Registration number 100092523

2017

Loop Modelling Server for Proteins

Supervised by Dr. Steven Hayward

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

Computational biologists often need to model the structure of a loop in a protein as part of a process known as "protein homology modelling." In order to help to simplify homology modelling, Dr. Steven Hayward has created a MATLAB program based on research by Hayward and Kitao (2010).

This report is based on an attempt to create a web-based tool to expose this MATLAB program as a simple to use online software package. The report contains some basic biological foundation in order to understand the essence of the issue, as well as details about its development process, deployment instructions, and technical aspects of maintenance and modification of resulting software.

Writing of the current report has been done as a part of undergraduate program in Computing Science of University of East Anglia, Norwich, UK.

Contents

List of Figures

List of Tables

1 Introduction

Computational biologists often need to model the structure of a loop in a protein as part of a process known as "protein homology modelling." This activity involves comparing amino acid sequence of a protein, 3D-structure of which is unknown, to a similar protein, the template, 3D-structure of which has already been discovered. This step of amino acid sequence comparison allows to approximate the 3D-structure of target amino acid sequence based on the template. However, loop regions connecting protein's secondary and tertiary structures cannot be modelled in this a way. Therefore, computational biologists require a method to connect these loop regions to the ends of predicted 3D-structures in order to finish the modelling process.

1.1 Project Aim

Dr. Steven Hayward has created a MATLAB script to assist in the issue. The script is based on a paper Hayward and Kitao (2010) co-authored by Dr. Hayward.

The aim of this project is to create a web-based tool around the aforementioned MATLAB script. The tool – a term used interchangeably with the word "website" – is to present a convenient user interface to interact with MATLAB code. This includes providing a way for website's users to create, interact with, save and share resulting 3D-structures.

I am to deploy the produced software on a publicly available server for seamless online access. In this case, the server is hosted by UEA.

2 Related Work and Context: Literature Review

Given that the reader of this report might have no background in biology, this section will provide you with a basic insight into the project's underlying biological principles. Also, this section is a proof of my understanding of basics of protein structure.

The literature review consists of two parts. The first part contains general information about proteins, and is based on textbooks about protein structure (Branden and Tooze, 1999) and bioinformatics (Lesk, 2002).

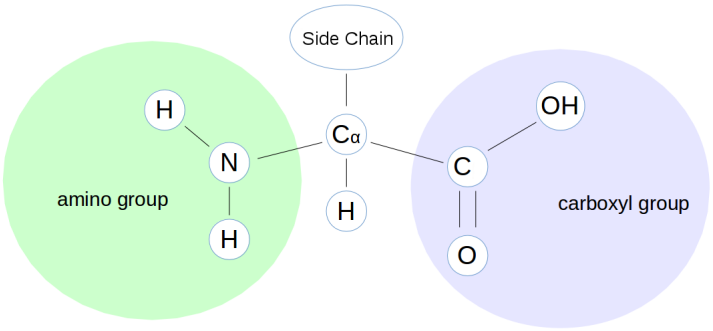

Figure 1: Structure of a left-handed amino acid. Right-handed amino acid would mirrored: the carboxyl group would be on the left, and the amino group on the right.

The second part is more advanced, and consists of condensed versions of papers read I read in order to better understand the process of protein homology modelling.

2.1 Basics of Protein Structure

Proteins are built up by amino acids that are linked by peptide bonds to form a polypeptide chain (Branden and Tooze, 1999). A peptide is made of two or more amino acids, and therefore a number of joined peptides form a polypeptide. An amino acid is a molecule that consists of a central carbon atom, C_α, a carboxyl group, $COOH$, an amino group, NH_2, and a side chain. All in all, peptide bonds are located between an amino acid's amino group and another acid's carboxyl group. Amino acid's structure is presented in Figure 1 on page 8.

There are 20 different amino acids used in protein chain formation, each of which has a unique side chain. The simplest one is Glycine, usually abbreviated as Gly or G, and its side chain consists of a single hydrogen atom, making it the least structurally complex amino acid. Different alternations of 20 basic amino acids produce an enormous number of possible peptide chains.

