is the best place to start again if something bad went wrong or they somehow tried to access a page when they hadn't previously submitted the required data. If there were a future revision of this page, it could be more descriptive of the exact reason why the error occurred, but was out of scope for the current project to make a fully detailed error reasoning. There is an ErrorController class in place however that can deal with this functionality in the future if the application were to be continued. Currently any errors are run through the Error controller but it only returns this page to the user.

2.3 Support Tools

For writing the code, I used JetBrains IntelliJ Community Edition IDE. This included writing all the code be it Java, HTML, CSS, Javascript, ThymeLeaf and any additional properties for setting up Spring Boot. This was very useful for text highlighting, debugging the code, and tied into my version control, using Git and hosted on GitHub.

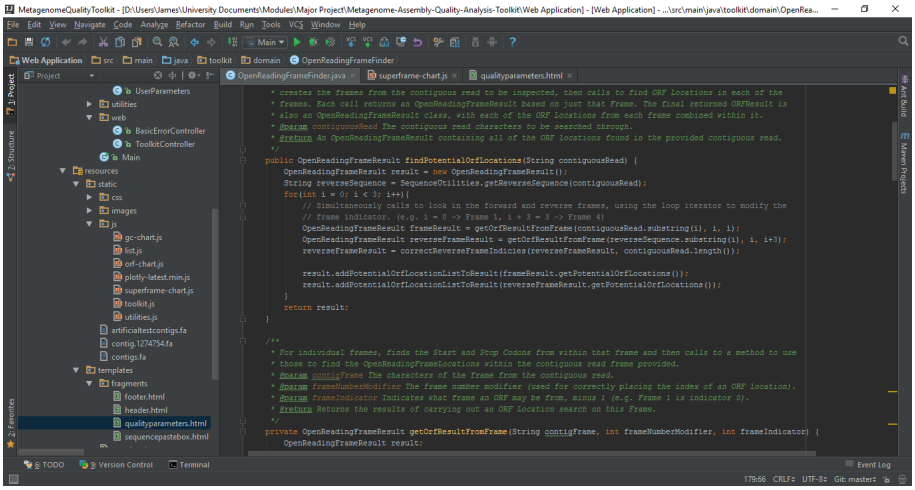


Figure 2.19: The Jet Brains IntelliJ Integrated Developement Environment (IDE) I used for developing my application.

I also used `http://www.tomato.es/` for counting my pomodoros. The web site provides a timer and a count for how many pomodoros have been completed during the time the browser has been opened.

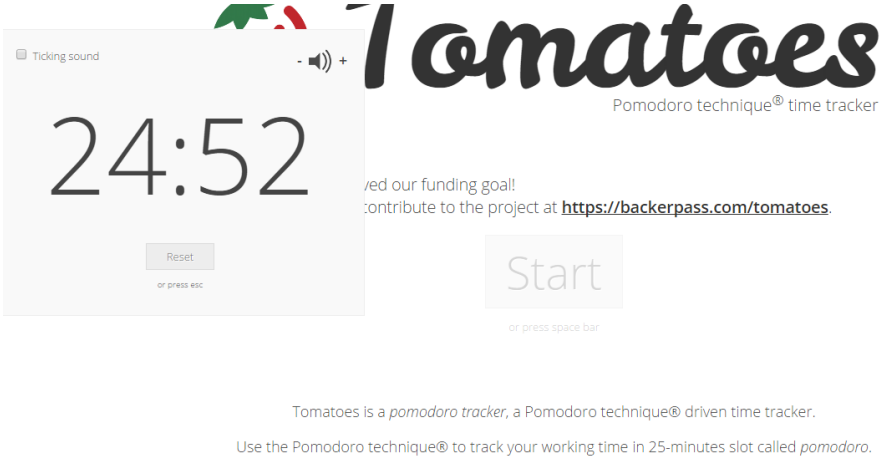


Figure 2.20: A countdown of the pomodoro tomato timer on `http://www.tomato.es/`, used for breaking up development time and keeping track of work done over time.

2.3.1 Version Control and Continuous Integration

I used version control to keep a repository of my code in case of losing data in event of hardware crash or software corruption. For this I selected to use GitHub as I already had a repository on there and Git is supported in the IntelliJ IDE I was using. This can be found at `https://github.com/coderghast`.

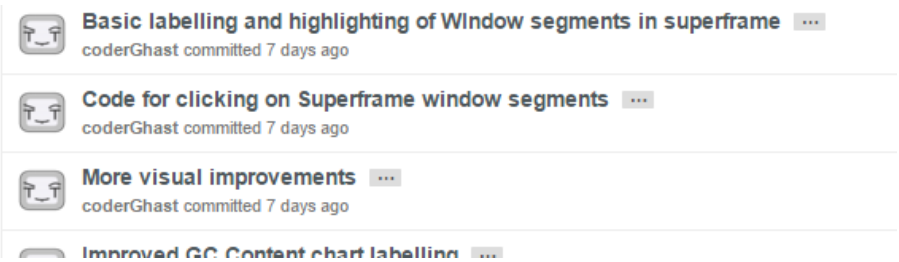


Figure 2.21: A few of the commits made to my major project repository.

In addition, every time I checked in, I used continuous integration with CodeShip [3], allowing me to receive e-mails any time I made a commit to my repository that didn not pass the tests written. This was very helpful in discovering issues with failing builds when checking in a change and forgetting to update tests or breaking previously written code as the design changed.

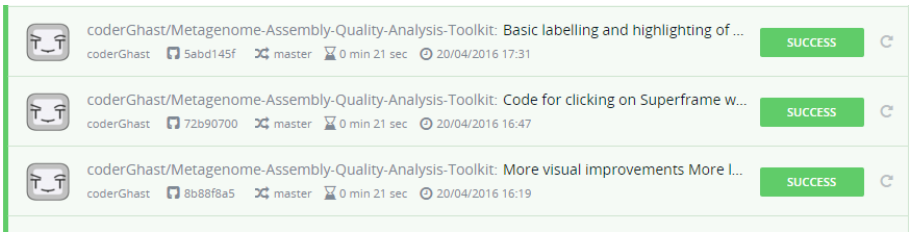


Figure 2.22: Examples of the continuous integration on CodeShip running my tests every time I committed to my repository.

Chapter 3

Implementation

This chapter serves to discuss the implementation specifics of the design of my application, including the development environment and any issues I encountered in implementing the features set out in my project objectives.

3.1 Development Environment

The application was developed primarily on a Fujitsu Lifebook A series with an Intel i5-3230M CPU @ 2.60GHz processor, 4GB of RAM and running Windows 10. The code was developed using Jet Brains IntelliJ IDE and Google Chrome developer console.

3.2 Features

3.2.1 Reading User Input

3.2.1.1 Implementation

The user input was one of the first parts of the application to be implemented, and consisted of reading in a .fa FASTA file and just outputting the content of the file to the console and return it from the method to be tested against with automated unit tests. It was developed further to extract out the header, checking that it began with the '>' character and then every other line was included as part of that contig.

Originally it dealt with only one contig at a time, but was then expanded to be able to read in a full list of contigs and separate them based on where the header line starts. The code was eventually modified so that the user could paste data, that would be broken into components and processed in the same way. This was for the pasting of the user data rather than file upload, as I decided I would rather have pasted data than uploads for the current state of the application.

```
for(int i= 0; i < contigCompontents.length; i ++){
    String line = contigCompontents[i];
    // If the next contig compontent from the String array is the header of a new contig
    if (line.startsWith(">")) {
        // And if there is already contig information (header) stored
        if (currentContig.getContigInformation() != null) {
            // Set the content of the contig to the contig context, as we're finishing one contig
            // and moving onto a new one
            currentContig.setContigContext(_sb.toString());
            // As long as the length of this contig is above the threshold, add it to the contigs
            // we want to return to the user.
            if (_sb.length() > minimumLengthThreshold || minimumLengthThreshold == 0) {
                _contigsList.add(currentContig);
            } else {
                discardedContigs++;
            }
            // Clear out the string builder to build up new contig context.
            _sb.setLength(0);
        }
        // Set the contig information back to new.
        currentContig = new ContiguousRead();
        // Then set the header to be the current line that starts with '>'
        currentContig.setContigInformation(line);
    } else {
        // Otherwise just add this line to the string builder for the contig context, as it is part
        // of the content of the contig sequence.
        _sb.append(line.trim());
```

Figure 3.1: Part of the code for reading in a users data when they have pasted it into the text area of the web service. Deals with creating new ContiguousRead objects and adding them to a ContigResult every time it finds a new header for a contig (or reaches the end of the input).

