Building an Online Resource for *Candida tropicalis*

Final Report for CS39440 Major Project

Author: Owen Garland (owg1@aber.ac.uk)
Supervisor: Dr. Wayne Aubrey (waa2@aber.ac.uk)

May 6, 2017
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BEng degree in Software Engineering (G600)

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

Abstract

Understanding data produced by next gen sequencing technologies is a difficult problem, not helped by a lack of well designed software and user experiences. Researchers at Aberystwyth University are looking into the possibly of modifying a species of yeast, *Candida tropicalis*, to convert xylose into xylitol. If xylitol can be produced by *C. tropicalis* it would provide a cheaper supply of the sugar, which has antibacterial properties, as well as a very low glycaemic index, making it suitable for people suffering from diabetes.

To do this they have sequenced the DNA of three species of yeast, *C. tropicalis, C. shehate* and *C. boidinii*, but now need an accessible and meaningful way to interact with the data produced. Most genome database software is already over a decade old, and continue to use dated practices, and technologies. The aim for this project is to apply modern web development techniques to this problem and produce a web site to display their data using NodeJS, with the data stored in a NoSQL database.

The implementation of this database and website has shown that NoSQL is a viable approach to storing and viewing genomic information, especially for smaller genomes. However it does also acts as a proof of concept and provides a step stone for the building of a larger open source project that aims to accommodate larger genomes. Importantly it shows that modern technologies and approaches can have a meaningful benefit over the old monolithic solutions, especially in terms of producing high quality maintainable software.

CONTENTS

LIST OF FIGURES

Chapter 1

Background & Objectives

1.1 Background

As part of the BEACON Wales[2] project, researchers at Aberystwyth University have been studying three species of yeast, *Candida tropicalis*, *Candida boidinii*, and *Candida shehatae*. To study the genetics of these species, they have each undergone Illumina sequencing producing a library of short reads between 80 and 120 bp in length. Each library of reads for each species was then assembled into contigs (contiguous DNA fragments). This process produces an approximate representation of the underlying species DNA, which then can then be compared to other better understood genome databases and annotated accordingly.

Arabinose and xylose are five carbon sugars ubiquitously found in plants such as grass, which can be used as a feedstock for industrial biotechnology. *C. tropicalis* is able to convert arabinose and xylose into arabitol and xylitol respectively. Xylitol is a commercially valuable anti-bacterial food grade sugar use in the manufacture of chewing gum.[3] However arabitol and xylitol are stereo isomers and cannot be easily separated. Unlike *C. tropicalis* and *C. shehatae*, *C. boidinii* is unable to utilise arabinose (for an unknown reason), but does process xylose into xylitol, just at a commercially unviable rate.

Understanding at a genomic level why *C. boidinii* is unable to utilise arabinose and genetically modifying *C. tropicalis* to have the same phenotype, may enable *C. tropicalis* to exclusively produce xylitol and not arabitol. This would provide a cost effective way to produce a source of xylitol, which can be used for many applications. One exciting possibility is using it as a low glycaemic index table sugar that can be used by those suffering from diabetes.

As Sboner et al. highlight in their paper "The real cost of sequencing: higher than you think!"[4] the cost of sequencing DNA is falling faster than Moore's law, which is in turn increasing the amount of data being produced from sequencing. The real cost of sequencing DNA is now in processing the sequence data into meaningful results, that can be studied in a wet lab experiment.

Currently researchers have the genetic information for the three species of yeast in the form of a FASTA file containing thousands of contigs. These files are large >16MB text files, making it difficult to extract meaningful information quickly. To make predictions about which genes encode which proteins in yeasts they will need to be able to search and browse this data in a much more accessible form.

This is why a website representing this data, and offering search functionality would be advantageous to them, as it would provide a simple and familiar user experience to other web based genome browsers such as the Candida Genome Database,[5] which they are already familiar with and regularly use as a reference.

1.1.1 Existing Databases

As the core of the project focuses on having the data stored in a database, the first action was to investigate the current database solutions that had been developed for genomic information.

The most well established database is CHADO,[6] which was initially developed back in 2005 for the FlyBase[7] project. It has since then grown to accommodate information about other model organisms such as plants, and other complex multicellular animals. It uses PostgreSQL[8] to store it's data and Perl[9] to setup and maintain the database. Due to it's monolithic approach of being a one size fits all solution, it has over 200 tables in a very complex schema. This makes working on it rather difficult, as there are many relations to consider when entering and manipulating data from the database.

When installing and populating CHADO, it was clear that the project wasn't very well maintained as during the installation several of the Perl modules that it depends on were not able to successfully build, due to bugs in the modules causing the tests that ensure the build was made correctly to fail, and thus not install. This meant several modules had to be manually fixed just to get the application to install correctly. Additional complexity of the systems architecture stemming from it being an older project and not having the active surrounding ecosystem that other platforms may have, increases the time taken to develop for and maintain the database.

This combined with the fact that Perl now has less market share[10] due to the emergence of Python[11] and JavaScript[12] among other scripting languages, has resulted in fewer online resources available for Perl developers when compared to newer scripting languages. This is mainly due to newer languages having more popular package management, ecosystems and syntax.

Developing the application in Perl to work with these existing platforms, would be a viable option, but this project is also an opportunity to show that genomic data can be adequately handled by newer technologies such as JavaScript and NoSQL databases.

1.1.2 Alignment Tools

The data provided by the client in this project was insufficiently documented and so it was deemed prudent to investigate how the existing data was produced. This would ensure that any new data added throughout the project was interoperable, and could be easily produced.

The original alignment tool BLAST[13] was developed in the 90's and is very widely accepted as the *de-facto* standard for performing alignments.

Alignment is when a DNA, or protein sequence is compared against a database of known protein sequences that have already been determined via previous research. This allows a researcher to match their sequences against known genes and annotate them accordingly. These annotations aren't proof that genes with similar sequences have the same function, but it is a good starting point. To prove it the researchers will test it in a lab experiment, however as there are many thousands of potential genes to test it is a lot more efficient to infer function via sequence similarity

before undergoing wet lab experiments.

Aligning two sequences of DNA is a computationally intensive task, that there have been many efforts to improve the efficiency of. One area that has been pursued is the use of GPU's to perform alignments in parallel, utilising hundreds or thousands of cores of the GPU, compared to the tens of cores available on modern CPU's.

Initially Diamond,[14] a CPU only tool, was tested and was able to align the data sets against the NCBI[15] non-redundant protein database[16] using the blastx algorithm to align the nucleotide sequences against the known proteins. This took about an hour and a quarter for each species, this was only using the CPU and may have been bottle necked by the mechanical drives being used for storage. It is important to note the hardware being used for this, as the time taken is determined by the hardware. The following hardware was used, Intel i7-6700k @ 4.6Ghz, 16GB of RAM, a 7200rpm Hard Disk Drive, and a Nvidia GTX 980.

As there was a high performance graphics card available it would be advantageous to make use of a GPU accelerated alignment tool to make the most of the hardware available, and reduce the time that any future alignments may take.

The first attempt to use a GPU powered tool was BarraCUDA,[17] which installed successfully and appeared to run fine, however it would attempt to read in the entire reference database into memory. This was an issue as the NCBI non-redundant database is around 66GB in size, and the system that was being used only had 16GB of RAM available.

The next tool tested was CLAST,[18] which unfortunately had compilation errors on the system. Debugging this was out of scope for my project as in this initial stage it didn't seem necessary. Next tested was Cudasw++[19] this did install, but ran into the same issues as BarraCUDA, being limited by the system memory available.

In an attempt to get GPU acceleration working, the NCBI nr database was split up into 10GB chunks for use with these tools, however even after being reduced in size, BarraCUDA and Cudasw++ both had errors and failed to complete an alignment.

It was decided that although the speed gains of a GPU accelerated alignment tool would be significant, the Diamond alignment was quick enough to not be a major bottleneck for the project, when compared with the extra time and effort that would have to be put into investigating how to get the GPU alignments to work.

1.2 Analysis

From researching the current solutions to the problem of annotating, storing and presenting genetic information, it was clear that a complete solution using modern web development technologies such as Node.js wasn't currently in the public or open source domain.

Of the open source projects that were currently in use the majority appeared to be very old and monolithic. This is often due to their integration with the most popular genome viewers GBrowse[20] and JBrowse[21] which are both written in Perl.

Perl is less suited to modern web development than some of the newer languages, mainly due to it's smaller market share[10] and greatly smaller number of available third party modules, and support.

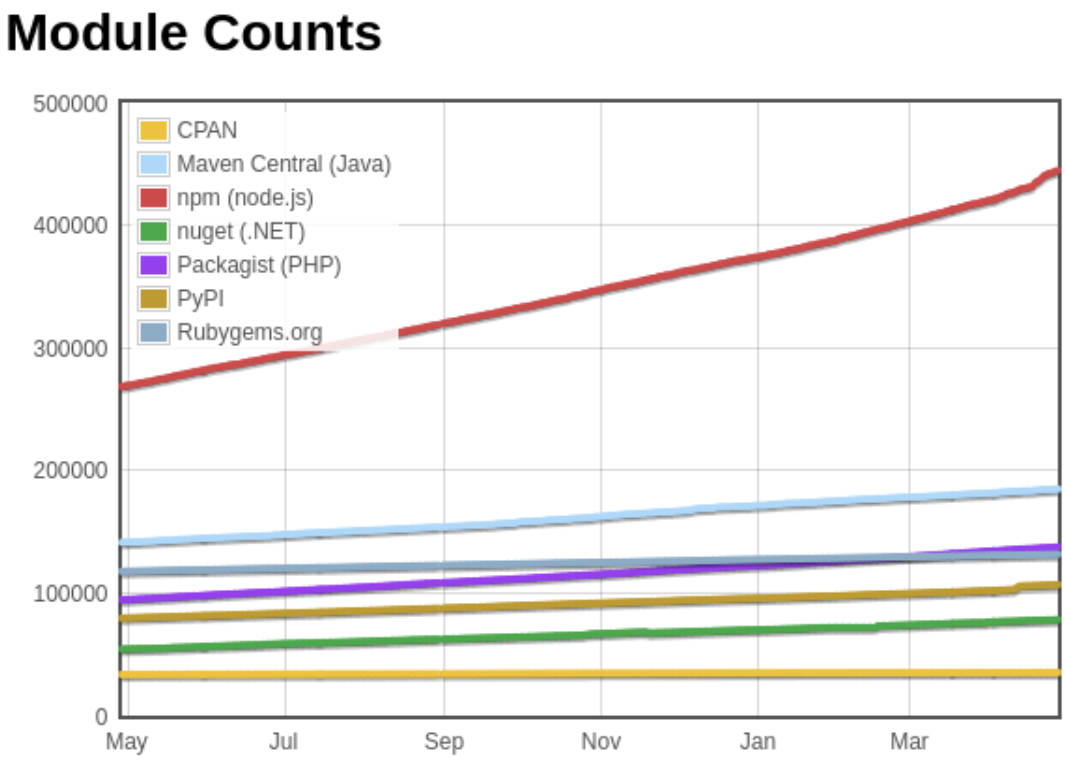

Figure 1.1: Number of modules available for server side languages, Perl (CPAN), Java (Maven), JavaScript (npm), .NET (nuget), PHP (Packagist), Python (PyPI), Ruby (Gems)[1]

Web development in Perl is clearly on the decline as seen by it's market share,[10] and that means that the ecosystem surrounding it just can't compete with newer technologies such as Ruby on Rails or Node.js.