All amino acids, except for Glycine, are chiral – asymmetric and can be mirrored – and therefore can have different "hands". The "hands" are called the L-form and D-

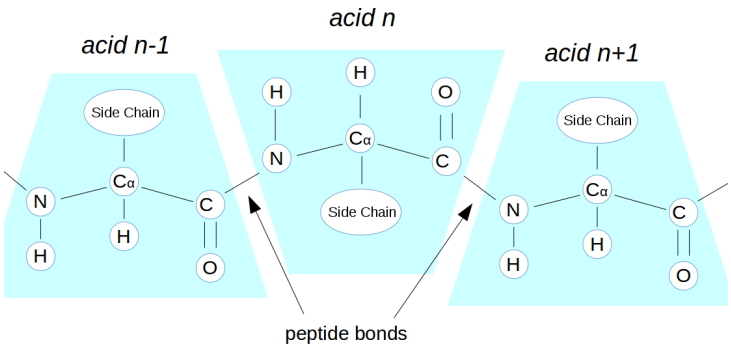

Figure 2: Structure of a peptide chain.

form depending on the way they are constructed and oriented. However, all of the amino acids in nature are left-handed, even though it is unclear why did the L-form become dominant during evolution.

Proteins rotate around two bonds of their C_α atoms, while other bonds are essentially fixed in their rotations. There are two essential torsion angles per C_α atom. The first one is located between the amino group's N and the C_α, and is called phi (ϕ). The second torsion angle is located between the carboxyl group's C and the C_α, and is called psi (ψ).

Since ϕ and ψ torsion angles are the only degrees of freedom in a peptide unit, the conformation of the whole main chain of a protein is largely determined by individual rotations of these bonds.

Most combinations of ϕ and ψ angles for an amino acid are not allowed due to collisions between the side chains and the main chain. However Glycine, for example, can adopt many more conformations because of its smaller and simpler structure. The angle pairs of ϕ and ψ are usually plotted against each other in a diagram called a Ramachandran plot (Branden and Tooze, 1999).

Protein's primary structure is just the polypeptide chain of amino acids. Secondary structure consists of structures, such as α-helices andβ-strands. Tertiary structure consists of various secondary structures connected by loop regions into a globular unit; these kinds of units are called domains. Quaternary protein structure involves the clustering of several individual peptide chains.

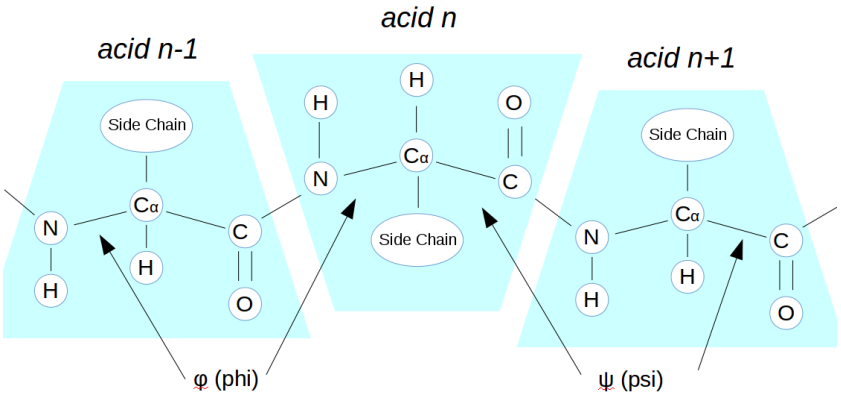

Figure 3: Locations of bonds, rotation along which affects ϕ and ψ torsion angles accordingly in a polypeptide chain.

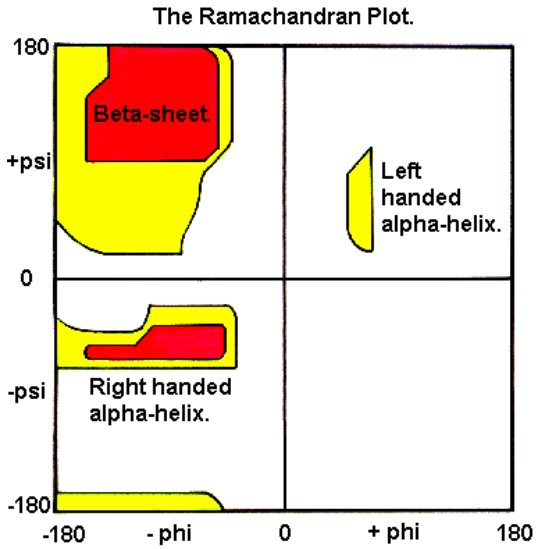

Figure 4: Ramachandran plot showing the possible ϕ and ψ torsion angle conformations for α-helices and β-sheets. Source: *http://www.cryst.bbk.ac.uk/PPS95/course/3_geometry/rama.html*