3.2.1.2 Issues

It took some time to deal with the formatting of the file and knowing when one contiguous read starts and one ends. There was some issue with the escape/return characters in the dealing of pasted content by a user, but this was solved by breaking the pasted content into an array of its components.

While not an issue as such, the original version of the application would carry out the quality assessment of each contig as and when they were read in, to avoid holding too much data in memory. As the design evolved this was no longer a possibility, as it didn't allow the user to see their contigs before they were processed, or do any additional inspection on the contigs without holding the entire user data submitted in memory and going through it a second time upon an inspection request to find the contig they wanted. This option is still a possibility, but is an optimization aspect that wasn't in scope for the project. Additionally, the separation of responsibilities is better in this version of the code after being refactored from the time when the read class both read and assessed the contigs.

It did take longer to implement this than I had anticipated due to me changing from file reading to user pasting, and trying to find contig start and ends to read in the data versus read and assess at the same time.

3.2.2 Counting GC Content & Percentage

3.2.2.1 Implementation

Counting the GC content of the application involved breaking up the contig characters into sizes based on the set window length by the user, then working out the percentage of G and C characters in each of those windows. Each part of the contig that is split is put into a 'GcWindow' object, that has methods for counting the number of G and C characters within it.

```
private static ArrayList<GcWindow> splitIntoWindows(String contiguousRead, int windowSize){
    ArrayList<GcWindow> windowedAssembly = new ArrayList<>();
    for(int i=0; i < (contiguousRead.length()); i += windowSize){
            StringBuilder stringBuilder = new StringBuilder();
            if(i + windowSize < contiguousRead.length()){
                stringBuilder.append(contiguousRead.substring(i, i + windowSize));
            } else {
                stringBuilder.append(contiguousRead.substring(i));
            }
            windowedAssembly.add(new GcWindow(stringBuilder.toString()));
    }
    return windowedAssembly;
}
```

Figure 3.2: The code for splitting a contiguous read into windows available for calculating the GC content window percentages.

Once split, the GcWindows are passed to have their percentages calculated and added to an ArrayList of Doubles This is used for calculation with the mean, standard deviation and for returning results to the user for use in the View.

```
private static ArrayList<Double> calculateGcContentPercentages(ArrayList<GcWindow> windows){
    ArrayList<Double> percentages = new ArrayList<>();

    for (GcWindow gcWindow: windows) {
        int gcContent = gcWindow.getCCount() + gcWindow.getGCount();
        int contentWindowSize = gcWindow.getWindowContentSize();
        if(gcContent > 0) {
            percentages.add(((new Double(gcContent) / new Double(contentWindowSize)) * 100));
        } else {
            percentages.add(new Double(0));
        }
    }

    return percentages;
}
```

Figure 3.3: The code for calculating the GC content percentages for each window passed into the method.

3.2.2.2 Issues

In principle this is, and was, an easy task, yet took me longer to complete because my domain knowledge was still somewhat lacking and I got caught up on the little details that kept me from progressing, even though I didn't need to know of them or use them in the process of working out

the GC content windows. Overall there were no issues with the implementation once I got my head around why it was worth doing this and how it should be presented as a quality measure to the user.

3.2.3 Displaying GC Content percentage

3.2.3.1 Implementation

I began to attempt to implement GC content viewing with Plotly.js right from the beginning, and so I had a prototype up and running fairly quickly where I manually took the results from the console output of the Java application and pasted them into the final containing my Plotly.js prototype to be used as data.

Next I worked this in with Thymeleaf to get the results directly from my Java application, then turned them into JavaScript and used the data from that for the chart. The x axis labeling for the chart is manually created in order to represent where the GC window for each bar starts and finishes. The earlier design had window numbers along the x-axis, populated by Plotly.js. These were of no use to a user though, as if they wanted to find out what GC window they were looking at they would have to manually work out where the window started and ended from their window size multiplied by the window number of interest.

```
<script th:inline="javascript">
/* Setup data needed */
/*<![CDATA[*/
    var windowdata = [[${gcResult.gcPercentages}]];
    var windownums = [[${gcResult.gcWindows}]];
    var gccontentmean = [[${gcResult.gcAverage}]];
    var gcColours = [[${gcResult.gcBarColours}]];
    var gcMeanForAllWindows = [[${gcResult.gcMeanForAllWindows}]
    var windowSize = [[${contiguousread.gcWindowSize}]];
    var contigLength= [[${contiguousread.contigLength}]];
    var contigName = [[${contiguousread.contigInformation}]];
    var contigContent = [[${contiguousread.contigContext}]];
    var orfData = [[${orfResult}]];
    function init(){
        drawGcChart();
        setupOrfChart();
        setupSuperframeChart();
    }
    init();
/*]]>*/

</script>
```

Figure 3.4: Extracting the data from the Model, provided by the Controller to the View, using Thymeleaf's inline tag to be able to use CDATA to convert the Thymeleaf extracts into JavaScript.

The red bars of the GC chart displaying where the percentage of a window is over or under the threshold of the mean of all GC window percentages was implemented as a useful aid to the user, and the colours are set in the Model rather than made in the View. The reasoning for this was to implement a system that would provided the View with what it needed and not needing the View to calculate anything, only consume the values (in this case the GC window data and RGB data) to display results.

```
public class GcResultViewData {
    public ArrayList<Double> gcPercentages;
    public int[] gcWindows;
    public double gcAverage;
    public ArrayList<String> gcBarColours;
    public double[] gcMeanForAllWindows;
}
```

Figure 3.5: 'GcContentViewData', the object provided to the View in order to display the GC content data to the user.

The final thing to add was the Mean line. This is just displaying to the user where the mean of all the GC windows lie, and helps with the visualization of if any windows of their contiguous read look like they might have issues.

3.2.3.2 Issues

The main issue I had with displaying the GC content percentages with Plotly.js was getting data to actually display as I was unfamiliar with Plotly.js and Thymeleaf once I started using Spring Boot. Thymeleaf makes getting data from the Objects in the Model very easy and simple, but since I was trying to integrate two technologies together that I had never used before, it took some time to find the right way of accessing the data from my Object through Thymeleaf, into JavaScript and then the correct format for use by Plotly.js.

The next issue was trying to get the line for the Mean to show. The way I got this to work was using an additional data input for the same chart as displaying the GC content window bars and having the same number of data points as there are GC windows, where each data point is the mean number, in order to get the chart to display a line across the entire chart to match the GC windows.

3.2.4 Finding Open Reading Frames

3.2.4.1 Implementation

The implementation of Open Reading Frames happened in a few steps, with the design and code evolving as I understood more what they were and how best to present the results. At first, I was only finding a single, longest ORF Location, as I did not fully comprehend what an ORF was. This was just looking at a contig and finding the first start Codon of ATG and the last stop Codon of TAG/TAA/TGA. Once I understood them though, the process became about how to find whatever ORF Locations I could. Using Test Driven Development was extremely useful for this step, as I wrote tests for finding specific ORF Locations in small sequences I created myself and ran the code against them.