1.2.1 Analysis of the problem

Looking at the data that I was provided with there was three clear stages to the development of my solution. The first would be to ensure that the data I had was valid and in the same format for each of the species that were being analysed. The next was to import that data into a database, and then from there develop a web front end to interact with the data in the database.

An alternative approach to this project would be to use one of the existing databases such as Intermine[22] or CHADO,[6] and then use a web front end compatible with those databases such as GBrowse[20] or JBrowse.[21] This would require changes to how the data was processed before going into these established systems, however the benefit of using this approach would be access to all of the bioinformatics tools that are compatible with these platforms.

As a computer scientist looking at this problem, there doesn't seem to be much need for a very complex solution. Once the data is generated, it just needs to be made into a database friendly format and then stored in a database. From there building a simple web front end to make the database accessible is a simple task. There isn't any need to manipulate or modify the data, simply

create and read it. As the requirements for the system were so simple, one of the more advanced projects like CHADO and GBrowse, didn't seem necessary, as although mature and well supported in Bioinformatics they come with a lot of extra features that wouldn't be required and just add bloat to the project.

1.2.2 Comparison of SQL and NoSQL databases

Structured Query Language (SQL) databases such as MariaDB, PostrgeSQL and SQLite, store data in a relational manner, this means that data is stored across many linked tables, that are grouped by the data type. They are the interfaced with via SQL to select related data from multiple tables to return a meaningful result.

- SQL based systems have been used for a long time and have a large amount of support and compatibility with different frameworks.
- Efficient storage of data, as data is put into first normal form and not replicated.
- Complex queries can be ran with relative ease.
- Requires a strict schema.
- Development is more complex as you have to manage and maintain relations.

NoSQL databases store data in collections of documents. Each collection is usually linked to a use case for the data, for example a website that has blog posts would have a 'blog' collection, that stores all of the data for blog posts. The collection is made up of individual documents representing individual items, so the 'blog' collection will store lots of individual blog posts. Each document is then made up of key value pairs, commonly represented as JSON.

- Collections are based on use cases so no complex selection logic.
- Data is represented as JSON so it is very interoperable with JavaScript.
- Dynamic schema, not every document has to have the same information as the other documents in the collection.
- Newer technology so less legacy support.
- Data can be stored twice which is a less efficient use of disk space.

The decision on which technology to use is discussed in the Design section.

1.3 Research Method and Software Process

It was intended to use a blend of agile and extreme programming methodologies for this project. The idea was to ensure that as the understanding of the task at hand increased and new requirements or changes to the structure of the application became necessary, an agile workflow would allow adaptations to be made without setting the project's progress back.

For this project there were clearly defined clients that could be accessed and used for feedback during development iterations. David Bryant and Abhishek Somani, the scientists conducting the research, were available for meetings to provide feed back and discuss progress of the project.

When suitable we would meet on a Friday at the end of an iteration to discuss the projects progress and any difficulties that may have been encountered. Towards the end of the project this time was used to get detailed feedback on how the website could be improved. Unit testing was also used to develop the key algorithms, as well as continuous integration and deployment, this process is described in much more detail later on in the document. It was key to being able to deliver working software at regular intervals and demonstrate it to the clients.

Once requirements were gathered a Kanban board would be made with each individual task listed, based on the requirements of the application. Those tasks would then be weighted with an estimated difficulty, and then worked on. Once development had started on each task it would be moved to the ”In progress” board, and then across to the ”Done” board, once completed.

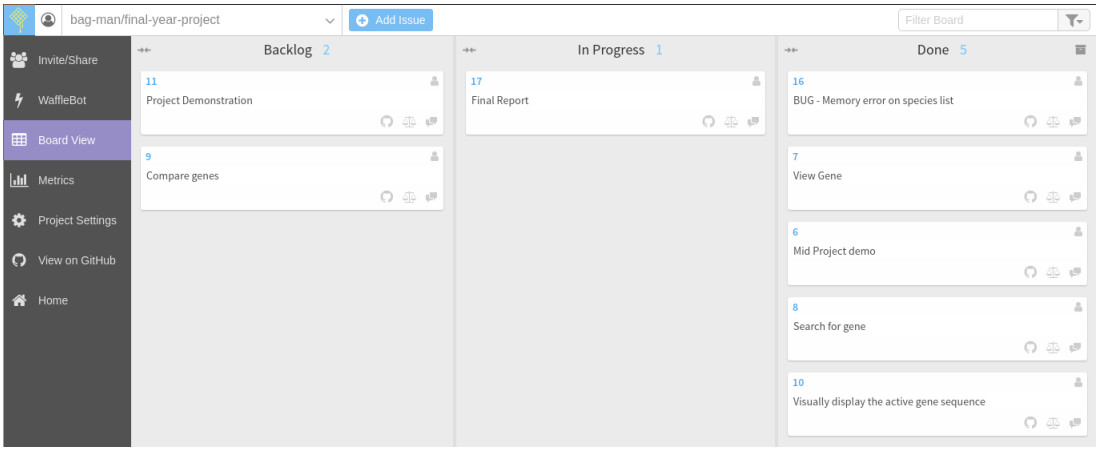

Figure 1.2: Kanban board in use. Waffle.io is the service providing the board

However as this project involved a lot of domain knowledge that wasn't familiar, there was a very significant amount of time spent learning what the data meant and how it should be used. During that time I was prototyping potential solutions to how the data could be interpreted, without a specific set of requirements in mind. This lead to very vague requirements being created and development following a path closely linked to the data and the understanding of it.

An example of this process was the reverse compliment feature, that didn't appear to be necessary until a couple of weeks before the project deadline. This is exactly why a traditional structured development process like waterfall would not have worked for this project, as new requirements and features were appearing deep into the project, something that wouldn't have been able to be accommodated for if using a strict waterfall model, where each phase of development is scheduled a head of time.

1.4 Provided data

The data for this project was produced by a contracted researcher who had moved onto another project. Because of this there was limited opportunity to talk with them about the data, how it was produced and what needed doing to it. Unfortunately, the data did not include any form of documentation, or annotation as to how it was produced, the only metadata available was the filenames.

This lead to a lot of confusion as it wasn't immediately obvious what each bit of data was and what it meant. Examining the data and learning about the biology behind it took several weeks, and ascertaining what bits of data were actually relevant to the project took even longer. Eventually it was determined that for each species there were three files that would need to be incorporated into the project.

- Raw assembled contigs of DNA, these would be used to get a genes position in the genome, and it's surrounding bases.
- Protein sequences found from using BLAST, annotated with Gene Ontology ID's.
- Coding sequences, these are the nucleotide sequences that coded for the proteins.

Linking this data to the Candida Genome Database (CGD) was a key bit of information that was missing. To do this the annotated proteins[23] for *C. albicans* were downloaded, then turned into a reference database with Diamond. With this the three species could then have their coding sequences aligned against *C. albicans* to produce accession ID's from the Candida Genome Database. The script that performed this step is included in the project as `run_diamond.sh`.

Several sets of data were then required to map these proteins to the Candida Genome Database, which would allow the researchers to easily compare the genes to well annotated species. First was a file of GeneOntology[24] annotations for all the proteins stored in the Candida Genome Database.[25] This file conveniently also stores all the accession ID's for those proteins. This is what was used to map the alignment results to the Candida Genome Database, by selecting the accession ID from the Diamond alignment results, and looking that up in this data set, the Candida Genome ID was able to be found.

In addition to this was a mapping file that mapped Candida Genome Database ID's to UniProt[26] ID's. With a small JavaScript script and some manual cleaning with a text editor, a complete mappings file was able to be produced in JSON, ready to be read in by the importation script. This allows for genes found to be linked to other well known genes in the Candida Genome Database and UniProt.

Chapter 2

Software Design, Implementation and Testing

2.1 Requirements

There are two main components to the software being developed for this project. The first is the manipulation of the data into a database, and the second is a web front end for the data to be accessed via.

2.1.1 Database

The database has to store the following information for each hit from the alignment with *C. albicans*:

- ID of the hit that was found when the species coding sequences have been aligned with *C. albicans*.
- Species it came from.
- The gene name and ID from Candida Genome Database.
- UniProt ID.
- Contig of raw DNA.
- Coding sequence of nucleotides.
- Protein sequence that the coding sequence codes for.
- Annotations from NCBI nr database to provide a description of the proteins.
- A list of Gene Ontology[24] ID's for the protein.
- If it can be found, the positions of the start and end of the coding sequence in the raw contig.

From these requirements it is clear there are a few pieces of data that need to be extracted from the data that has been collated. For each alignment with *C. albicans* the protein sequence and GO annotations need to found, the coding sequence that was used in the alignment and the contig that it came from. In addition to this, the metadata about the genes such as the name and ID's need to be read from the mapping file that was created earlier.

The final bit of data that needs to be added is the position of the coding sequence in the contig. An algorithm will have to be developed to detect this.

2.1.2 Website

The website will only have a few funcitonal requirements:

FR1 Display each record in the database in an easy to read manner.

The first requirement, is that each document in the database can be accessed, and presented to the user. It will have to display all of the fields of the document without overwhelming the user, this is a risk as some of the sequences are very large. Links to other databases such as the GeneOntology database, and UniProt database will have to be made into hyperlinks for the user to easily navigate to the relevant information for the found gene.