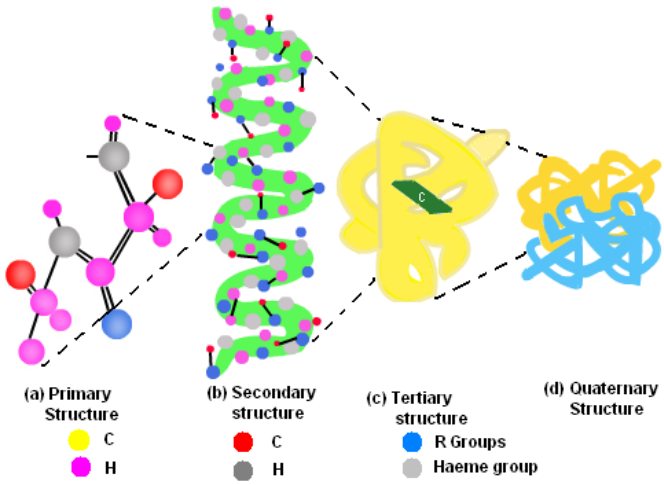

Figure 5: Types of protein structure. Source:
http:// chemistry.tutorvista.com/ biochemistry/ tertiary-structure-of-protein.html

The function of a protein is determined by its amino acid sequence and the overall globular structure it is folded into. The active site is a region of a protein which reacts with other chemical species. The positions loop regions of a protein might affect its function since these are able to open up or close "holes" in a protein leading to the active site.

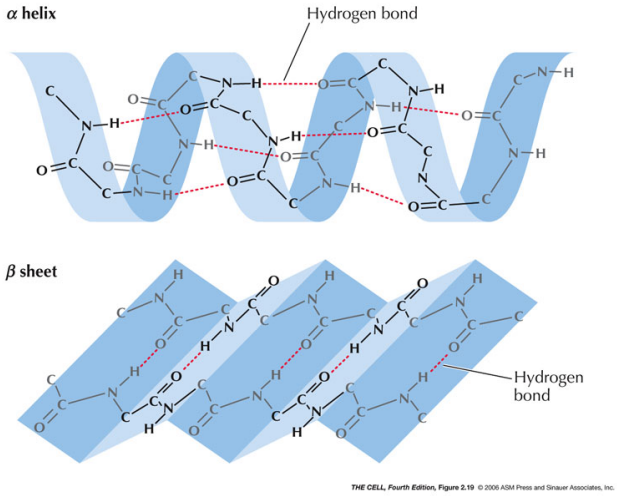

Fig. 6 *Structure of an α-helix and a β-sheet. Source: dnacamp.cgrb.oregonstate.edu*[1].

2.2 A Brief Introduction to Inverse Kinematics

As mentioned in the section 1, the process of protein homology modelling includes a step of connecting loop regions with predicted secondary protein structures on both sides.

"...loop modelling remains a difficult task for several reasons. For long loops, the number of available conformations is enormous, and rapid and thorough sampling is a challenge" (Canutescu and Dunback, 2003). MATLAB algorithm developed by Dr. Hayward attempts to solve this issue by implementing an approach based on inverse kinematics as described in a paper by Hayward and Kitao (2010).

Inverse kinematics algorithms are designed to move an "end effector" (e.g., a robotic gripper) to reach for a specific location to pick up an object by changing joint angles and modifying segment lengths. As such, it is essentially the same problem as loop closure in proteins or other molecules ... One algorithm used in robotics that is flexible in allowing constraints to be placed at each step, easy to program, conceptually simple, and computationally fast is "cyclic coordinate descent" (CCD) (Canutescu and Dunback, 2003). CCD is described in more detail in section 2.3.1 that reviews the paper Canutescu and Dunback (2003).