After a number of attempts at trying to find the ORF Locations within a contig frame, I settled on a method of going through the entire frame, breaking it up into characters of 3 and only keeping the start and stop Codons in lists of each. Each Codon was stored with data about where it starts and ends, in order to be able to later reconstruct the contig between a start and stop Codon based

on the character positions.

```
private OpenReadingFrameResult getOrfResultFromFrame(String contigFrame, int frameNumberModifier, int frameIndicator) {
    OpenReadingFrameResult result;

    ArrayList<Codon> startCodons = new ArrayList<>();
    ArrayList<Codon> stopCodons = new ArrayList<>();

    for (int i = 0; i < contigFrame.length(); i += 3) {
        if ((i + 3) <= contigFrame.length()) {
            if (contigFrame.substring(i, i + 3).equalsIgnoreCase("ATG")) {
                startCodons.add(new Codon("ATG", i + frameNumberModifier));
            }
        }
    }

    for (int i = 0; i < contigFrame.length(); i += 3) {
        if ((i + 3) <= contigFrame.length()) {
            String nextCodonInFrame = contigFrame.substring(i, i + 3);
            if (stopCodonDetected(nextCodonInFrame)) {
                stopCodons.add(new Codon(nextCodonInFrame, i + frameNumberModifier));
            }
        }
    }

    result = constructOrfsFromCodons(startCodons, stopCodons, contigFrame, frameNumberModifier, frameIndicator);
    return result;
}
```

Figure 3.6: Finding all of the start and stop Codons from within the passed frame from the contig.

Each frame is separated by creating a substring of the original contiguous read, removing 0, 1 and then 2 characters for the first three frames, and the same for the reverse frames but with the contiguous read sequence reversed with the opposing base pair of each character used.

```
public OpenReadingFrameResult findPotentialOrfLocations(String contiguousRead) {
    OpenReadingFrameResult result = new OpenReadingFrameResult();
    String reverseSequence = SequenceUtilities.getReverseSequence(contiguousRead);
    for(int i = 0; i < 3; i++){
        // Simultaneously calls to look in the forward and reverse frames, using the loop iterator to modify the
        // frame indicator. (e.g. i = 0 -> Frame 1, i + 3 = 3 -> Frame 4)
        OpenReadingFrameResult frameResult = getOrfResultFromFrame(contiguousRead.substring(i), i, i);
        OpenReadingFrameResult reverseFrameResult = getOrfResultFromFrame(reverseSequence.substring(i), i, i+3);
        reverseFrameResult = correctReverseFrameIndicies(reverseFrameResult, contiguousRead.length());

        result.addPotentialOrfLocationListToResult(frameResult.getPotentialOrfLocations());
        result.addPotentialOrfLocationListToResult(reverseFrameResult.getPotentialOrfLocations());
    }
    return result;
}
```

Figure 3.7: Extracting each frame and calling to run the ORF Finding process.

For each ORF Location found in a reverse frame, the start and stop indexes are swapped, representing that what is the ‘Start’ of an ORF Location is actually closer to the end of the contiguous read. This is so that it can be displayed appropriately in the View, and match up with the proper direction with the forward frames.

```
public static String getReverseSequence(String sequence){
    StringBuilder reverseSequence = new StringBuilder();
    int i = sequence.length() - 1;
    while(i >= 0){
        switch(sequence.charAt(i)){
            case('t') : reverseSequence.append('a');
                break;
            case('a') : reverseSequence.append('t');
                break;
            case('g') : reverseSequence.append('c');
                break;
            case('c') : reverseSequence.append('g');
                break;
            case('T') : reverseSequence.append('A');
                break;
            case('A') : reverseSequence.append('T');
                break;
            case('G') : reverseSequence.append('C');
                break;
            case('C') : reverseSequence.append('G');
                break;
            default : reverseSequence.append('N');
                break;
        }
        i--;
    }

    return reverseSequence.toString();
}
```

Figure 3.8: Getting the base pair characters of a reverse frame is as simple as a switch statement and building the reversed contig from back to front.

Through this process, it made it possible to find which Codons came before and after which within the contiguous read in order to find which would construct the longest ORF Locations, which Start Codons were redundant (as they were between an earlier start Codon and the next Stop Codon), and in my first completed version of the algorithm, where the last Stop Codon before the next Start Codon was.

At this time, I still had a misunderstanding about protein coding regions, in that I believed the end of an ORF Location was at the last stop Codon before the next start Codon, when in fact it should have been the first stop Codon it reaches is the end of the current Codon. This was a simple fix though as it just involved removing a section of my code and fixing my tests.

```
private OpenReadingFrameResult constructOrfsFromCodons(ArrayList<Codon> startCodons, ArrayList<Codon> stopCodons,
                                                       String contigFrame, int frameNumberModifier, int frameIndicator) {
    OpenReadingFrameResult result = new OpenReadingFrameResult();

    //...
    while(startCodons.size() > 0 && stopCodons.size() > 0){
        // Remove every Stop Codon that comes before our current first known Start Codon.
        for (int i = 0; i < stopCodons.size(); i++) {
            Codon stopCodon = stopCodons.get(i);
            if(stopCodon.getContigStartIndex() < startCodons.get(0).getContigStartIndex()){
                stopCodons.remove(stopCodon);
            }
        }

        // Remove every Start Codon that comes after the current Start Codon but before the next Stop Codon
        if(stopCodons.size() > 0){
            for(int i = 0; i < startCodons.size(); i++){
                if(startCodons.size() > 1){
                    Codon nextCodon = startCodons.get(1);
                    if(nextCodon.getContigStartIndex() < stopCodons.get(0).getContigStartIndex()){
                        startCodons.remove(nextCodon);
                    }
                }
            }
        }

        // If at the end of all of this there is still an ORF to be made, construct it, then remove the components.
        if(startCodons.size() > 0 && stopCodons.size() > 0){
            // Since we're sweeping through the lists in passes, make sure the Start and Stop Codons are somewhat aligned
            if(startCodons.get(0).getContigStartIndex() < stopCodons.get(0).getContigStartIndex()){
                //...
                OpenReadingFrameLocation newLocation = createCompletedOrf(contigFrame, startCodons.get(0), stopCodons.get(0), frameNumberModifier,
                        frameIndicator);
                stopCodons.remove(0);
                startCodons.remove(0);
                result.addPotentialOrfLocationToResult(newLocation);
            }
        }
```

Figure 3.9: 'Zipping' together ORF Locations from start and stop Codons within the frame.

Once each frame has been processed to find the ORF Locations within them, the results of each frame are combined together in a single OpenReadingFrameResult. Each ORF Location is aware of what frame it belongs to, and so combining them all into one result doesn't make a difference when determining what ORF Location belongs where.

3.2.4.2 Issues

The two main issues with implementing the Open Reading Frame finding algorithms were my lack of understanding of what they were when I began, leading to the second issue of implementing more than was necessary. It took me longer than I had anticipated to implement the code for finding ORF Locations because I believed I had the additional requirement task of finding the last stop Codon before the next start Codon, where every stop Codon between the initial start Codon and that last stop Codon was to be included in the ORF Location but ignored as an actual start and stop.

Thinking up how to do this algorithmically took some planning and time, and eventually I came to the conclusion of finding all start and stop Codons and then running a while loop, checking that there was at least 1 of each left in each list to make an ORF Location, where the Start came after the Stop (for reverse frames, this logic still applies as the indexes are only reverse after the ORF Locations have been constructed) and then carrying out a number of conditional checks to remove any start and stop Codons that came between the longest ORF that could possibly be made.

Once I had realized this was erroneous and the first stop Codon encountered is the actual end of an ORF Location and not to continue, it was too late to recover any time and I just had to remove the code that caused the issue and rewrite my tests to match the actual expected outcome. Aside this, once I designed the code based around the 'zipping' of start and stop Codons together, the implementation was relatively simple, using unit tests to ensure I was correctly implementing

the algorithm for expected results.

3.2.5 Displaying ORF Locations

3.2.5.1 Implementation

The first part of the ORF Location displaying for the View takes place in the Controller, where it strips away ORF Locations under minimum length threshold set by the user and then sorts all ORF Locations by length using a comparison method. The display of the ORF Locations in their particular frames is done using HTML5 Canvas, one for each frame. As each ORF Location holds an integer value representing what frame it belongs to, when painting the canvases it is a simple matter finding the correct context for the particular frame from an array of canvas contexts

```
var currentContext = contextList[orfData[i].frameIndicator];
```

Reverse and forward frame ORF Locations are painted in the same way, except the start and stop indexes are swapped for reverse frames. Likewise, for the highlighted ORF Location to be painted, the only difference is that the current highlighted frame is stored in a global variable (updated any time an ORF Location is clicked in the list or on a frame) and when re-painting the frame canvases and iterating through the list of ORF Locations, when the loop reaches that ORF Location the fill colour is set to be different than every other ORF Location.