FR2 Highlight coding sequence in contig if possible.

Visually representing the coding sequence inside of the raw assembled contig, is helpful for researchers to see the surrounding bases up and down stream of the coding sequence. It also allows them to see the full coding sequence with any introns (non-coding regions of bases) included.

FR3 Copy coding sequence and +/- a user defined number of bases to the clipboard.

The user should also be able to copy the coding sequence to their clipboard, for use in other applications. They should also be able to select a number of bases to be copied that surround the coding sequence.

FR4 Search the database for a gene, by name, description, ID, or even by nucleotide sequence.

The search functionality is the main feature of the site as it is what takes the previously difficult to use data, and makes it truly accessible.

2.2 Build Process

One of the first things that was setup for this project was the build process. This doesn't take long to setup but provides a huge benefit for the rest of the project. It helps to reduce errors and speed up the process of deployment and testing.

2.2.1 Development Environment

This project will be developed on an Arch Linux system, as it will only ever be ran on a Linux host this isn't an issue, as the production hosts' architecture will match the development one.

All of the code and LaTeX form the report will be written with the Vim text editor, which has been configured to have many optimisations to this workflow. The full Vim configuration that was used can be found here.[27]

2.2.2 Code Style

For this project the Clock Limited code standard[28] will be followed. This has a few key variations to typical Javascript.

- Semicolons are not required in JS, so they are not used to denote end of line.
- Strings are encapsulated with single quotes, not double.
- Comma first[29] listing of variables, objects and arrays.
- No commenting, to avoid confusion, encourage descriptive naming and reduce file length.

These rules ensure that the code written is clean and easy to read, something that is crucial for producing high quality and easily maintainable code.

2.2.3 Version Control

This project is being entirely tracked with Git version control. This is to ensure that all of the work is backed up to a remote repository, with an easily accessible history. It also allows for branching of the project to test out experimental features or bug fixes. The project is remotely hosted on Github[30] for remote backup and access via multiple hosts. Github is also widely supported by third party integrations, something that will be utilised quite extensively, in this build and deployment process tool chain.

2.2.4 Test Suite

When pushing code to Github, a module called Husky[31] will run the test suite on the developers local machine.

The test suite is set up to first clean the project of old builds and coverage reports, then run ESlint.[32] This will scan the entire codebase for files that do not adhere to the code style. This means that any badly formatted code will cause an error and prevent the code from being pushed until the issues are fixed.

After the linting check, the test suite will use Mocha[33] to run all of the JavaScript test files that are in the project. If any of the assertions made in the test files fail, the project will not be pushed.

Once the test's have been ran Istanbul[34] will provide a coverage report, that will show to the user the percentage of tested code that has been covered by the assertions. After this the code can finally be pushed to Github.

2.2.5 Continuous Integration

Once the code has been pushed to Github, a hook on Github will trigger Travis CI[35] to clone down the latest version of the code to it's servers and run the test suite again on it's own server. This is very useful as occasionally developers can bypass Husky to push code, or have something setup in their system that isn't defined in the application that causes test to pass on their system but not when the application is built from scratch. The CI test will expose any bugs that arise from these situations and prevent the code being deployed to the testing server.

Once those tests are successful, it will send the coverage report generated by Istanbul to another third party service called Codecov.[36] This tracks the test coverage over time and provides interactive charts highlighting areas that need more test coverage.

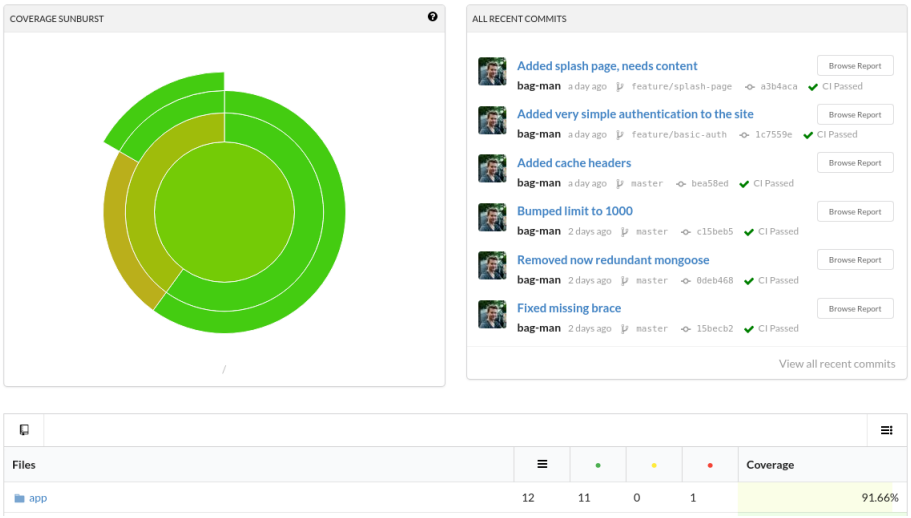

Figure 2.1: Codecov interactive coverage report

2.2.6 Deployment

If the tests pass on Travis CI, a Github hook will then activate the deployment. The project is setup to use Heroku,[37] and to build on successful CI results. This means that when code is pushed to the master branch, it will automatically be deployed to their service. This is excellent for development as it allows changes made to the system to be almost immediately reflected on the live website. The only drawback of Heroku is that this project is using their free offerings which only allow 500MB's of database storage, which means that anything larger than that will be truncated.

The benefits of this build process are clear, firstly code is automatically deployed, which saves time as deployments don't have to be done manually, as well as encouraging regular client feedback. However the main advantage is that before the deployment the code will have had the test suite ran on it twice, once on the developers local machine and then again on a brand new build on the CI service. This means that any mistakes are caught before they are put into production and the developer is notified of these issues so they can't be avoided.

2.3 Design

2.3.1 Choice of Database Technology

The first stage in designing this system was to decide upon what database technology would be used for the data store. In the Analysis section the merits and disadvantages of NoSQL and SQL solutions were highlighted in generic terms. For this project there is a focus on using modern web development technologies, and MongoDB (A NoSQL database solution) fits this well, as it stores data based on use case, which fits well with the MVC design pattern, as typically each Model in the design refers to a use case.

In this project there will only be one set of data to be stored, the list of genes and their various related data. From the users perspective they will only be interacting with this set of data, they will never be trying to combine a gene with another dataset for example. The only use case for this data then is to be displayed.

If this data was stored in a relational system like PostgreSQL, then there would need to be several tables for shared data, to store it in a normalised form. For example, there would be a contig table and each gene would store a link to the contig that it was from. This means that when a user requests to view a gene the database will have to perform a join between the gene table and the table storing the contig to retrieve all of the relevant information. This makes it harder to develop and maintain the database, as the relations between the datasets have to be considered throughout the project.

This would be advantageous if the data was going to be updated frequently, as a piece of data only needs to be updated once, as it is only stored once. However as this project is effectively read only, the data is never going to change, this isn't an issue with MongoDB.

This choice was also influenced by the choice of server side language, which for this project will be Node.js.

2.3.2 Choice of Server Side Technology

Node.js was chosen for several reasons, firstly as both server side and client side portions of the code are written in JavaScript, it is easier to develop, since you are working with one unified language.

It also works very well with MongoDB, as MongoDB represents it's data as JSON objects, which are native to JavaScript. This makes querying and operating on data returned via MongoDB much easier, as it is already in a format native to the language.

Node.js has a very good packaging system, called Node Package Manager (npm), the npm repository has access to the largest collection[1] of software packages, as well as being very easy to use and maintain thanks to the package.json file, which is automatically updated when new modules are used.

JavaScript has historically been looked down on due in part to it's syntax and type system, however with the latest stable specification[38] (ES6) of the language, a lot of these issues have been addressed.

As outlined in Ioannis K. Chaniotis et al's paper, 'A study on the viability of Node.js for

web development from the perspective of performance.', they state that 'Our study concludes that Node.js offers client-server development integration, aiding code reusability in web applications, and is the perfect tool for developing fast, scalable network applications.'[39]

In addition to this there is a lot of prior experience with the language and ecosystem, which reduces the barriers to development that learning a new language would create.

2.3.2.1 Nodestack

As well as Node.js the project will also be utilising a set of boilerplate code called Nodestack.[40] This code helps to setup a web service and provides many extra features to the basic Node.js stack.

Node.js / ES6

'Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem open source libraries in the world.'[41]

Since Node >4.0 it has supported the ES6 standard, which provides a lot of syntactic sugar, and makes the formatting of the code a lot easier to read and write. This is also supported for the client side code via the babel transpiler, which compiles ES6 code into the more widely supported ES2015 standard.

Webpack

Webpack is a module bundler that allows for easy packaging of client side javascript. It allows the client side javascript to be written in ES6, then when built it will transpile the ES6 code into more widely supported ES2015 code, minify it for efficient, and package it into one single file. This means that the code can be written in a nicely organised manner across multiple files in ES6, and then reduced to one small file for actual usage.

Mongoose

Mongoose is an Object Data Manager, it allows developers to define schemas and validation for their data objects as well as extensible models for those objects. This makes interactions with MongoDB a lot simpler, as well as easy enforcement of validation rules.

Pug

Pug is an HTML templating language that allows developers to write HTML with dynamic variables from the server side, and include some logic elements. The main benefit is that it has a much cleaner syntax than HTML so it is a lot easier to read and write.

Stylus

Stylus is an expressive CSS language that, like Pug and ES6, improves the syntax of CSS by minimising unnecessary elements. It has many other advantages over CSS, although for this project they aren't likely to be utilised as the front end design work of this project will be minimal.

Bootstrap

Bootstrap is a CSS and JS framework that is aimed at making responsive websites easier to write. With the addition of bootstrap to the project adapting the front end to work on mobile will be minimal. It also adds useful features for laying out the page and some attractive default CSS.

Using this boilerplate will save a huge amount of time in development as the basic framework of the web application is already laid out, so the development time can be focussed on developing software to meet the functional requirements rather than investing time in setting up a good web framework.

For more detail on how the boiler plate works, you can read the guide here: `http://blog.owen.cymru/nodejs-es6-boiler-plate/`

2.3.3 Database Import

To import the genenomic data into the database from the fasta file format, a script will be written that will use a npm module called fasta2json[42] to read the fasta files into JSON format.