2.3 Paper Review

2.3.1 Cyclic Coordinate Descent: A Robotics Algorithm for Protein Loop Closure (Canutescu and Dunback, 2003)

The paper explains the base ideas and workings of a cyclic coordinate descent (CCD) algorithm, a concept borrowed from the field of robotics dealing with inverse kinematics. Inverse kinematics methods calculate the necessary changes in internal and external degrees of freedom in order for an object component to reach a desired position

[1] *http://dnacamp.cgrb.oregonstate.edu/w1d5_930.html*

(Canutescu and Dunback, 2003). Using this algorithm, the authors were able to derive conformations of protein loops to close at two arbitrary terminals.

The method described consists of fixing two atoms of the protein, from now on referred to as the N-anchor and C-anchor, between which the loop chain is to be inserted. In order for the loop to fit connect the two anchors, first step is to connect a carboxyl group of the loop structure to the N-anchor. Therefore, the problem becomes similar to that of robotics: having a flexible 'robotic arm', the peptide chain, with rotatable 'joints', the individual atoms, move the other unconnected end of the loop into position to connect to the C-anchor.

The procedure itself modifies each ϕ and ψ angle in sequence, using some simple vector and angle manipulations. This way, the procedure does not suffer from either complexity of implementation or inconsistent and unreliable matrix inverse calculations used in other proposed solutions to the problem. Since the algorithm is iterative and evaluates each of the angles separately, it also allows constraining any of the arbitrary angles, if needed. In addition, the rotations can be restricted using favourable Ramachandran plot angles, therefore increasing the accuracy of the proposed approach.

The algorithm requires an energy equation to be provided in order to succeed, and also does not check for any steric collisions of the backbone chain or the residues. In addition, even though it is probably a rare condition according to the authors, it might be the case that only the first few angles will change – enough to find a solution – and all of the others are going to remain as they were from the beginning. Therefore, CCD the algorithm just lies a groundwork, but is as an important and useful tool for more sophisticated solutions to protein closure problem.

2.3.2 The Effect of End Constraints on Protein Loop Kinematics (Hayward and Kitao, 2010)

The paper explores the topic of influence of amino acid chain's ϕ and ψ torsion angle constraints on movement of loop regions within the protein.

Loop regions take up many functions, one of which is to block or open a way for different enzymes and ligands to access protein's active site. In this paper, using various algorithms and techniques from previous works on protein loop movements and the field

of robotics – the inverse kinematics principles – the authors were able to find a way, even if it is simplified, to simulate loop movements while having its end points constrained: by constraining the corresponding angles of two C terminal atoms. In addition, one of the algorithms allows to not only 'lock' the end points in place, but also constrain any of the angles within the simulated loop movement, as long as there are at least six angles which can be changed.

One of the examples on the above-mentioned algorithm follows the torsion angle targeting of the enzyme horse liver alcohol dehydrogenase, LADH, and some of its mutants. By choosing LADH segment 290-301 as the modelled group, the authors constrained Pro residues at positions 295 and 296: ψ_{294}, phi_{295}, psi_{295} and phi_{296}. This is a necessary in order to allow to open and close up the protein's active site by moving to and away from residues 51, 56 and 57. Simulations with the original protein and different mutants (by targeting the torsion angles) have shown that, in order for the protein to open up its active site, the four above-mentioned residue angles have to be constrained so that to allow proper access and, if needed, isolation of the active site. The original protein acted as expected, blocking or opening up the access. However, the mutants have not been as successful in retaining the loop's function.

The algorithm used in simulations is the one that the project is based upon, and so understanding its possible uses for the protein modelling community is important. In short, the algorithm will allow the protein modelling specialists to target the torsion angles in order to see whether the predicted functional conformation of some protein is possible, perhaps to assess whether the model is correct or not, and also provide them with a tool to explore yet unknown or vaguely understood loop formations.

2.3.3 Monte Carlo Sampling with Linear Inverse Kinematics for Simulation of Protein Flexible Regions, (Hayward and Kitao, 2015)

This paper by Hayward and Kitao introduces a Monte Carlo Linear inverse-kinematics method for simulating protein chains with fixed ends.