Finding a click within a canvas frame is as simple as checking if the click is horizontally within a frame location by calculating where the start and stop points of all the ORF Locations (with that frame in particular) are and if the click falls within one of the the ORF Locations.

```
// Look if the passed x coordinate and frame match with any ORF Locations
function checkIfWithinORFLocation(x, frameNumber){
    for(var i = 0; i < orfData.length; i++){
        if(orfData[i].frameIndicator == frameNumber){
            if(orfData[i].frameIndicator >=3){
                if( x >= (((orfData[i].orfStopIndex + orfData[i].frameIndicator) / contigLength) *  canvasWidth) &&
                    x <= (((orfData[i].orfStartIndex + orfData[i].frameIndicator) / contigLength)) * canvasWidth) {
                    displayOrfInformation(orfData[i], i);
                }
            } else {
                if( x >= (((orfData[i].orfStartIndex + orfData[i].frameIndicator) / contigLength) *  canvasWidth) &&
                    x <= (((orfData[i].orfStopIndex + orfData[i].frameIndicator) / contigLength)) * canvasWidth) {
                    displayOrfInformation(orfData[i], i);
                }
            }
        }
    }
}
```

Figure 3.10: Finding if a click if within an ORF Location is as simple as going through the list of ORF Locations, only checking against those within the same frame as the click, then looking at whether the click is within the start and stop points of an ORF Location within the canvas, based on the size and location of where it was painted (for reverse frames, the start and stop points are swapped, as reverse frames are displayed in the same direction as forward frames, with the indexes in the right order for a reverse frame).

When an ORF Location is clicked to be viewed in more detail, the characters within that sequence needed to be formatted to be able to be reasonably displayed. This process took some time to build, and through refactoring was re-written twice over until I found a result that was

reasonably fast at formatting and worked the way I wanted it to. This is that it highlighted every start and stop Codon within a sequence, split the characters into groups of 3, every 15 groups of 3, a new line is to be created and where the start of the line has the index within the contiguous read where that line of the ORF Location begins.

```
function formatOrfSequence(sequence, startIndex, frameIndicator){
    var spacedSequence = new Array();
    spacedSequence.push('<p>' + startIndex + " ");
    for(var i = 0; i < sequence.length; i+=3){
            spacedSequence.push(sequence.charAt(i) + sequence.charAt(i+1) + sequence.charAt(i+2) + " ");
    }
    for(var i=0; i<spacedSequence.length; i++){
        if (spacedSequence[i] == "ATG "){
            spacedSequence[i] = "<span style='color:#339933'><b>ATG </b></span>";
        }
        if (spacedSequence[i] == "TAG " || spacedSequence[i] == "TGA " || spacedSequence[i] == "TAA ") {
            spacedSequence[i] = "<span style='color:#e60000'><b>" + spacedSequence[i] + " </b></span>";
        }
    }
    // Every lots of 15, make a new paragraph and label with current character index.
    for(var i=0; i < spacedSequence.length; i++){
        if(i % 15 == 0 && i > 0){
            if(frameIndicator >= 3){
                spacedSequence[i] = spacedSequence[i] + "</p><p>" + (((startIndex)) - (i*3)) + "    ";
            } else {
                spacedSequence[i] = spacedSequence[i] + "</p><p>" + (((startIndex)) + (i*3)) + "    ";
            }
        }
    }

    spacedSequence.push("</p>");
    var formattedSequence = "";
    for(var i=0; i<spacedSequence.length; i++){
        formattedSequence += spacedSequence[i];
    }

    return formattedSequence;
```

Figure 3.11: Formatting the characters from within an ORF Location into HTML tags to display the data in the way I wanted it to be designed, then inserting it into the div for displaying ORF Location information within the page.

3.2.5.2 Issues

When formatting the ORF Location characters into the list, I found I kept having results where the newline break would be incorrect, or not all of the start Codons were highlighted. This issue was because the way I built the formatted text was by adding text into an array, and then breaking up the text to add newlines whenever the array had a remainder of zero when divided by 15. This meant that it was including the indicator of what index the line of the ORF Location was on as part of the calculation. The result was that the line would be split incorrectly. Once I changed how the formatting was done and included the character index as part of the first set of characters of each line, the calculation worked correctly for splitting the lines where I wanted them to be split.

Although not an issue as such, my first version of drawing the canvas frames involved a single canvas. While the displayed result was the same, it meant that the detecting of user clicks within ORF Location algorithm took longer to process as it had to determine which frame the user was clicking on and then find if it was within an ORF Location. Additionally, having it all in one canvas made the formatting of the canvas results on the page a little more challenging as I couldn't place values (like the frame indicator) exactly next to each frame without painting them within the

canvas.

By just splitting the single canvas into 6 different canvases, one for each frame, it allowed me to easily surround the canvas elements with text for frame indicators and cut down on the code for detecting where and what the user was clicking on within each frame, as shown in the implementation of the click detection above in the section above.

3.2.6 Superframe Comparisson

3.2.6.1 Implementation

Implementing the Superframe was a relatively simple procedure as all of the data required, and part of the code, had already been done within the GC content and ORF charts. Since the Superframe just displays both of these together, it was simply overlaying the 6 frames together and painting on the GC content windows that were out of the threshold (above it, not below) and painting lines to represent the breaks between windows. The algorithm for detecting and highlighting user clicks within windows is also much the same as the code user for the ORF Location clicks and highlighting, just modified to detect within windows rather than within the frames.

The canvas is set up so that it could also be used with other future techniques from within windows, such as k-mer frequency analysis results, and be a tab page where any comparison or overall results could be displayed.

3.2.6.2 Issues

There were no issues implementing this content as the code had mostly been done previously. The main challenge was trying to think of a good way of displaying to the user where there might be potential issues, and giving them a report on the quality. What the application does is just displays them this overlay of the techniques used right now and leaves them to come up with their own decision.

It would be nice in future to be able to expand on this and perhaps be able to truly give the user a report that tells them where there are good or bad areas of their data, instead of making them infer it for themselves through this chart, but without additional techniques this is a challenging prospect.

3.2.7 Implementation Review

During implementation, I found that GC Content percentage and ORF Location finding took a lot longer than expected, especially considering that I implemented an additional part of ORF Location finding that wasn't needed, and refactored it 3 times until I reached the point where it correctly found them.

The user interface design and coding to format it also took a lot longer than I had originally anticipated. I wanted to present the information in a clean and useful way, without cluttering the page, and a number of redesigned happened due to the underlying code evolving over time to meet the requirements. Getting used to Thymeleaf and how it took the data from the Objects took a while to get used to, too, as I was unfamiliar with Spring Boot and Thymeleaf and wasn't sure

how I was adding the data to the Model and then not clear on how it accessed it, or how it was used in form submissions.

The Superframe also needs additional consideration. A lot of the quality assessment is still on the user to look at the produced reports and their content to consider whether they themselves feel their assembly file is good or not through factors highlighted by my application.

Overall, these issues and lack of understandings delayed me to the point where I didn't have time to implement k-mer frequency analysis or any of the 'nice to have' features, and there are certainly areas of the current implementation where I feel bits are lacking, such as file upload, verbose logging and validation and the aforementioned Superframe. What I have got working though, I feel is of decent quality, my process of iterative development and design evolution with prototyping worked quite well and the requirements for those sections are fulfilled.

Chapter 4

Testing

4.1 Overall Approach to Testing

My approach to testing was using Test Driven Development and refactoring, taking note from XP practices, ensuring that I only developed as much as was required to make tests pass based upon my requriements for eachi part of the applications intended functionality. The aim was to have as many tests automated as possible, as this would allow me to test frequently and reliably and tie into testing everytime I checked into my Git repository with CodeShip's continuous integration of my code. Where automation could not be carried out, I created test files and used a test table to determine whether my application met the requirements I could test against.