From there it will use the alignment results generated by the Diamond script to look up the contig, coding sequence, and amino acid sequence (protein) for each of the alignment results.

It will also compare the hit ID with the mappings file that was produced to get the ID's for the equivalent UniProt and Candida Genome Database hits, if they are available.

That data will then be inserted into MongoDB, and an index built on the searchable fields. This index will allow for a very fast and efficient search for the key fields that will be searched for in the database; description, gene name and ID's.

Listing 2.1: The database schema that will represent a gene in the database.

```
{ id: Schema.ObjectId
, hitid: String
, species: String
, name: String
, cgdid: String
, uniprot: String
, contig:
  { head: String
  , seq: String
  }
, codingseq:
  { head: String
  , seq: String
  }
```

```
, protein:
  { head: String
  , seq: String
  , goids: [String]
  , desc: String
  }
, codingRange:
  { start: Number
  , end: Number
  , fail: Boolean
  }
}
```

2.3.4 Website Architecture

As the application only has one data object to model, the architecture isn't that complex. However it is still beneficial to follow good design principles which is why a traditional Model View Controller (MVC) design pattern[43] was followed.

The MVC design pattern allows for each data object that is being modelled to have three components, a model which defines the data's structure and it's functionality, a controller that is how the application interacts with that model, and finally a view, which is what is displayed to the user and how they interact with the model. For example a user may click on a gene, which sends a request to the controller, the controller then requests that gene from the model, which in turn updates the view, that the user can then see the gene that they had selected.

Abstracting the functionality of the project out into these three components allows for the development to be a lot easier to manage and work with, than having one monolithic class to handle the entire functionality of a gene.

There will only be two routes in the application, one to the home page, which will simply display a splash page with some information about the website and project on, and another route that handles searching and viewing genes, this will be done by having a URL parameter for the gene ID in the database.

If this parameter is present then the view to display a gene in full will be displayed to the user. If it isn't present then a search must have been performed and the query string in the URL will provide the options for the search. A search will return just a few key fields for each search result and then display the results in a list.

When viewing a gene, the coding sequence will have to be highlighted inside of the contig. To do this the coding range that was determined when the data was imported will be used to select the substring, and then some client side JavaScript will modify the HTML to wrap the coding sequence inside of a span tag.

As the coding sequences are stored in the database in only one compliment, there will be a reverse compliment button to reverse the compliment of the coding sequence. This is useful if the coding sequence is highlighted in the contig as the reverse compliment, it will allow the users to check that the data is correct.

2.3.5 User Interface

Before building the user interface for this project, server other similar sites were examined to see what information they were displaying and how it was laid out. The first was the Candida Genome Database[5] and it's locus page. It lists column headings on the left, and the the infromation on the right. This design is easy to read and doesn't clutter up the page too much.

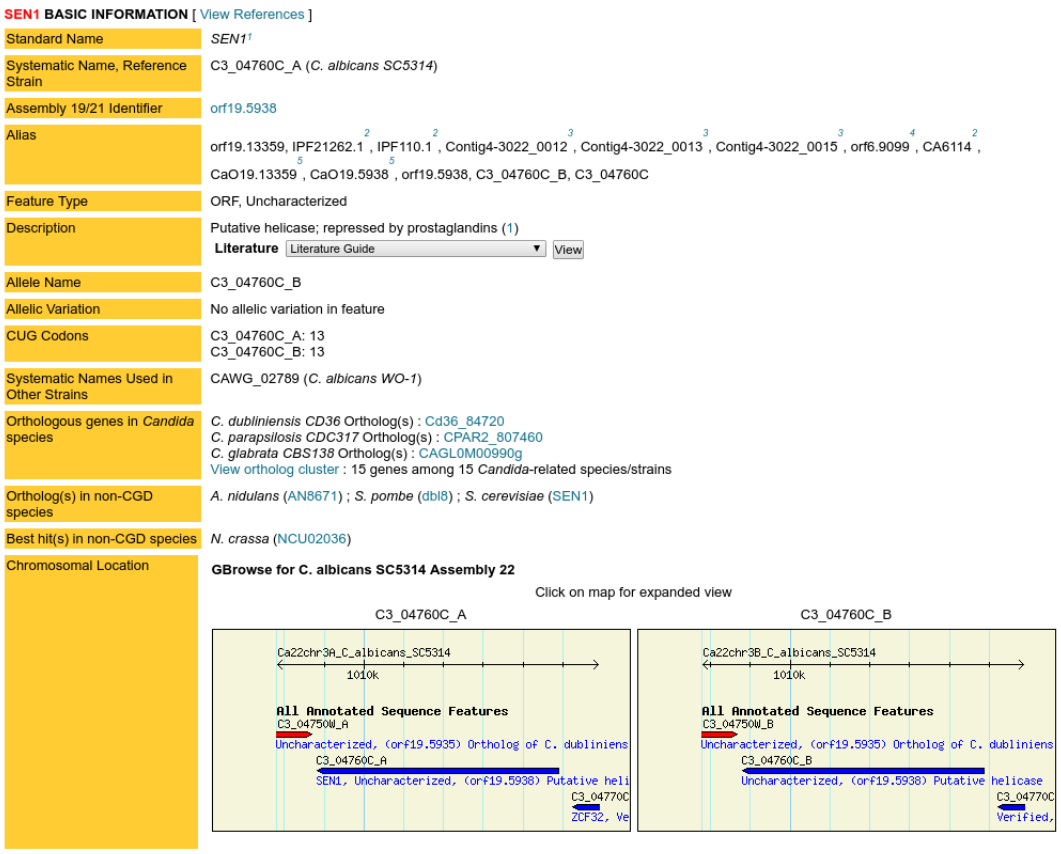

Figure 2.2: Candida Genome Databases's locus page

Next the Yeast Genome[44] project's locus page was inspected:

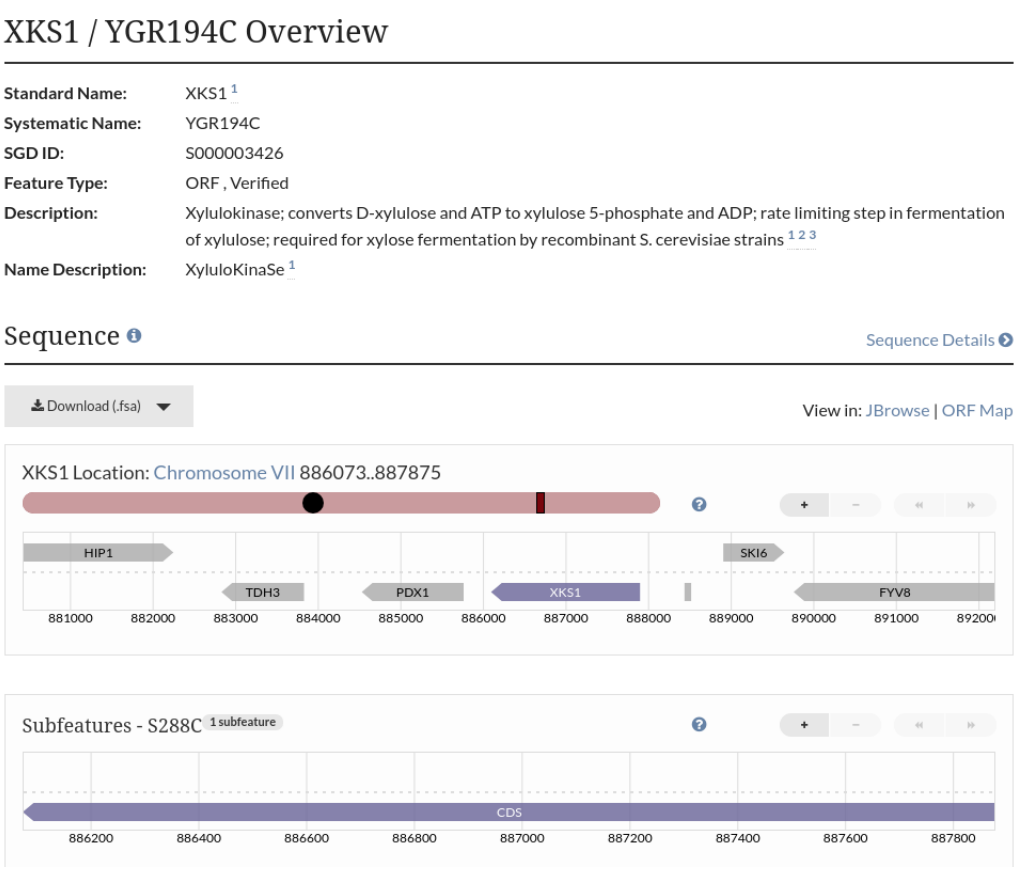

Figure 2.3: Yeast Genome Project's locus page

This page was very nicely formatted and laid out, it clearly had a lot of development, as it had integrations with two genome browsers as well as a large amount of extra information and graphical representations of the genes locations. Replicating this would be ideal, however it is too far out of the scope of this project, as many extra systems would have to be setup to get this data and make it as interactive.

Much like the rest of this project the UI was prototyped along with the understanding of the data at the time. This means that the UI was evolving with the data, and wasn't truly decided upon and polished until the data that was being displayed was known to be correct, and wouldn't change. Unfortunately this didn't happen until quite late in the project.

2.3.5.1 Final design

The focus for this design was to make it as simple to implement and use as possible, as this isn't a public facing service it doesn't need to be a cutting edge design aimed at grabbing users attention. Instead it just needs to be functional and represent the data clearly.

There are only two main pages on the website, a search view, and a locus page for specific genes. These share the same template, and adapt based on the data provided to them. The tasks for these pages are simply to display the data in the database, and provide a form to query it.

Tooltips were used to display information about each row. As the site is only going to be used by a couple of researchers there wasn't much benefit to creating a help section or a how to guide on how to use the site, as firstly it has been changing a lot during development and secondly it is quicker to just have a direct conversation with them about areas that they might not understand. However if in the future this were to become a public service, the home splash page would be updated to provide more information to external users.

The page was constructed with bootstraps[45] grid system, which allows the content to be dynamically repositioned to fit different size displays.