Unlike in another paper, Hayward and Kitao (2010), where the loop movement simulation was done in small incremental steps, the method introduced in this paper is based on the Monte Carlo sampling, and therefore uses random numbers to change the protein

backbone's ϕ and ψ angles (see section 2.3.2 on page 13) in order to simulate the peptide chain's movements. Moreover, yet again unlike the previously reviewed paper by Hayward and Kitao, the focus of this research is to simulate an already existing protein's dynamics instead of trying to help model a loop closure.

The introduced technique calculates energy levels required to rotate bonds of a set of atoms and move them in space at each step; if the required energy exceeds a threshold, the move is rejected and recalculated with all of the atom's original positions before the move being restored. In addition, any of the angles of a peptide chain might be fixed, therefore allowing modelling of whole domains connected by a number of flexible regions to simulate their movement. Also, the Proline loops are made flexible in the underlying method as well, thus helping to avoid situations where movement of Proline residues might actually affect the change of protein's backbone conformations.

2.3.4 Clustering and Percolation in Protein Loop Structures (Peng et al., 2015)

This paper introduces a systematic quantitative approach to classify the fragments of building blocks of loops in crystallographic folded proteins using a generalised discrete nonlinear Schrödinger equation, an energy function, and a clustering technique on their basis.

The authors have made clusters of different loop fragments using high quality models from the PDB database by first defining a random loop fragment as an initiator of the cluster, and then all of the other loop fragments are either assigned to the same cluster or to other ones depending on prescribed the root-mean-square cut-off distance deviation (0.2Å). The clusters determine possible torsion and bond angles of loop fragments, and can be used to analyse and determine loop structures.

Peng et al. were able to determine that identical sequences sometimes correspond to different structures in different proteins, and therefore a sequence does not necessarily determine a unique structure. In addition, the authors have figured out by doing a local chain inversion operation on the cluster fragments that in case of protein loops, local chain inversion – unlike the unchanging α-helices and β-strands – does not provide either a way to find new clusters or map the clusters onto themselves, and thus the local chain inversion is a broken symmetry in case of protein loops. Moreover, Peng et al.

speculate that uncommon loops' nature is most probably related to function, as shown in their analysis of myoglobin.

All of the data analysis was done using PDB structures with resolution of 1.0Å or less for increased accuracy.

2.3.5 Comprehensive Structural Analysis of the Open and Closed Conformations of *Theileria Annulata* Enolase by Molecular Modelling and Docking (Mutlu et al., 2016)

The paper provides a sound overview of a variety of tools and techniques available to model a protein by someone knowing only its amino acid sequence.

Mutlu et al. performed a sequence analysis by using the information available from various literature and the NCBI protein database to retrieve similar sequences, and then aligning them by using MUSCLE multiple sequence alignment tool. The results were validated by using ERRAT protein verification server, ProSA, the Protein Structure Analysis tool, ProQ, Protein Quality Predictor, and RAMPAGE, Ramachandran Plot Assessment tool – all showing promisingly high results.

Then, the authors performed a prediction of secondary structure by using PSIPRED protein sequence analysis workbench, and then a domain analysis using CATH-Gene3D protein structure classification database. Finally, the active site was predicted by another set of tools not relevant to this review.

In conclusion, Mutlu et al. have managed to predict three-dimensional structures of open and closed conformations of the target organism's protein using the homology modelling method, and also identify possible regions to develop new drugs to fight the parasite.

2.4 Excluded Papers

Paper by Blundell et al. (1988) has been excluded from the review due to its more basic contents and looks on homology modelling. Nevertheless, the paper provides a good overview of conceptual basis of homology modelling; however, more recent papers had been prioritised in this review.

Paper by Kolodny et al. (2005) has also not been included into review even though its topic closely correlates with the main topic of the project. It has not been reviewed since I have already read two other papers – by Canutescu and Dunback (2003) and Hayward and Kitao (2010) – and therefore I thought that adding this thesis would be redundant.

I have immediate access to over 20 more papers outside this review that have not been included however, due to little return on spent time, I did not read them in a sufficiently slow and detailed manner.

2.5 Conclusion

Homology modelling of proteins is a complex, though important, topic in scientific community, and methods used while applying this technique are numerous and vast.

By reading and attempting to dissect various literature on topic of protein structure – and homology modelling using inverse kinematics methods in particular – into more manageable slices of knowledge, I managed to achieve an elementary level of understanding of more trivial aspects of the field of computational biology.