4.2 Automated Testing

4.2.1 Unit Tests

Writing unit tests before writing any functional code to match against was very useful, allowing me to develop code againsts the tests, written one by one, so that I would not develop any more than necessary. This made is so that what was developed fit the requirements, as the tests written before the code were based upon the exact specifications of the requirements. Refactoring the code after each test kept my design evolving and improving maintainability and simplicity to remove duplicate and unnecessary code.

```
@Test
public void RemoveLowerThanThresholdOrfLocationsShouldRemoveAnyLocationsLowerThanThreshold(){
    _sut.addPotentialOrfLocationToResult(new OpenReadingFrameLocation("ATGCCCCCCCTAG", 0, 12, 0));
    _sut.addPotentialOrfLocationToResult(new OpenReadingFrameLocation("ATGCCCCCCCCCCCCCCCCCCCCCTAG", 0, 26, 0));
    _sut.removeLowerThanThresholdOrfLocations(20);
    assertEquals(1, _sut.getPotentialOrfLocations().size());
}
```

Figure 4.1: An example of a unit test used in my application.

Each unit test was made to only carry out one function check at a time, and the naming of each test reflected what the functionality of the system part under test should do. It is for this reason that the item under test is called 'sut', system under test, and each test is named with a convention

that has a 'should' clause in the name, to demonstrated that the function called should produce particular expected results.

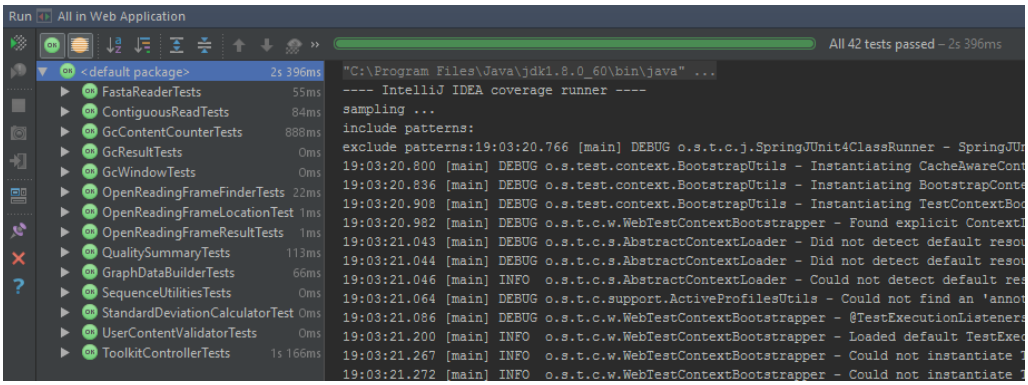


Figure 4.2: Results of running the set of unit tests developed for the application.

IntelliJ gave me the tools to run my unit tests with coverage, allowing me to see how much of my code was actually tested against. In the figure below it can be seen that most of my methods were covered. Those methods that were not covered I chose not to test against, as they are mostly setters and getters with no further functionality. In practice, it would perhaps be best to test against these too, in case for some reason the way a value is set or retrieved had to take into account some processing.

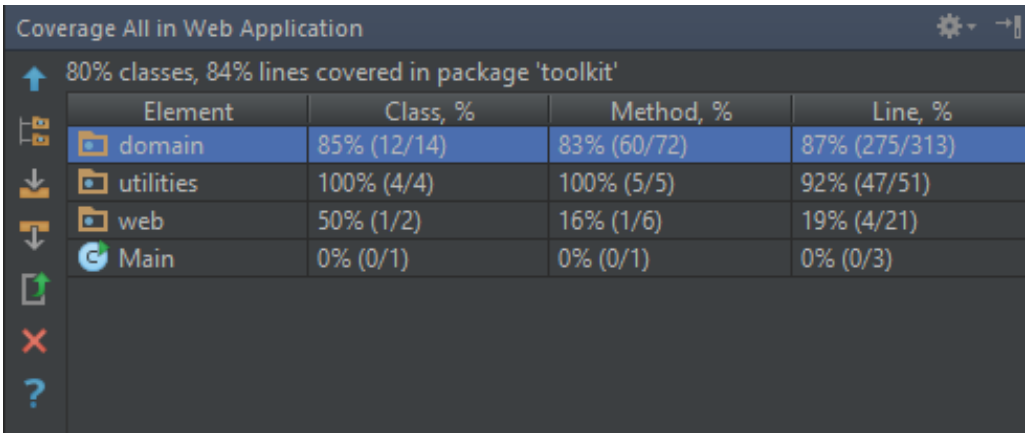
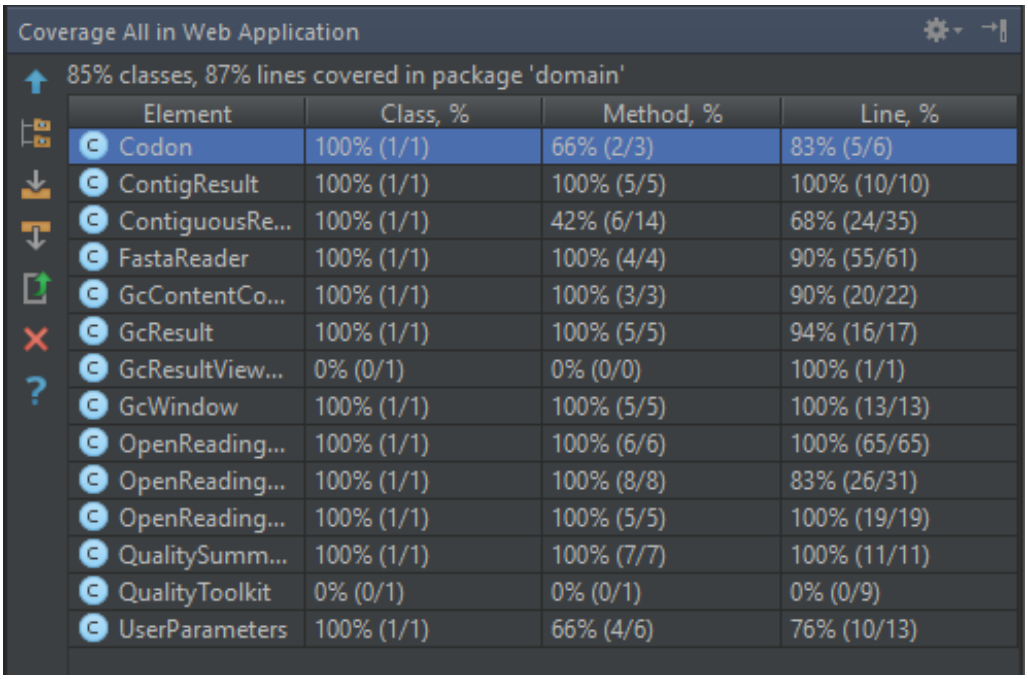


Figure 4.3: The test coverage of my unit tests over the developed application code.

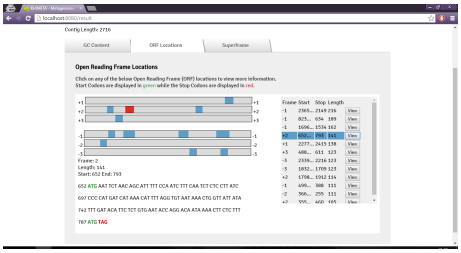


Element	Class, %	Method, %	Line, %
Codon	100% (1/1)	66% (2/3)	83% (5/6)
ContigResult	100% (1/1)	100% (5/5)	100% (10/10)
ContiguousRe...	100% (1/1)	42% (6/14)	68% (24/35)
FastaReader	100% (1/1)	100% (4/4)	90% (55/61)
GcContentCo...	100% (1/1)	100% (3/3)	90% (20/22)
GcResult	100% (1/1)	100% (5/5)	94% (16/17)
GcResultView...	0% (0/1)	0% (0/0)	100% (1/1)
GcWindow	100% (1/1)	100% (5/5)	100% (13/13)
OpenReading...	100% (1/1)	100% (6/6)	100% (65/65)
OpenReading...	100% (1/1)	100% (8/8)	83% (26/31)
OpenReading...	100% (1/1)	100% (5/5)	100% (19/19)
QualitySumm...	100% (1/1)	100% (7/7)	100% (11/11)
QualityToolkit	0% (0/1)	0% (0/1)	0% (0/9)
UserParameters	100% (1/1)	66% (4/6)	76% (10/13)

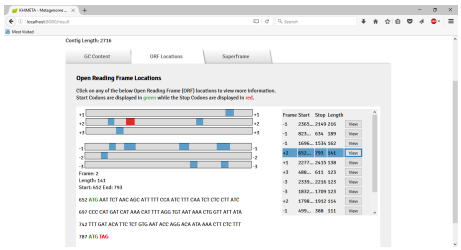
Figure 4.4: The test coverage of my unit tests over the developed application code, broken down for each class.

