# 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)

April 23, 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

# 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...

# Abstract

Include an abstract for your project. This should be no more than 300 words.

# CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# Chapter 1

# Background & Objectives

## 1.1 Background

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 had their genomes sequenced by Illumina sequencing machines. From that point the sequences of DNA that have been read are then assembled into contigs (contiguous DNA fragments). This process produces an accurate representation of the species DNA, which then can then be aligned to other better understood DNA and annotated as such.

Arabinose and Xylose are five carbon sugars ubiquitously found in plants such as grass, which can be used as a feedstock for industrial biotechnology. *Candida tropicalis* is able to convert arabinose & xylose into arabitol and xylitol respectively. Xylitol is a commercially valuable anti-bacterial foodgrade sugar use in the manufacture of chewing gum. However arabitol and xylitol are stereoisomers and cannot be easily separated. *Candida boidinii* cannot metabolise arabinose, for an unknown reason, but does metabolise Xylose but at a much slower rate which is commercially in-viable.

Understanding at a genomic level why *Candida boidinii* is unable to utilise arabinose and genetically modifying *Candida tropicalis* to have the same phenotype, may enable *Candida topicalis* to exclusively produce xylitol and not arabitol. This would mean a cheap and easy to produce 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 diabetics.

## 1.2 Existing Databases

Before undertaking this project –¿I¡– had some experience working with genomic data, and had been sitting in on lectures for a functional genomics module. This helped me get a good understanding of the basics of genetics and get a foothold on the terminology that is used in the world of bioinformatics. As the core of the project focuses on having the data ingested into a database, the first action was to investigate the current database solutions that had been developed for genomic information.

The most well established was CHADO**??** which was initially developed back in 2005 [**?**]

for the FlyBase [?] project. It has since then grown to accommodate genomic information for eukaryotes, plants, and other complex multicell animals. It uses PostgreSQL [?] to store it's data and Perl [?] 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 you have to consider a huge amount of relations between your tables.

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 where failing their tests and not installing. This meant several modules had to be manually fixed just to get the application to install correctly. If it takes an experienced programmer and linux admin several hours to follow the default installation instructions, debugging issues at several steps, it isn't going to be appropriate to impose such a system on potentially less technically minded Biologists. This combined with the fact that Perl has now been superseded by more modern scripting languages such as Python [?] and JavaScript [?], thanks to their superior package management, eco-systems and syntax, it was clear that a more modern approach would be more sustainable and maintainable for this project and for the future.

## 1.3 Alignment Tools

As the source of the data was uncertain, it was decided to investigate performing alignments in the event that more data, or different data was needed for the project than what was provided. The original alignment tool BLAST?? 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 proteins and annotate them accordingly. These annotations aren't proof that the gene codes for that exact protein, but it is a strong indicator that it does. 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 test genes that are predicted via an alignment. An alignment is a computationally intensive task there have been many efforts to improve the efficiency of the algorithm and make the most of developments made in computing since the 90's. 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 on a modern CPU.

Initially I tested Diamond [?] and was able to blast the data sets against the NCBI [?] non-redundant protein database, in about an hour and a quarter for each species, this was only using the CPU and may have been bottlenecked by the mechanical drives being used as storage. The hardware being used for this was an Intel i7-6700k @ 4.6Ghz, 16GB of RAM, a 7200rpm Hard Disk Drive, and a Nvidia GTX 980. As there was a high performace graphics card available it would be advantageous to make use 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 [?], 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's in size, and the system that was being used only had 16GB's of RAM available.

Looking for other options I encountered CLAST [?], 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 to try was Cudasw++ [?] 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.

## 1.4 Similar systems

From studying bioinformatics and researching the most common methods of manipulating and analysing the data gathered, a trend was found that indicated that unlike computer science, where tools are highly developed and have evolved to a very stable state where the best tools are known and highly specialised to suit their purpose, in bioinformatics a combination of factors including the relative youth of the field and the large number of competing projects that don't collaborate; there have been a wide array of tools created, often by biologists with no software engineering experience, none of which have been adopted and honed as the de facto standard. This means that for every task in the bioinformatics space there is often many different solutions offered, to what can be a very simple problem.

Gbrowse?

My interest in this project stems from my interest in genetics, something that has always fascinated me since I learned about how we evolved into being. As a computer scientist I relish the chance to apply the knowledge of my domain, to a real life application that can have a measurable positive impact on the world. Hopefully I will be able to use this experience to further my career into bioinformatics as well as helping contribute to the research being done.

## 1.5 Analysis

From researching the current solutions to the problem of annotating, storing and presenting genetic information, it was clear that a modern solution (describe that) wasn't currently in the public open source domain. Of the open source projects that were currently in use the majority appeared to be very old and monolithic, commonly written in Perl. Expand on all this etc.

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 have been to use one of the existing databases such as Intermine or CHADO, and then use a web front end compatible with those databases such as GBrowse or Jbrowse. This would like require slight changes to how the data was processed before going into these established systems, however it would mean that the data would be in a format that is well understood by bioinformaticians.

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, the hard bit, it just needs to be made into a database friendly format and then ingested to the database system. From there building a simple web front end to make the database accessible is a very simple task. There isn't any need to manipulate or modify the data, simply create and read it. Because of this it seemed that the current systems were greatly over complicating matters.