3 Design & Methodology

Formal initial step to designing a relatively complex system is either similar system analysis or creation of use cases. However, due to relatively unique nature of the project, I have decided not to perform a comprehensive similar system analysis. Additionally, there were no use cases written, because of my very weak understanding of the project's ultimate purpose at the design stage.

Dr. Hayward, the project supervisor, has suggested the following rough set of website functionality. A user should be able to:

- select a protein by either providing a PDB[2] code or uploading their own structure, also providing the first and last ordinals of amino acids comprising the loop and selecting the chain to use in case the protein consists of multiple chains.
- select and preview a protein loop segment, including initial torsions in order for the user to see what exactly one is working with.
- provide a value in degrees to target angles of each torsion to, or to constrain them.
- view the resulting loop segment and its final torsion values, including the difference between the actual value that the underlying script has generated and the target value provided by the user. This way, a user can see what the initial structure was, and how well could it be aligned to target parameters.
- hide his or her own protein structures from public view when using the tool.

I have built the following sections basing any further design decisions on this short list. Also, I have added some functionality requirements of my own. A user should be able to:

- use the website in a number of languages.
- sign up and log in to store his or her protein structures.
- share his or her protein structures on the main page.

[2]RCSB Protein Data Bank

Moreover, I put a single restriction on the website: it will not work with mobile devices.

3.1 Generic Architecture Overview

One of the easiest and more modular ways to build a website is to use an API-based approach. The process of serving any content on a page is as follows:

1. The client (browser) requests an initial page – the homepage, for example.
2. The server replies with contents of the requested page.
3. Embedded within the page is JavaScript code that sends a request for additional data to the server. In this particular project, the JavaScript code might ask for a protein model, or protein meta data.
4. The server replies to the incoming request.
5. Go to step 3 until all of the requests have been made.

Apart from the server-side page-serving and client software, there are a number of other high-level components to select:

- a database to store user and protein data.
- an AJAX[3]-based framework so that it is easy to fetch data from and send it to the server in real-time.
- molecular viewer for proteins.

Figure 6 depicts a high-level overview of the proposed system.

[3] Asynchronous JavaScript and XML

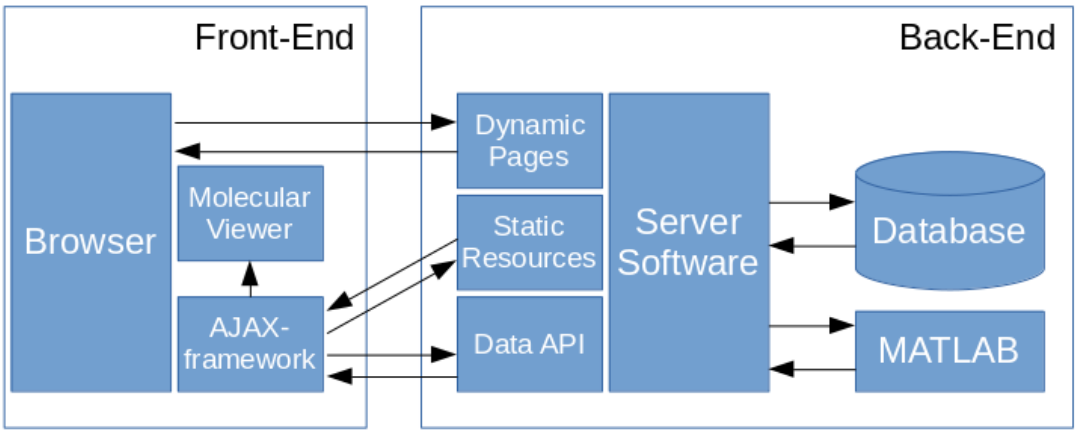

Figure 6: High-level overview of the system. The browser software loads pages created by the server software. The pages contain scripts for AJAX-framework to fetch other data from static APIs and data APIs. The framework feeds the acquired data into the molecular viewer. On the back-end side, the server software interfaces with the and MATLAB script to provide its functionality.

3.2 Component Selection

3.2.1 Web Server

The project requirements have revealed that the server will not have to serve many requests, and therefore the simplest and easiest to maintain solution is preferred.