4.2.2 User Interface Testing

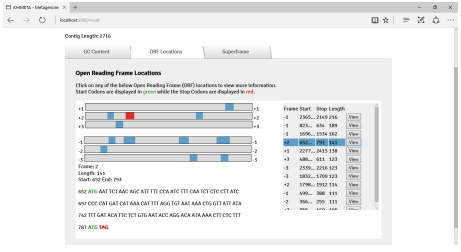
For testing the user interface, I used Chrome's Developer Tools for checking the correct running of the JavaScript, and debugging the code as it ran on loading of the page. I also confirmed that each of my pages loaded and functioned correctly in Google Chrome, Firefox and Microsoft Edge.



(a) Google Chrome.



(b) Firefox.



(c) Microsoft Edge

Figure 4.5: The same page and results from the application shown in different modern browsers.

For testing the speed performance of the web pages of the applications loading time, I used YSlow [10] which runs a number of checks on the page setup and loading speed to return a report about whether a page loads quickly or slowly, and how it could be improved.

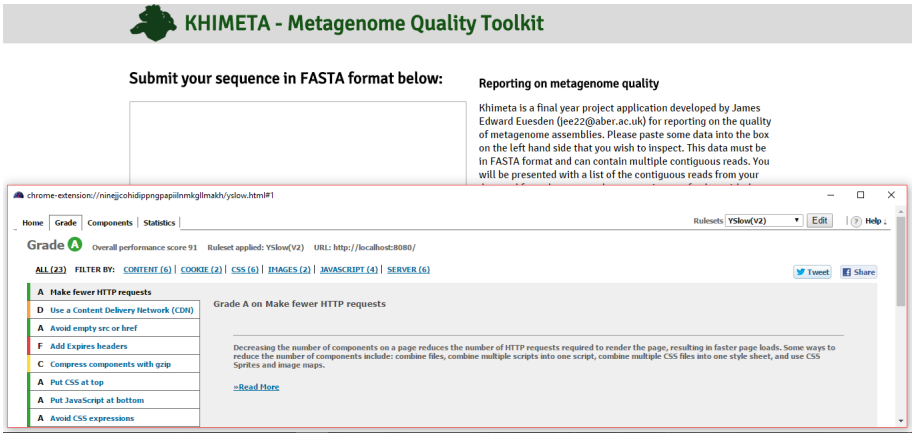


Figure 4.6: YSlow report after running on the Welcome page.

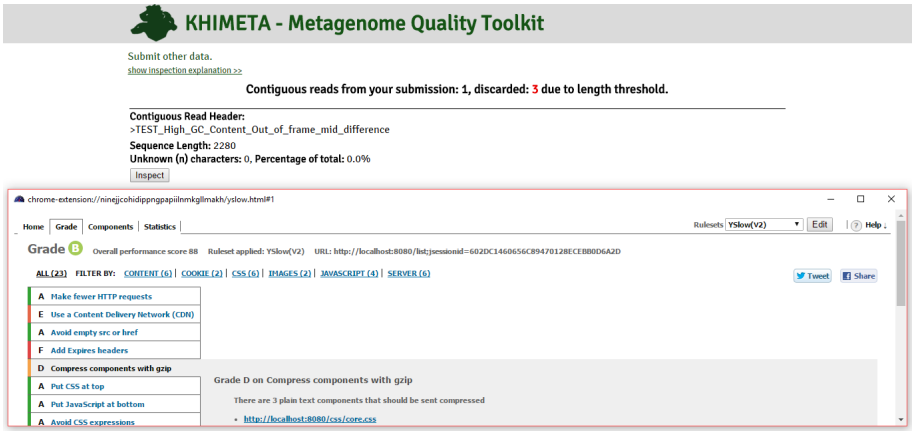


Figure 4.7: YSlow report after running on the List page.

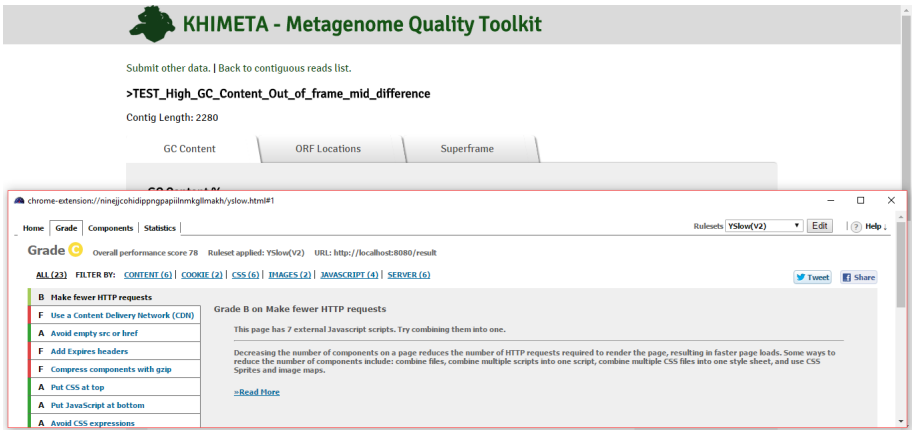


Figure 4.8: YSlow report after running on the Results page.

My results showed me that for the most part there was not much I could do to improve my application outside adding expiry headers to some session variables and implementing a Content Delivery Network (CDN), which was far outisde the scope of the project.

4.2.3 Manual Testing

A number of artificial test files were created in order to be used for testing. Some of these were used in automated tests, such as checking for a files existence and possibility of being read, or throwing exceptions when a test file did not exist while other files were for running manually, either being entirely artificial and checking for individual expected behavours or composed of actual data from multiple species that I manually split and combined together to view the results of. The completely artificial file data is included in the appendices. While the results of the mixed real species data is not included, the results of one very obvious combining of species can be seen in the figure below.

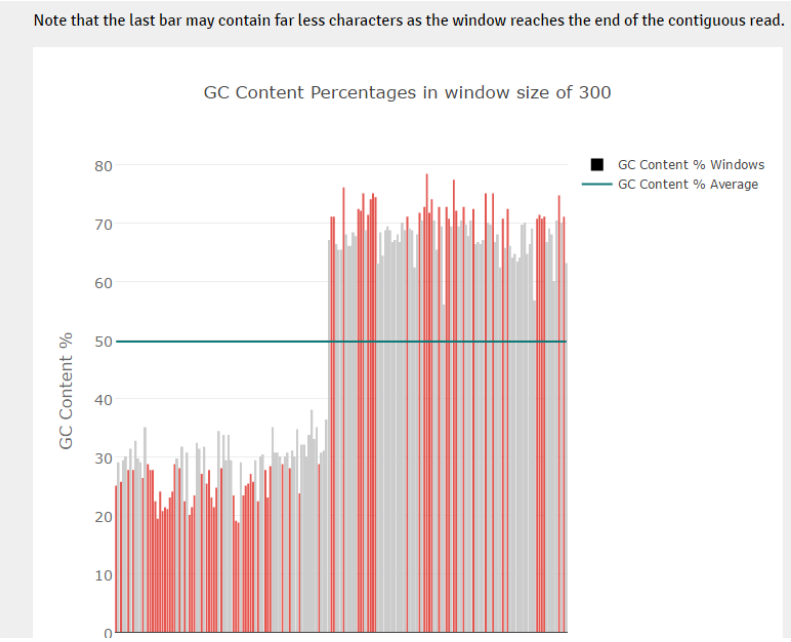


Figure 4.9: Combining two species contiguous reads together at 50% of each of the file (first half one species, second half another species), we see a very obvious split in the GC chart.