### 1.5.1 SQL vs 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, not duplicated
- 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 indivdual 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 itis 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 less efficient

## 1.6 Research Method and Software Process

# Chapter 2

# Experiment Methods

## 2.1 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 for me to ascertain what bits of data were actually relevant to the project, and what other data needed producing. For each species there were three files that I would need to use. What was provided:

- Raw assembled contigs of DNA, these would be used to get a genes position in the genome, and it's surrounding bases.
- Protein sequences annotated with Gene Ontology ID's, these had been extracted from an NCBI nr blast
- Coding sequences, these are the nucleotide sequences that coded for the proteins. Uncertain as to how they were produced.

Linking this data to the Candida Genome Database was a key bit of information that was missing. To do this the annotated proteins 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.

There was then several sets of data required to map these proteins to the Candida Genome Database, that would allow the researchers to easily compare the genes to well annotated species. First was a file of GO annotations for all the proteins stored in the Candida Genome Database. [**?**] 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.

In addition to this was a mapping file that mapped Candida Genome Database ID's to Uniprot ID's. With a small JavaScript script and some manual cleaning with a text editor, I was able to produce a complete mapping file in JSON, ready to be read in by the importation script.

# Chapter 3

# Software Design, Implementation and Testing

## 3.1 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.

### 3.1.1 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 [**?**] 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. A primitive algorithm will have to be developed to detect this as it is out of the scope of the project to create a 100% accurate way of finding it.

### 3.1.2 Website

The website will only have a few requirements:

- Display each record in the database in an easy to read manner.
- Highlight coding sequence in contig if possible.
- Copy coding sequence and +/- a user defined number of bases to the clipboard.
- Search the database for a gene, but name, description, ID, or even by nucleotide sequence.

## 3.2 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.

### 3.2.1 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 environment will match the development one.

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

### 3.2.2 Version Control

This project is being entirely tracked with Git version control. This allows the developer to have all of the work backed up, 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 [**?**] for remote backup and access via multiple hosts. Github also is widely supported for third party integrations, something that will be utilised quite extensively.

When pushing code to Github, a module called Husky [**?**] will run the test suite on the developers local machine. If the tests fail, the code will not be pushed.

### 3.2.3 Test suite

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

After the linting check, the test suite will use Mocha [?] 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 [?] 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.

### 3.2.4 Continuous Integration

Once the code has been pushed to Github, a hook on Github will trigger Travis CI [?] 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.

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

### 3.2.5 Deployment

If the tests pass on Travis CI, a Github hook will then activate the deployment. The project is setup to use Heroku [?], 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. This isn't an issue for development, however it will mean that when the project is to go live, funding would need to be found to pay for more data storage, or alternatively the project can be hosted elsewhere, most likely within the University network.

The benefits of this build process are clear, firstly code is automatically deployed, which saves the developers time greatly. 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.

## 3.3 Design

The first stage in designing this system was to decide upon what database technology would be used for the data store. As discussed in the analysis a NoSQL database called MongoDB has been chosen to store the data. This choice was also influenced by the choice of server side language, which for this project will be NodeJS.

NodeJS was chosen because:

- Client and server side languages are the same.
- Integrates well with MongoDB as it uses BSON which works with native JSON.
- Provides easy package management via npm.
- ES6 syntax is very nice.
- I have a lot of experience with it and learning a new language is outside of the scope of the project.

The website for this interface will be built off of nodestack**??**, a set of boilerplate code that uses several technologies including:

- 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 of open source libraries in the world.” [**?**] 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.

- 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 schema’s 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.

### 3.3.1 Overall 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 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 data object.

```
{ 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
  }
}
```

### 3.3.2 Some detailed design

#### 3.3.2.1 Even more detail

### 3.3.3 User Interface

### 3.3.4 Other relevant sections

## 3.4 Implementation

## 3.5 Testing

### 3.5.1 Overall Approach to Testing

### 3.5.2 Automated Testing

#### 3.5.2.1 Unit Tests

#### 3.5.2.2 User Interface Testing

#### 3.5.2.3 Stress Testing

#### 3.5.2.4 Other types of testing

### 3.5.3 Integration Testing

### 3.5.4 User Testing

# Chapter 4

# Results and Conclusions

# Chapter 5

# Evaluation

# Appendices

# Appendix A

# Third-Party Code and Libraries

# Appendix B

# Ethics Submission

# Appendix C

# Code Examples

# Annotated Bibliography

[1] W. Press *et al.*, *Numerical recipes in C.* Cambridge University Press Cambridge, 1992, pp. 349–361.

This is my annotation. I can add in comments that are in **bold** and *italics and then other content.*

[2] M. Neal, J. Feyereisl, R. Rascunà, and X. Wang, "Don't touch me, I'm fine: Robot autonomy using an artificial innate immune system," in *Proceedings of the 5th International Conference on Artificial Immune Systems.* Springer, 2006, pp. 349–361.

This paper...

[3] H. M. Dee and D. C. Hogg, "Navigational strategies in behaviour modelling," *Artificial Intelligence*, vol. 173(2), pp. 329–342, 2009.

This is my annotation. I should add in a description here.

[4] Various, "Fail blog," http://www.failblog.org/, Aug. 2011, accessed August 2011.

This is my annotation. I should add in a description here.

[5] S. Duckworth, "A picture of a kitten at Hellifield Peel," http://www.geograph.org.uk/photo/640959, 2007, copyright Sylvia Duckworth and licensed for reuse under a Creative Commons Attribution-Share Alike 2.0 Generic Licence. Accessed August 2011.

This is my annotation. I should add in a description here.

[6] Apache Software Foundation, "Apache POI - the Java API for Microsoft Documents," http://poi.apache.org, 2014.

This is my annotation. I should add in a description here.

[7] ——, "Apache License, Version 2.0," http://www.apache.org/licenses/LICENSE-2.0, 2004.

This is my annotation. I should add in a description here.