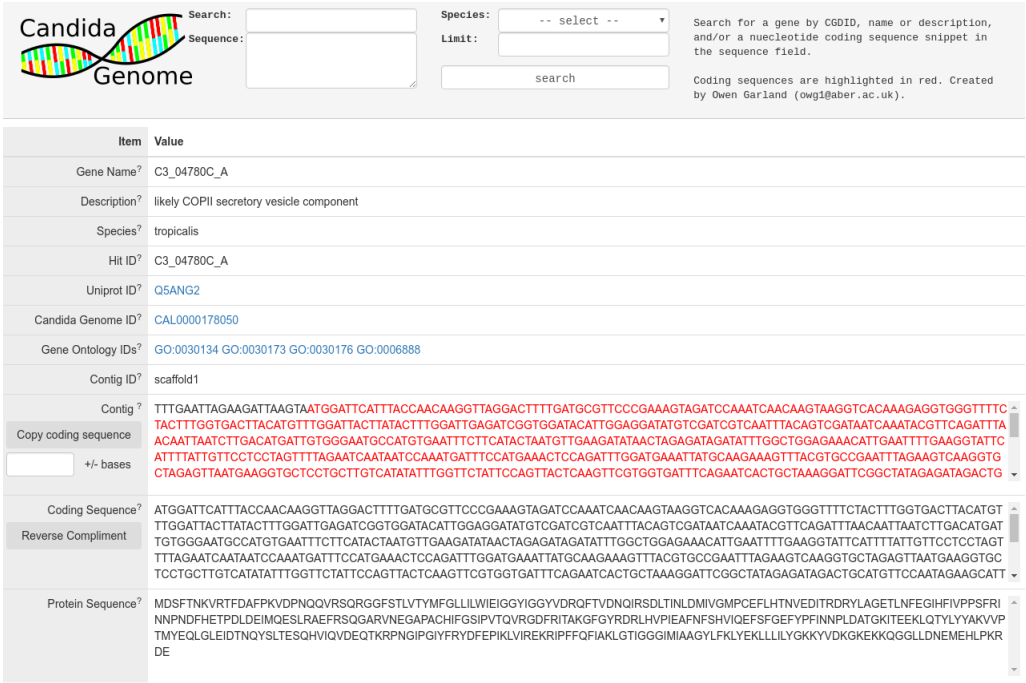

Figure 2.4: The final locus page for the project

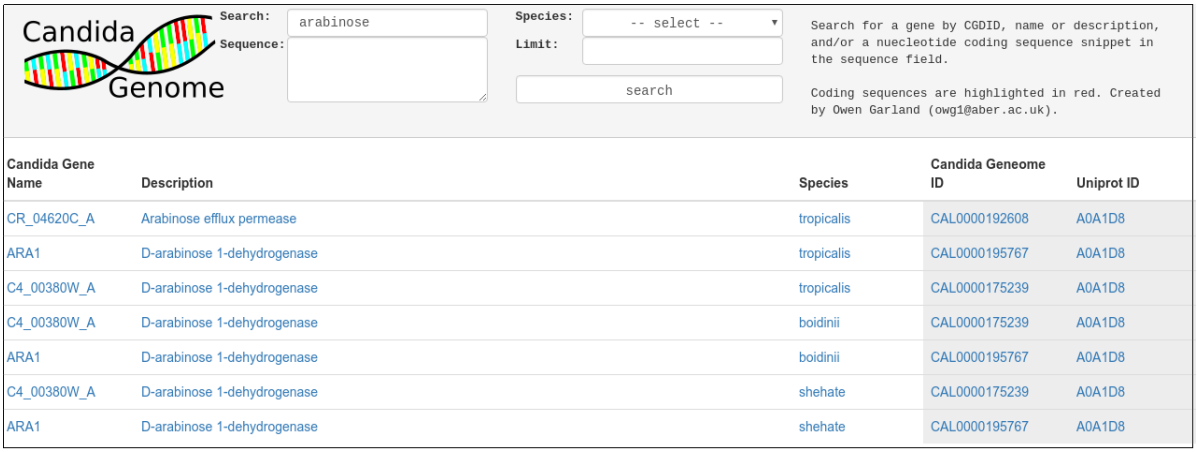

Figure 2.5: The final search page for the project

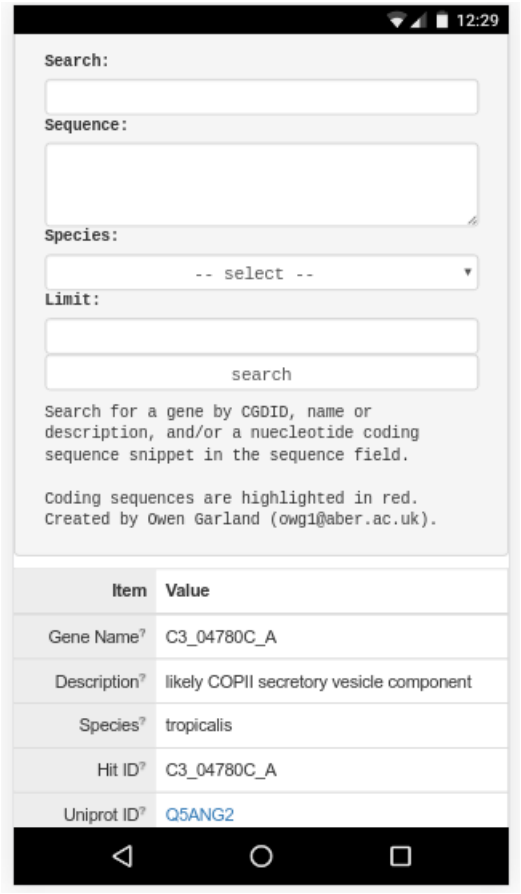

Figure 2.6: A locus page viewed on a mobile device

Figure 2.7: Tooltip on mouse hover describing what row means

2.4 Implementation

During the investigation into the data, many issues were encountered. Mainly due to a lack of understanding of what the data meant, and how it was produced. Initially the plan was to take the raw contigs for each species and use diamond to align them against the NCBI nr database. Then

from those results link the data back to the Candida Genome Database via the RefSeq ID's that the NCBI nr database uses. However it was later discovered that there was an annotated set of aligned data in the provided data, meaning that this step was no longer necessary.

A prototype was built around this alignment data, however it was then realised that the data for *C. boidinii* was not aligned against the NCBI nr database, but rather another unknown data set. This meant that to pursue this line of prototyping a replacement dataset for *C. boidinii* would have to be made, or alternatively reproduce all three datasets with another pipeline, as the results from the alignments weren't consistent.

The core problem was that it was unknown what tools had been used to produce this data as there was no accompanying documentation. It appeared that a proprietary tool blast2go[46] had been used to align and annotate the data. However after using a trial copy to try and replicate the results it was evident that this tool had not actually been used to create the alignments with the NCBI nr database.

Eventually it became clear that the alignment had been performed with the BLAST tool, on the universities high performance cluster, with an XML format that wasn't known to have been an option at the time. This meant that the data could actually be reproduced quite easily with Diamond, using the newly discovered XML output that BLAST alignments offer.

After these revelations, and another meeting with the researchers, we discovered in the original datasets that there were in fact already annotated protein sequence files and coding sequence files.

2.4.1 Collating the data, and linking it to Candida Genome Database

This meant that the data was already processed and largely annotated already. There was however one missing piece which was the link back to the Candida Genome Database (CGD), something that would be invaluable to the researchers who were already familiar with the CGD.

Because of this, thankfully the weeks of effort spent learning about alignments and annotations weren't wasted, as a new alignment was needed to be produced, using the coding sequences in the data set against the proteins found in *C. albicans*, the latter being provided by CGD.[23]

This data generated was then able to be used to link the coding sequences found in the data sets to the well documented genes in *C. albicans*. In addition to this the other annotations provided by the original dataset, such as Gene Ontology ID's, are able to be enhanced by using a mapping from the Candida Genome Database to get UniProt ID's which offer even more information on the known genes.

Now each species had, the raw assembled contigs, the coding sequences, the amino acid sequences, the annotated protein information, a link to the Candida Genome Database, and in some cases a link to the UniProt database.

2.4.2 Finding the coding sequence in the contig

The next key piece of information to recover that would be of great use to the researchers was the position of the gene (coding sequence) inside the raw assembled contig. With this information they would be able to easily find the sequence of bases that surround the gene.

To do this mock data was produced manually, that contained the correct results, and unit tests

made to check if a dummy function was returning the correct results as defined in the mock data. Then an algorithm was developed to find the position of a coding sequence inside a contig.

The initial algorithm was only finding around fifty percent of the genes in the dataset. After some discussions with the supervisor, it became apparent that the reason for this is that the coding sequences were stored in one commpliment, but the alignment results were finding genes that were the reverse compliment of that. This explains why around fifty percent of the genes weren't found.

Modifying the algorithm to search for the reverse compliment of the coding sequence if it wasn't found at first, resulted in every gene being detected. The algorithm, simply found the index of the first twelve bases of the coding sequence in the contig, and the last twelve bases of the coding sequence in the contig. If the start or end couldn't be found it was assumed that the gene was spread across two contigs and the start or end respectively were marked to indicate that the gene was split across two contigs.

```
1  module.exports = function findCodingRange (codingSeq, contig, reverse) {
1    let codingLength = codingSeq.length
2      , start = 0
3      , end = 0
4      , selector = 12
5      , fail = false
6
7    start = contig.indexOf(codingSeq.substring(0, selector))
8    end = contig.indexOf(codingSeq.substring(codingLength - selector, codingLength)) + selector
9
10   if (start < 0 && end <= selector) fail = true
11   if (start === -1) start = 0
12   if (end <= selector) end = contig.length
13
14   if (fail && !reverse) {
15     return findCodingRange(reverseCompliment(codingSeq), contig, true)
16   } else {
17     return { start, end, fail }
18   }
19
20   function reverseCompliment (sequence) {
21     let reverse = sequence.split('').reverse().join('')
22
23     return reverse.replace(/[ACTG]/g, (base) => {
24       return 'ACTG'.charAt('TGAC'.indexOf(base))
25     })
26   }
27 }
28
```

Figure 2.8: Algorithm to find coding sequence inside a contig.

2.4.2.1 Search functionality

Implementation of the search feature underwent several iterations, initially a search based on a regex match was implemented. This meant that each field was searched individually for any subset of the query string. This was very powerful as it would allow substrings to be found for each property of the searched for term.