Running this test showed me that it is possible to see where there is a huge change in GC content, but that the threshold won't pick it up because with a case like this, while extremely unlikely to happen where there is a mix of only two species, the threshold only shows those outside of the mean, not drastic changes. It highlights that there is room for improvement with detecting these changes and reporting on them to the user, rather than having them infer it themselves.

4.2.3.1 Test Table

For confirming functional requirements were met and testing things that couldn't be done with automation, I produced a test table with tests matching requirements, expected results and avoiding

unwanted results. The tests from the table were carried out by fulfilling the action in the table, and then viewing the results and marking whether the expectations were met or not.

When it came to running large files, or files with many contigs, I ran a number of my laptop and for the most part it handled them quite well with a few hundred contigs of moderate length, although with contigs in the sizes of a hundred thousand characters and up, the process of reading in the contigs and then processing one starts taking noticably longer. I believe this is a restriction with my own machine, however, as should the JVM be allotted enough memory while the application is hosted on a larger, faster machine it is likely it could perform far better.

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Third-Party Code and Libraries

- Jquery - Plotly.js - Spring Boot - Thmyeleaf

1.1 Tabs for results

The code for displaying the results in tabs using CSS3 and Jquery [1]

1.1.1 HTML

```
<ul id="tabs">
    <li><a href="#" name="tab1">One</a></li>
    <li><a href="#" name="tab2">Two</a></li>
    <li><a href="#" name="tab3">Three</a></li>
    <li><a href="#" name="tab4">Four</a></li>
</ul>

<div id="content">
    <div id="tab1">...</div>
    <div id="tab2">...</div>
    <div id="tab3">...</div>
    <div id="tab4">...</div>
</div>
```

1.1.2 CSS

```
#tabs {
  overflow: hidden;
  width: 100%;
  margin: 0;
  padding: 0;
  list-style: none;
}
```

```
#tabs li {
  float: left;
  margin: 0 .5em 0 0;
}

#tabs a {
  position: relative;
  background: #ddd;
  background-image: linear-gradient(to bottom, #fff, #ddd);
  padding: .7em 3.5em;
  float: left;
  text-decoration: none;
  color: #444;
  text-shadow: 0 1px 0 rgba(255,255,255,.8);
  border-radius: 5px 0 0 0;
  box-shadow: 0 2px 2px rgba(0,0,0,.4);
}

#tabs a:hover,
#tabs a:hover::after,
#tabs a:focus,
#tabs a:focus::after {
  background: #fff;
}

#tabs a:focus {
  outline: 0;
}

#tabs a::after {
  content:'';
  position:absolute;
  z-index: 1;
  top: 0;
  right: -.5em;
  bottom: 0;
  width: 1em;
  background: #ddd;
  background-image: linear-gradient(to bottom, #fff, #ddd);
  box-shadow: 2px 2px 2px rgba(0,0,0,.4);
  transform: skew(10deg);
  border-radius: 0 5px 0 0;
}

#tabs #current a,
#tabs #current a::after {
  background: #fff;
```

```
  z-index: 3;
}

#content {
  background: #fff;
  padding: 2em;
  height: 220px;
  position: relative;
  z-index: 2;
  border-radius: 0 5px 5px 5px;
  box-shadow: 0 -2px 3px -2px rgba(0, 0, 0, .5);
}
```

1.1.3 Jquery

```
<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
<script>
$(document).ready(function() {
    $("#content").find("[id^='tab']").hide(); // Hide all content
    $("#tabs li:first").attr("id","current"); // Activate the first tab
    $("#content #tab1").fadeIn(); // Show first tab's content

    $('#tabs a').click(function(e) {
        e.preventDefault();
        if ($(this).closest("li").attr("id") == "current"){ //detection for
         return;
        }
        else{
          $("#content").find("[id^='tab']").hide(); // Hide all content
          $("#tabs li").attr("id",""); //Reset id's
          $(this).parent().attr("id","current"); // Activate this
          $('#' + $(this).attr('name')).fadeIn(); // Show content for the cu
        }
    });
});
</script>
```

Appendix B

Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

Appendix C

Examples and Extras

3.1 Example User Story Breakdown

NOTE: This break down is from my blog entry at `http://users.aber.ac.uk/jee22/wordpress/?p=147`.

```
As a researcher
I want to get a report on the quality of my metagenome
So that I know whether it is of good or bad quality
```

Okay, super high level. This can be broken down into:

```
As a researcher
I want to get a report on the GC content of my metagenome
So that I can see where there might be inconsistencies
```

So, that could be explained better (i.e. what are inconsistencies? Areas where there might be a split/chimera, or just gene encoding regions and completely natural).

```
As a researcher
I want descriptions of the GC content of my metagenome
So that I can pinpoint areas of interest
```

Perhaps a better way of making a story for GC content in this instance. What about the report?

```
As a researcher
I want a textual and graphical description of my metagenome quality
So that I can see and understand where there might be quality issues
```

Again, quite high level, but not too bad. This could be broken down further.

```
As a researcher
I want a graph plotted to show me the GC content in my metagenome
So I can visualise the distribution of GC content to better understand my me
```

From some of these, further tasks can be broken down, so, lets take one and do that with the last story I defined. I suppose, before we can do that though, since we dont have an application developed, we might need some initial setup stories.

```
As a researcher
I want an application to read in my metagenome assembly
So I can see it outside of the FASTA file
```

Maybe thats pushing it a little. Theres not really much to be gained from this in business value, but, as far as development goes it can give us some nice little tasks:

```
Read in FASTA file
Output display of metagenome visually for researcher to understand
```

Thats just two simple tasks. Read in a file type, and with the contents, display it. It might not be much, but its a start where we can say to a hypothetical researcher Okay, weve taken your file, and we can show you that your metagenome looks like this. Theres no processing done to it, but you can see how with this visualisation, theres the room for labelling and noting the interesting points later. What do you think?

This story with others could then help us propose a Sprint Goal, as below:

Sprint Goal: To display a metagenome in an application after reading in a FASTA file, with the look at implementing GC Content counting should time allow.

3.2 Artificial Test File

```
>TEST_High_GC_Content_Out_of_frame_mid_difference
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
```

```
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
>TEST_Low_GC_Content_In_Frame_Throughout
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
ATGATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTTGA
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
>TEST_Sinlge_Frame_Many_Start_Stop_Codons
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
ATGGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCATGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGATGCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
ATGATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATT
ATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTATTTGA
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCTAGGCCCGGGCCCGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGTAAGGGCCCGGGCCCGGGCCCGGGTGAGGG
CCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGGCCCGGG
>TEST_50_percent_N_characters
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
TTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAA
TTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAA
TTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAA
TTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAATTTGGGCCCAAA
```

Annotated Bibliography

[1] Catalin Rosu, "CSS3 & jQuery folder tabs," [Online] Available: http://red-team-design.com/css3-jquery-folder-tabs/, 2012, [Accessed on: 29 April 2016].

The tutorial and code for creating tabs using CSS3 and Jquery that I used in order to implement tabs for the inspection results page.

[2] F. Cirillo, *The pomodoro technique*. Lulu, 2009, 1445219948.

The pomodoro technique, used during the development of my application for efficient time management.

[3] Codeship Team, "Continuous Delivery with Codeship: Fast, secure and fully customizable." [Online] Available: https://codeship.com/, 2016, [Accessed on: 27 April 2016].

CodeShip, the web service I used to help provide me with contiurous integration for my project, connected to my GitHub repository to be run every time I checked in to the repository. This was integral to developing my application as it highlighted any time that I broke my build through forgetting to modify tests when content was updated, or when I broke parts of the application and so old tests would fail.