An example of this would be if 'glucose-6' was searched in the description field, all documents in the database with that string in the description would be returned. This could then be combined with other fields to narrow down the query.

This implementation was done during the prototyping phase to get a very basic search working for testing purposes, the reason it wasn't going to remain in the application is that a project wide regex search is very inefficient as the search has to be matched against every value in the entire

database for every single query made. This would have resulted in search operations on teh website being too slow to be functional.

A better solution was to create a generic search field using MongoDB's textSearch[47] feature, that utilises pre-compiled indexes to lookup data in a much more advanced manner. Indexing is quite an intensive task, however for this application where the data is only being read from the service, there is only a need to generate an index once, when the data is being imported to the database, this means the negative impact of an index isn't really in effect, as the import stage is a one time event.

The fields indexed for the text search are the gene name, protein description, UniProt ID and Candida Genome Database ID. Weights are then applied to each field to determine their value when sorting the results of a search. For example a match with a UniProt ID is weighted higher than a protein description, allowing for results that have a match to a UniProt ID to be displayed higher in the search results, than those without.

With this system in place the search was functioning very well, however there was an issue, if a search returned a large amount of documents back, the sort was unable to be ran due to the default MongoDB sort buffer limit being too small to handle the number of results returned. To get around this issue the search function was re-written as an aggregate function. Aggregate[48] in MongoDB allows for a pipeline of steps to be applied to a query, as well as the option to use disk if memory limits are reached.

```
search (options) {
  let constraints =
      { _id: true
      , name: true
      , species: true
      , cgdid: true
      , uniprot: true
      , 'protein.desc': true
      , score: { $meta: 'textScore' }
      }
    , limit = options.limit < max ? parseInt(options.limit) : max
    , search = JSON.parse(JSON.stringify(options))
    , query = {}

  if (options['codingseq.seq']) {
    search['codingseq.seq'] = options['codingseq.seq']
    let seq = new RegExp(options['codingseq.seq'], 'i')
    query['codingseq.seq'] = seq
  }

  if (options.species) {
    query.species = options.species
  }

  if (!options.search) {
    delete constraints.score
  }

  let aggregateQuery = [ { $match: query }, { $project: constraints }, { $limit: limit } ]

  if (options.search) {
    query['$text'] = { '$search': options.search }
    let sort = { $sort: { score: { $meta: 'textScore' } } }
    aggregateQuery = [ { $match: query }, { $project: constraints }, sort, { $limit: limit } ]
  }

  HitModel.aggregate(aggregateQuery).allowDiskUse(true).exec((err, data) => {
    if (err) {
      this.render('Something went wrong', null, options, err)
    } else {
      if (!data.length) {
        this.render('Candida Search', null, search, 'No results.')
      } else {
        this.render('Candida Search', data, search, null)
      }
    }
  })
}
```

Figure 2.9: Function responsible for creating the aggregate search query.

This fixed the sorting bug, and ensures that all of the correct results for a query are returned, while still maintaining a high level of performance.

2.5 Testing

Initially it was planned to develop this application in a test driven manner, as it can help to produce high quality code with fewer defects that code produced in a traditional waterfall style.[49]

The difficulty with this approach was that when developing prototypes rapid development is key, as it allows you to try out many different solutions. If a strict TDD pattern was followed during this phase the prototyping would have been drastically slowed as the amount of development work would have doubled, due to the need to write robust unit tests prior to making the functionality. Unfortunately this wasn't a worthy endeavour as time was limited and developing tests for prototype code that may never end up in the final codebase was not going to be feasible.

Another difficulty in testing this project was that the majority of the projects logic is in the seed script that imports the data into the database. Testing this would be difficult because the only way of really checking the validity of what it was producing was to look at what was in the database

and to see if that made sense in a biological context.

Unfortunately it would be beyond the scope of this project to test the biological accuracy of the data that was imported. However I was able to manually verify that the data was correct at several stages, by aligning sequences from this project against other genomic databases such as the Candida Genome Database[5] and the Yeast Genome[44] project.

2.5.1 Automated Testing

For the area's of logic that were testable, a TDD design was followed, mainly for the selection of the coding sequence in the contig, as this was the only real area of logic in the project that had it's own custom algorithm to be tested. In addition to this the test suite that was being used for the project also checked the project for code formatting issues with the ESLint[32] tool. This will cause the tests to fail if potentially dangerous assertions are made in the code. For example if an equal to value (==) operator was used instead of an equal to value and type (===) operator it will throw an error highlighting the issue to the developer.

2.5.1.1 Unit Tests

Where possible test driven development was used to produce unit tests, these were mainly written for the logic that detects the coding sequence inside the contig. This is a crucial algorithm as the sites functionality depends on this data being accurate, so it was important that the algorithm be thoroughly tested. To do this several mock database records were created that had all the different possibilities that a gene could be found in, a normal hit, a reverse compliment hit, a missing hit and a hit that was split above and below the contig.

Tests were then written to compare the result of the finding function against the correct values stored in the mock data. The function was then able to be developed ensuring that the data it was returning was correct.

2.5.2 User Interface Testing

The user interface hasn't had automated tests written for it unfortunately, as there was only one HTML page and only two sets of data that were being returned it didn't feel particularly necessary to write tests to check that the data was coming through correctly.

That being said the UI has been manually tested on Chrome, Firefox and Internet Explorer to check for any functional issues. None were found, however it was noted that in Internet Explorer and Firefox there was some font rendering issues, but this isn't a concern as it doesn't impact the functionality of the site. I will be recommending that the site is used on Chrome though, as that it was it was developed on, so is the most thoroughly tested, with no apparent styling issues.

Once the site was at a stage where it could be demonstrated, the UI was shown to the researchers and they were asked to provide any feedback that might make the site easier to use for them. This feedback was mainly focussed on producing a splash page that would act as the home page for external users who were not familiar with the project, data or system. They requested that the University logo be included on the page as well as some text, which has yet to be provided.

2.5.3 Stress Testing

As this site is hosting commercially sensitive data, it won't be publicly accessible, this means that only the researchers who are working on this data will be accessing it. Because of this there isn't a need to perform any stress testing on the service, as the stack is more than capable of handling ¡ 10 users at a time, and isn't vulnerable to public attacks. To help combat any potential risks of stress on the system a Varnish Cache will be placed in front of the service to cache common requests, thus reducing the load on the server.

2.5.4 Integration Testing

As listed in the build process section 3.2.4, continuous integration services were used throughout this project, meaning that every time a new build was pushed to github, it had to pass tests on the CI server before being deployed to the staging server on Heroku. This has ensured that any modifications to the code base won't break the core functionality of the staging build.

2.5.5 User Testing

Once the site has been developed to a stage where the users can get a real sense of how it is looking and how it will function, they have been invited to suggest changes that need to be made, in line with the agile practices that this project has been developed with.

User testing was invaluable as the researchers were able to spot mistakes in the biological data that had previously gone unnoticed. An instance of this is when they noted that one of the UniProt IDs was linking to a gene found in the flu virus, something that shouldn't be present at all in the yeast genome. After investigating, it was found that some of the UniProt IDs had been truncated which caused this anomaly. Writing tests for these kinds of checks would have been nigh on impossible, thankfully user testing was able to pick up these kinds of issues.

Chapter 3

Deployment

3.1 User feedback

Once the application had been through several development iterations and was nearing completion, it was presented to the researchers for signing off, discussions about deployment and maintenance provisions. Only one major change was requested, to have the site password protected, as they wanted to restrict access to only trusted researchers working on the project, until the research was published.

Ideally a password system would allow for multiple users with different passwords, with some levels of account controls by an administrator. This would be quite an extensive bit of work to undergo as a new users model would have to be created and the user interface would have to be updated to reflect all of the these new controls. Due to the time restrictions on this project, this was not a feasible request to complete to a satisfactory level.

An alternative, but far inferior solution was instead implemented, simply adding a Basic HTTP Auth header to the application. This involved only adding a module (basic-auth-connect) to the application, and one line of code to enable it in the middleware of the app. This does provide a password protection to the site to keep curious eyes out of the project, however it is very insecure as the password is stored on the server in plain text, as well as transmitted unencrypted over HTTP, since HTTPS hasn't been implemented for this service.

3.2 Deployment

To deploy this project, an Ubuntu 16.04 Virtual Machine (VM) with three cores, 10GB of storage and 4GB of RAM was provisioned by the university. A user 'candida' was then made for the project to be ran under and the release branch of the project cloned down to the machine from the Github repository. Node.js was then installed from the Ubuntu repositories, so that the system had access to the Node Package Manager (npm). This was then used to install nave,[50] m,[51] and yarn.[52] Three tools that help with the management of packages and versioning for Node.js, MongoDB and npm packages.

Nave was then used to install Node.js version 6.9.0, which is what this application was developed for, and m was used to install MongoDB 3.4.4. The dependencies listed in the pack-

age.json file were then installed to the node_modules folder with yarn, the project could then be build with the build script defined in the package.json. This process is documented in the projects README.md file.

To generate the data for the database and import it to MongoDB, the seed script defined in the package.json was run, this runs the importation script using the data that is included in the repository to populate the database.

To ensure the stability of the service, it was put behind a Varnish[53] cache. Doing this enabled caching on all of the responses from the web server to client requests, meaning that if a request has already had a response processed for it, then a cached version will be served from Varnish rather than adding load to the Node.js application. This speeds up the sites response time as data can be immediately returned, once it has been processed already.

Unfortunately a side effect of the last minute need to include basic HTTP auth, is that the benefits from puttin ghte service behind Varnish are nullified, as Varnish won't cache content behind a basic HTTP auth. This another good reason to find a better solution to the password protection in the future.

A systemd[54] service file was then written and installed on the server for the Node.js process, this enables the application to be controlled from systemd. This is done so that the application can be started at boot time, as well as monitored, restarted and stopped from the systemd interface; which is how all the other services on the VM are controlled.

One difficulty faced during this deployment was the versioning on MongoDB, provided by the OS on the VM. The application had been developed on Arch Linux which provides a recent version (3.4) of MongoDB in it's package manager, however on the VM which was running Ubuntu, the package was for version 2.6, that doesn't support the new wiredTiger[55] storage engine.

The difference in versions was noted initially, and corrected by installing version 3.4. However when data was entered into the database, it was taking up around three times as much room, filling up the VM's disk space.

To correct this a manual process of updating the storage engine and database directory in the MongoDB configuration was necessary. This enabled the data to be imported correctly, taking up the expected amount of storage.

With the deployment complete, the README.md file in the project was updated to reflect the steps necessary to replicate a full deployment, so that other developers could replicate the production environment.

Chapter 4

Evaluation

The main challenge of the project was understanding the data from a different domain to computer science, and how it needed to be interpreted, to be presented in an accessible form. This meant that the requirements only really became clear very late in the project, as the requirements were constantly adapting with my understanding of the data.

Once the final requirements were decided upon, development went very smoothly. The end result is functional, and meets all of the achievable criteria, so I believe the design decisions were correct, and the implementation done in a clean and efficient manner. The codebase produced is only around 700 lines for the importation script and website combined, this isn't a terribly useful metric, but it does show that the code is at least concise.

The use of Node.js and MongoDB does set this project apart from older similar projects, that use Perl and PostgreSQL. One of the aims of this project was to demonstrate that the newer technologies can provide a tangible benefit over the older systems, and I think that the simplicity of the system, compared to it's levels of functionality do show that there is something to be gained from using a more modern stack.

Once the website was deployed to the university servers, the researchers were able to use it for a more extended period, and appeared to be very happy with what was produced. They did make suggestions of additional features that could have been included, but these weren't discussed during the project so weren't really in scope. In the future though it would definitely be an interesting project to expand the system to include some of these features such as a comparison tool between genes, and a users system that would allow for administrators to upload new datasets.

The area that really could have been improved upon was the amount of time it took to get to understand the data. If the data had been better documented, less time would have had to be spent trying to understand it and there could have been more features developed that would have helped the researchers a lot more than what I was able to produce. Additionally there would have been more time to work on improving the test coverage of the application to improve it's reliability.

The application was built with very robust software engineering principles, and the code is a testament to how cleanly it was developed, there are very few smells[56] throughout the code. This is mostly thanks to the discipline enforced by having linters and automated testing. The only area that falls down in this regard is the amount of unit tests. As the project stands there are only a handful of unit tests to check the finding of coding sequences. Ideally it would be good to expand the testing to cover the Pug templates, the controller logic, and the search functionality.

Unfortunately by the time the data was understood enough to develop the application fully, there was not enough time to test these features and write up this report.

Future work on the project would focus on improving the test coverage of the unit tests, and then working on implementing a proper users system that would allow the website to be password protected on a user basis. This would fix the issue with the basic HTTP auth, enabling the site to work with the Varnish cache as well as be made accessible to the public with ease in the future. It is somewhat outside of the scope for this project, however it would be interesting to develop a tool that processes data from other analysis tools into the format ready for ingestion to the database. If such a tool were to be created this would allow this web service to become an open source option for making annotated genomes more accessible.

I will continue to support this project after it's completion, and hopefully continue working with the researchers to develop new features and better the understanding of the data that has been produced, as well as potentially adding new datasets to the project for them.

Appendix A

Third-Party Code and Libraries

1.1 Nodestack

`https://github.com/bag-man/nodestack`

This project uses boilerplate code developed by myself for previous projects. The code sets up a basic Node JS project, with access to to third party services. During the development of this project some features were merged back into the boilerplate, such as the mongoose integration. This boilerplate makes use of several third party libraries, which are all listed in the package.json. Some extra modules have also been added specific to this project such as fasta2json and the clipboard module.

1.2 Externally hosted libraries

JQuery

`https://github.com/jquery/jquery` - MIT

JQuery is a client side javascript library that makes navigating and interacting with the HTML DOM far easier.

Bootstrap

`https://github.com/twbs/bootstrap` - MIT

Bootstrap is a library for styling webpages and making them easily responsive to different screen sizes.

1.3 package.json

Below is the modules listed in the package.json for the project, which can be found with the source code, all third party software is listed here along with it's version. This is how node modules are

packaged, this automates the management of all modules, and locks the versions. For this project I was using yarn[52] to manage the packages rather than the default node package manager (npm).[?] Yarn has a couple of advantages, mainly that it is a lot faster than npm3 and that it automatically creates a lock file for modules and their dependencies.

1.3.1 Development Dependencies

This section of third party modules are not installed on the production builds of the application, they are only for use when developing the application, they don't change the functionality of it at all, just make it easier to work on.

codecov

`https://www.npmjs.com/package/codecov` @ 1.0.1 - MIT

This provides integration with codecov's services, to provide interactive test coverage information.

basic-auth-connect

`https://www.npmjs.com/package/basic-auth-connect` @ 1.0.0 - MIT

This provides a basic HTTP auth middleware for express.

eslint

`https://www.npmjs.com/package/eslint` @ 2.3.0 - MIT

This is the linting engine that is used to check the source code for mistakes.

eslint-config-clock

`https://www.npmjs.com/package/eslint-config-clock` @ 1.2.0 - ISC

This is a set of configurations for eslint that describe the code formatting that is preferred for this project.

eslint-config-standard

`https://www.npmjs.com/package/eslint-config-standard` @ 5.1.0 - MIT

The clock config is built on top of this standard configuration

eslint-plugin-promise

`https://www.npmjs.com/package/eslint-plugin-promise` @ 3.3.0 - ISC

Eslint didn't support promises natively at the time of writing, so this was used to detect promises in the code.

eslint-plugin-standard

`https://www.npmjs.com/package/eslint-plugin-standard` @ 1.3.1 - MIT

Add's a few extra rules to eslints configuration options.

husky

`https://www.npmjs.com/package/husky` @ 0.13.2 - MIT

Simply runs the test suite before code is pushed to remote repositories.

istanbul

`https://www.npmjs.com/package/istanbul` @ 1.0.0-alpha.2 - BSD-3-Clause

Provides coverage information at the end of the test suite, and for codecov's usage.

mocha

`https://www.npmjs.com/package/mocha` @ 3.1.2 - MIT

Testing framework for javascript.

nodemon

`https://www.npmjs.com/package/nodemon` @ 1.11.0 - MIT

Monitors source files for changes, and relaunches the application when files are changed.

1.3.2 Build Dependencies

These third party modules are the frameworks libraries and modules that are actually used to by the application to function.

babel

`https://www.npmjs.com/package/babel` @ 7.5.2 - MIT

Transpiler for javascript. Used for converting ES6 code into ES5 for browser compatibility.

babel-core

`https://www.npmjs.com/package/babel-core` @ 6.18.0 - MIT

Core configurations for babel.

babel-loader

`https://www.npmjs.com/package/babel-loader` @ 6.2.5 - MIT

Allows for transpiling to be done from webpack build manager.

babel-preset-es2015

`https://www.npmjs.com/package/babel-preset-es2015` @ 6.18.0 - MIT

The standard ES5 configuration.

clipboard

`https://www.npmjs.com/package/clipboard` @ 1.6.1 - MIT

Clipboard module used for copying text to a users system clipboard from the browser.

express

`https://www.npmjs.com/package/express` @ 4.14.0 - MIT

The web framework that powers the node application.

fasta2json

`https://www.npmjs.com/package/fasta2json` @ 0.1.1 - MIT

Module that reads fasta files into JSON objects.

mongoose

`https://www.npmjs.com/package/mongoose` @ 4.8.6 - MIT

MongoDB object modelling framework.

morgan

`https://www.npmjs.com/package/morgan` @ 1.7.0 - MIT

Cleaner and clearer logging output.

pug

`https://www.npmjs.com/package/pug` @ 2.0.0-beta11 - MIT

HTML templating language.

stylus

`https://www.npmjs.com/package/stylus` @ 0.54.5 - MIT

CSS templating language.

webpack

`https://www.npmjs.com/package/webpack` @ 2.2.1 - MIT

Javascript build manager.

Appendix B

Ethics Submission

Ethics submission ID: 6907

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
owg1@aber.ac.uk

Full Name
Owen Garland

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
Building an Online Resource for Candida Tropicalis

Proposed Start Date
01/02/2017

Proposed Completion Date
08/05/2017

Are you conducting a quantitative or qualitative research project?
Quantitative

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Annotating genomic data and making the results available to researchers via a web interface.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Yes

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

Declaration of originality

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 .....

Date .....

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name .....

Date .....

Acknowledgements

I am grateful to...

I’d like to thank...

Annotated Bibliography

[1] Unkown, "Data displaying number of packages available to different platforms," https://modulecounts.com, 2017, accessed April 29th 2017.

An interactive website that displays data on the number of modules available for various package managers. The data is collected via night cron job, and isn't definitive, but should provide a good indicator.

[2] B. Wales, "Beacon wales about page," http://beaconwales.org/en/what-is-beacon/, 2017, accessed May 2nd 2017.

The about page for the BEACON Wales project.

[3] S. R. Ravella, J. Gallagher, S. Fish, and R. S. Prakasham, "Overview on commercial production of xylitol, economic analysis and market trends," in *d-Xylitol*. Springer, 2012, pp. 291–306.

Explanation of what xylitol is, why it is used and how it is produced.

[4] A. Sboner, X. J. Mu, D. Greenbaum, R. K. Auerbach, and M. B. Gerstein, "The real cost of sequencing: higher than you think!" *Genome biology*, vol. 12, no. 8, p. 125, 2011.

A study showing that the expensive part of genome sequencing and annotation is now actually the annotation.

[5] M. B. Arnaud, M. C. Costanzo, M. S. Skrzypek, G. Binkley, C. Lane, S. R. Miyasato, and G. Sherlock, "The candida genome database (cgd), a community resource for candida albicans gene and protein information," *Nucleic acids research*, vol. 33, no. suppl 1, pp. D358–D363, 2005.

The Candida Genome Database is a web resource that has annotated genome data for serveral strands of *Candida* yeasts. It is the main reference resource for the researchers at Aberystwyth

[6] P. Zhou, D. Emmert, and P. Zhang, "Using chado to store genome annotation data," *Current Protocols in Bioinformatics*, pp. 9–6, 2006.

CHADO is one of the most mature genomic database implemetations. It is written in Perl and SQL.

[7] L. S. Gramates, S. J. Marygold, G. dos Santos, J.-M. Urbano, G. Antonazzo, B. B. Matthews, A. J. Rey, C. J. Tabone, M. A. Crosby, D. B. Emmert, *et al.*, "Flybase at 25: looking to the future," *Nucleic Acids Research*, p. gkw1016, 2016.

Flybase is the project that created the CHADO database.