[4] N. C. B. I. R. Coordinators, "Database resources of the National Center for Biotechnology Information," *Nucleic Acids Research*, vol. 41, no. D1, pp. D8–D20, Jan. 2013.

The NCBI database and resources was extremely useful to me while developing my application. It gave me access to papers and research articles, data sets for downloading sequences to play with and explore and tools for techniques I wanted to implement in my own application. It's safe to say that without the existence of the NCBI resources, I would have understood a lot less about my project while beginning my development.

[5] D. Huson and S. Mitra, "Introduction to the Analysis of Environmental Sequences: Metagenomics with MEGAN," in *Evolutionary Genomics*, ser. Methods in Molecular Biology, M. Anisimova, Ed. Humana Press, 2012, vol. 856, pp. 415–429.

A useful chapter in reading about a tool that has its own attempt at analysing metagenomic data. While the tool does not look at the quality, it looks at what the user provides and indicates to them if anything they have is already existing in reference data, and how their data compares to similar sequences already known. This could be considered as a quality measure, as if a sequence can be found to match a reference sequence, we know it is most likely not chimeric.

[6] P. T. Inc. (2015) Collaborative data science. Montral, QC. [Online]. Available: https://plot.ly

Plotly.js, the JavaScript library I used for creating the GC content window chart.

[7] V. Kunin, A. Copeland, A. Lapidus, K. Mavromatis, and P. Hugenholtz, "A Bioinformatician's Guide to Metagenomics," *Microbiology and Molecular Biology Reviews*, vol. 72, no. 4, pp. 557–578, Dec. 2008.

This paper was invaluable to me at the beginning of the project, understanding what metagenomics is, and how the quality of an assembly can be full of errors, and why. While much of the paper was not necessarily relevant to my project as it deals with the full process of working with a metagenomic sample, for my background reading it was very useful.

[8] B. Liu, Y. Shi, J. Yuan, X. Hu, H. Zhang, N. Li, Z. Li, Y. Chen, D. Mu, and W. Fan, "Estimation of genomic characteristics by analyzing k-mer frequency in de novo genome projects," Aug. 2013. [Online]. Available: http://arxiv.org/abs/1308.2012

A paper highlighting the usefulness of k-mer frequency analysis and distribution in determining characteristics of genomes. Similar to my projects objectives where I wished to have my project use k-mer frequency analysis for quality assement.

[9] N. J. Loman, C. Constantinidou, M. Christner, H. Rohde, J. Z. M. Chan, J. Quick, J. C. Weir, C. Quince, G. P. Smith, J. R. Betley, M. Aepfelbacher, and M. J. Pallen, "A Culture-Independent Sequence-Based Metagenomics Approach to the Investigation of an Outbreak of Shiga-Toxigenic Escherichia coli O104:H4," *JAMA*, vol. 309, no. 14, pp. 1502+, Apr. 2013.

This paper was useful in understanding what the study of metagenomics can be used for, and so helped me better understand what metagenomics is and how it could be useful to have a tool for quality control.

[10] M. Duran, "YSlow - Official Open Source Project Website," [Online] Available: http://yslow.org/, 2016, [Accessed on: 30 April 2016].

YSlow, a tool for performance testing the loading of html pages, which I used for testing each of the pages of the application.

[11] G. Marçais and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics*, vol. 27, no. 6, pp. 764–770, Mar. 2011.

Jellyfish, a tool for k-mer frequency anaylsis. A tool I looked into briefly at the beginning of my project when considering what techniques to use for my application.

[12] P. Melsted and J. K. Pritchard, "Efficient counting of k-mers in DNA sequences using a bloom filter," *BMC Bioinformatics*, vol. 12, no. 1, pp. 333+, 2011.

BFCounter, for k-mer frequency analysis. A tool I considered when thinking about the requirements and techniques involved in the development of my application.

[13] Mike Cohn, "Extreme Programming: A Gentle Introduction." [Online] Available: http://www.extremeprogramming.org/, 2016, [Accessed on: 26 April 2016].

The description of Extreme Programming (XP) that helped me understand the XP methodology and adapt it for my own use in this one person project.

[14] ——, "User Stories and User Story Examples by Mike Cohn," [Online] Available: https://www.mountaingoatsoftware.com/agile/user-stories, 2016, [Accessed on: 26 April 2016].

An explanation of how a User Story should look, when utilizing them to plan tasks for use with implementing functions of my application.

[15] Phillip Webb, Dave Syer, Josh Long, Stphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sbastien Deleuze, "spring-projects/spring-boot," [Online] Available: https://github.com/spring-projects/spring-boot, 2016, [Accessed on: 27 April 2016].

The GitHub repository of Spring Boot and documentation for using Spring Boot. Used by me in order to build my application as a web service with Java using Spring.

[16] R. Schmieder and R. Edwards, "Quality control and preprocessing of metagenomic datasets," *Bioinformatics*, vol. 27, no. 6, pp. btr026–864, Jan. 2011.

A tool that does what my project topic describes, with more understanding than I had at the start of my project. An interesting look at what other software is out there to do what my application also wanted to do and was a baseline for me analysising what sort of techniques I may wish to use to give the user of my own application a report on the quality of their assembly.

[17] K. Schwaber and J. Sutherland, "The Scrum guide," 2001.

The Scrum Guide, useful for knowing how to use the Scrum framework as part of my projects lifecycle model.

[18] J. C. Segen, *The dictionary of modern medicine*. Parthenon Pub. Group, 1992, 1850703213.

For the reference of what GC Content is. The most referenced link seen else where online (e.g. from Wikipedia) is to a web link that no longer exists. This reference backs up the description of GC Content, and is how I understand what GC Content is outside of Wikipedia's reference to the missing web link.

[19] Tatiana Tatusov and Roman Tatusov, "ORF Finder," [Online] Available: http://www.ncbi.nlm.nih.gov/projects/gorf/, 2016, [Accessed on: 25 April 2016].

A tool from NCBI for finding Open Reading Frames within a pasted sequences. I liked using this to compare my own ORF results against and get an idea of how I might want to display my own results in the quality report too. It was also nice to use for helping me understand what an Open Reading Frame actually is.

[20] The Board of Regents of the University of Wisconsin System, "Translation and Open Reading Frames," [Online] Available: http://bioweb.uwlax.edu/GenWeb/Molecular/Seq_Anal/Translation/translation.html, 2008, [Accessed on: 25 April 2016].

I started to understand what an Open Reading Frame was from reading this web page. It helped me get a grasp of what ORFs were, how they were constructed and what I should do to write code for my own application in how to find ORF Locations.

[21] The Thymeleaf Team, "thymeleaf/thymeleaf," [Online] Available: https://github.com/thymeleaf/thymeleaf, 2016, [Accessed on: 27 April 2016].

The GitHub repository of Thymeleaf, for use with Java Spring Boot. Invaluable for me when accessing objects from the Model via the View of my web service.

[22] T. Thomas, J. Gilbert, and F. Meyer, "Metagenomics - a guide from sampling to data analysis." *Microbial informatics and experimentation*, vol. 2, no. 1, pp. 3+, 2012.

A useful article in understanding what metagenomic study was. It was helpful to read as I tried to understand metagenomics at the beginning of the project.

[23] J. D. WUITSCHICK and K. M. KARRER, "Analysis of genomic g + c content, codon usage, initiator codon context and translation termination sites in tetrahymena thermophila," *Journal of Eukaryotic Microbiology*, vol. 46, no. 3, pp. 239–247, 1999. [Online]. Available: http://dx.doi.org/10.1111/j.1550-7408.1999.tb05120.x

An article that includes a discussion of the increase in GC count within coding regions. Useful for me to understand why using Open Reading Frames could be good for a user to see GC regions that were higher with potentially explainable reasons within their contiguous reads.

[24] Zhang Lab, "FASTA format," [Online] Available: http://zhanglab.ccmb.med.umich.edu/FASTA/, 2016, [Accessed on: 25 April 2016].

A description and example of what a FASTA file is and what they do. Gives a nice example of a fake FASTA file and helped me know what to expect with the file format when developing my application.