[8] B. PostgreSQL, "Postgresql," *Web resource: http://www. PostgreSQL. org/about*, 2010.

Home page for the PostgreSQL database project.

[9] L. Wall *et al.*, "The perl programming language," 1994.

The initial paper written to accompany the creation of the Perl programming language.

[10] Various, "Usage statistics and market share of perl for websites," https://w3techs.com/technologies/details/pl-perl/all/all, 2017, accessed April 29th 2017.

A webservice that montiors the market share of various technologies, this page shows Perls market share among serverside technologies.

[11] M. F. Sanner *et al.*, "Python: a programming language for software integration and development," *J Mol Graph Model*, vol. 17, no. 1, pp. 57–61, 1999.

The initial paper written to accompany the creation of the Python programming language.

[12] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

Describing the use of NodeJS as a serverise language.

[13] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic acids research*, vol. 25, no. 17, pp. 3389–3402, 1997.

The original protein alignment tool, that is considered an industry standard.

[14] B. Buchfink, C. Xie, and D. H. Huson, "Fast and sensitive protein alignment using diamond," *Nature methods*, vol. 12, no. 1, pp. 59–60, 2015.

Detailing how the Diamond alignment tool works

[15] S. T. Sherry, M.-H. Ward, M. Kholodov, J. Baker, L. Phan, E. M. Smigielski, and K. Sirotkin, "dbsnp: the ncbi database of genetic variation," *Nucleic acids research*, vol. 29, no. 1, pp. 308–311, 2001.

The NCBI database is the largest collection of annotated genes.

[16] K. D. Pruitt, T. Tatusova, and D. R. Maglott, "Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins," *Nucleic acids research*, vol. 35, no. suppl 1, pp. D61–D65, 2007.

Detailing the NCBI non-redundant database of proteins.

[17] P. Klus, S. Lam, D. Lyberg, M. S. Cheung, G. Pullan, I. McFarlane, G. S. Yeo, and B. Y. Lam, "Barracuda-a fast short read sequence aligner using graphics processing units," *BMC research notes*, vol. 5, no. 1, p. 27, 2012.

Information on how the BarraCUDA GPU alignment tool was created.

[18] M. Yano, H. Mori, Y. Akiyama, T. Yamada, and K. Kurokawa, "Clast: Cuda implemented large-scale alignment search tool," *BMC bioinformatics*, vol. 15, no. 1, p. 406, 2014.

Details on the CLAST GPU alignment tool.

[19] Y. Liu, D. L. Maskell, and B. Schmidt, "Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units," *BMC research notes*, vol. 2, no. 1, p. 73, 2009.

Details on the CUDASW++ GPU alignment tool.

[20] M. J. Donlin, "Using the generic genome browser (gbrowse)," *Current Protocols in Bioinformatics*, pp. 9–9, 2009.

Publication of GBrowse, one of the most widely used Genome viewers, still written in Perl

[21] M. E. Skinner, A. V. Uzilov, L. D. Stein, C. J. Mungall, and I. H. Holmes, "Jbrowse: a next-generation genome browser," *Genome research*, vol. 19, no. 9, pp. 1630–1638, 2009.

Publication of JBrowse, an improved genome viewer, but still with a backend written in Perl.

[22] R. N. Smith, J. Aleksic, D. Butano, A. Carr, S. Contrino, F. Hu, M. Lyne, R. Lyne, A. Kalderimis, K. Rutherford, *et al.*, "Intermine: a flexible data warehouse system for the integration and analysis of heterogeneous biological data," *Bioinformatics*, vol. 28, no. 23, pp. 3163–3165, 2012.

A competing and more modern genomics database to CHADO, still written in Perl.

[23] C. G. Database, "Annotated proteins found in *Candida albicans*," http://www.candidagenome.org/download/sequence/C_albicans_SC5314/Assembly22/current/C_albicans_SC5314_A22_current_default_protein.fasta.gz, 2017, accessed April 29th 2017.

The dataset of *C. albicans* proteins that were used to align with the coding sequences that were provided to this project.

[24] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, *et al.*, "Gene ontology: tool for the unification of biology," *Nature genetics*, vol. 25, no. 1, pp. 25–29, 2000.

The Gene Ontology website, that contains data on various chemical pathways that are found in known genes.

[25] C. G. Database, "All annotations and id's for every protein in cgd," http://www.candidagenome.org/download/go/gene_association.cgd.gz, 2017, accessed April 29th 2017.

Dataset containing reference ID's to all of the genes in the Candida Genome Database.

[26] U. Consortium *et al.*, "Reorganizing the protein space at the universal protein resource (uniprot)," *Nucleic acids research*, p. gkr981, 2011.

Explanation of the Uniprot database.

[27] O. Garland, "Vim configuration," https://github.com/bag-man/dotfiles/blob/master/vimrc, 2017, accessed April 29th 2017.

My personal editor configuration, used while developing this project.

[28] P. Serby, "Clock ltd. eslint configuration," https://github.com/clocklimited/eslint-config-clock, 2017, accessed May 1st 2017.

An ESLint configuration that uses comma first notation, and was used in this project.

[29] I. Z. Schlueter, "Demonstraiton of the advantages of comma first," https://gist.github.com/isaacs/357981, 2017, accessed May 1st 2017.

An explanation of why comma first notation is advantageous for software development.

[30] Various, "Github remote git repository," https://github.com/about, 2017, accessed April 29th 2017.

About page for github.com

[31] ——, "Husky," https://github.com/typicode/husky, 2017.

A node module to enforce the running of tests before code can be pushed to remote hosts.

[32] ——, "Eslint," https://github.com/eslint/eslint, 2017.

A node module + CLI tool to check for code style issues, configurable via JSON files.

[33] ——, "Mocha," https://github.com/mochajs/mocha, 2017.

NodeJS testing framework.

[34] ——, "Istanbul," https://github.com/gotwarlost/istanbul, 2017.

A node module to report test coverage.

[35] ——, "Travis ci documentation," https://docs.travis-ci.com/, 2017, accessed April 29th 2017.

Website for the documentation of Travis CI, a continuous integration platform.

[36] ——, "Codecov documentation," https://docs.codecov.io/, 2017, accessed April 29th 2017.

Website for the documentation of Codecov, an interactive test coverage data manager.

[37] N. Middleton and R. Schneeman, *Heroku: Up and Running*. " O'Reilly Media, Inc.", 2013.

Introduction to Heroku, and detailed guide on how to use it.

[38] E. Ecmascript, "Language specification," 2015.

The ES6 language specification

[39] I. K. Chaniotis, K.-I. D. Kyriakou, and N. D. Tselikas, "Is node.js a viable option for building modern web applications? a performance evaluation study," *Computing*, vol. 97, no. 10, pp. 1023–1044, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00607-014-0394-9

A study on the viability of NodeJS for web development from the perspective of performance.

[40] O. Garland, "Nodestack," https://github.com/bag-man/nodestack, 2017.

The boilerplate code for this project, written by myself. More information can be found at http://blog.owen.cymru/nodejs-es6-boiler-plate/.

[41] Various, "Nodejs homepage," https://nodejs.org/, 2017, accessed April 29th 2017.

The NodeJS home page.

[42] J. Juanes, "fasta2json," https://github.com/jmjuanes/fasta2json, 2017.

The node module that was used to read the FASTA files into JSON objects for use in the scripts.

[43] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*. IEEE, 2001, pp. 118–127.

Explaination of the MVC design pattern, and how it can be used for web development

[44] J. M. Cherry, E. L. Hong, C. Amundsen, R. Balakrishnan, G. Binkley, E. T. Chan, K. R. Christie, M. C. Costanzo, S. S. Dwight, S. R. Engel, *et al.*, "Saccharomyces genome database: the genomics resource of budding yeast," *Nucleic acids research*, p. gkr1029, 2011.

The yeastgenome.org project, has the most polished website, and is an excellent example of how genomic data can be best represented.

[45] Various, "Bootstrap," https://github.com/twbs/bootstrap, 2017.

A styling framework of CSS and JS files that allows for easy development of responsive web applications.

[46] A. Conesa, S. Götz, J. M. García-Gómez, J. Terol, M. Talón, and M. Robles, "Blast2go: a universal tool for annotation, visualization and analysis in functional genomics research," *Bioinformatics*, vol. 21, no. 18, pp. 3674–3676, 2005.

A proprietary genome annotation toolkit, that was used in the production of this projects data.

[47] K. Chodorow, *MongoDB: the definitive guide*, 2013, pp. 115–119.

A chapter explaining how the MongoDB textSearch function works.

[48] ——, *MongoDB: the definitive guide*, 2013, pp. 127–140.

A chapter explaining how the MongoDB aggregate function works.

[49] L. Williams, E. M. Maximilien, and M. Vouk, "Test-driven development as a defect-reduction practice," in *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*. IEEE, 2003, pp. 34–45.

[50] Various, "nave," https://github.com/isaacs/nave, 2017.

A Node.js version manager.

[51] ——, "m," https://github.com/aheckmann/m, 2017.

A MongoDB version manager.

[52] ——, "yarn," https://github.com/yarnpkg/yarn, 2017.

Fast, reliable, and secure dependency management.

[53] ——, "Varnish cache server," https://github.com/varnishcache/varnish-cache, 2017, accessed May 3rd 2017.

Varnish cache Github repository.

[54] ——, "Systemd system and service manager," https://github.com/systemd/systemd, 2017, accessed May 3rd 2017.

The systemd Github page.

[55] R. R. Gundreddy, "Performance evaluation of mmapv1 and wiredtiger storage engines in mongodb: An experiment," 2017.

A look at the difference in performance from MongoDB's storage engines.

[56] M. Mantyla, J. Vanhanen, and C. Lassenius, "A taxonomy and an initial empirical study of bad smells in code," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*. IEEE, 2003, pp. 381–384.

Detailing what bad code smells are, and how they effect software projects.

[57] Various, "Biostars - a discussion on splitting a fasta file, on the biostars forum," https://www.biostars.org/p/13270/, 2012, accessed April 2017.

A forum discussion on how best to split FASTA files into even chunks.

[58] G. Gremme, S. Steinbiss, and S. Kurtz, "Genometools: a comprehensive software library for efficient processing of structured genome annotations," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 3, pp. 645–656, 2013.

Documenting a suite of tools called "Genometools", that can among other things, split FASTA files.