From the beginning, I have considered using Node.js[4] platform and a framework like Express[5] to serve requests to the client-side part of the project. Node.js uses an event-driven, non-blocking I/O model. It runs on a single thread, but can accept up to 600,000 concurrent websocket connections if set up properly, according to Kleveros (2015). Running on a single thread removes the need to deal with concurrency issues, and thus leads to code simplification.

However, since there are numerous alternatives to these technologies, I have decided

[4] *https://nodejs.org/en/*

[5] *http://expressjs.com/*

to justify my choice.

Alternatives to Node.js Since every programming language and platform has its own downsides and upsides, and because the project does not require either high throughput or incredible efficiency, the choice was almost solely based on the current popularity of the chosen platform.

The first resource that I have accessed was an entry at a crowdsourced website AlternativeTo (2016). Only some of the platforms suggested by the website were taken into account, while others not even considered to be used in the implementation. Some of the ignored ones included:

- **Java** due to my previous experience with it. The programming environment is too inflexible for simple and rapid web server development. Even though the Java programming language is dominant within the UEA School of Computing Sciences, due to lack of experience in developing web applications using the language or any of its web frameworks, in addition to Java's ineffective database manipulation syntax, I have decided to refrain from using the language. Lastly, managing concurrency in Java requires additional effort, while Node.js is immune to the problem. For a more detailed discussion, see Wayner (2015a).

- **PHP** is a web-oriented programming language, but its inconsistency and prevailing tendency to write unmanageable code lead to not even considering PHP to develop the back-end of the client-server application. Also, the language feels obsolete despite its popularity among the web-development community – especially the older generation – and therefore, should not be used in new projects. For a more detailed discussion see Wayner (2015b).

- **ASP.NET** is based on Microsoft's C#, which can be used with Mono compiler to run on multiple platforms. The choice against this platform is its rigidity, like Java, and infrequent usage within the UEA computing school's undergraduate courses. Yet again, same as with Java, managing concurrency in C# needs additional effort. For a more detailed discussion see Bandt (2015), Petrov (2016) and Perrenoud (2015).

- **Haskell** is a functional language allowing to write far safer code compared to imperative programming languages. The language also has numerous web frameworks available[6]. However, due to low popularity and initially hard to learn concepts, I have decided not to use Haskell despite having some limited experience of working with the language. For a more detailed discussion, see gawl et al. (2010).

Ruby & Ruby on Rails The most popular Ruby web framework is Ruby on Rails (RT, 2016), with Rack being the basis of Ruby on Rails. According to personal experiences of Carr and community (2016), Ruby is a very powerful and expressive language, but its large stack frames make the debugging difficult. Node.js's most relevant issues are error handling, or rather a lack of consistency in it, and maintainability due to missing coding conventions.

However, due to Ruby being not that popular of a language, compared with JavaScript, Node.js seems more relevant for use for the project. The new consideration is that there needs to be some coding standard to conform to in order to keep Node.js application maintainable.

Python According to Kaplan-Moss, lead developer of the Django framework, the biggest pros of Node.js are that it is built to handle asynchronous I/O from the ground up. Also Node.js is simply JavaScript that the bigger part of programming community has seen and familiarised themselves with.

Python has a set of web frameworks to work with, but it has two considerable setbacks that make development of web applications in Python rather inconvenient:

- Discrepancy between Python 2 and Python 3. Python 2 is installed by default on almost all of the Linux machines, but this lack of cohesion leads to bewildering errors in otherwise correct code written for a different version.

- Unicode support in Python requires some tweaking in order to work. Python defaults to ASCII, and therefore international support becomes hard and time consuming.

[6] *https://wiki.haskell.org/Web/Frameworks*

In a follow-up article Vickery (2016), Vickery says that Node has its own set of hard to deal with problems, and notes that Node.js is not to be used for big projects. Since this project's back end does not need to be a complicated and incredibly scalable system, Node.js's flexibility still leaves it a better choice than Python. The biggest noted drawbacks of Node.js by Vickery are:

- Poor error handling. This has been encountered before while discussing Ruby in section 3.2.1 on page 22.
- Multiple callbacks – that is when some asynchronous event has been dispatched, and the callback handles the event's completion – make Node.js code hard to read. Remedy to this are promises and generators, but they require new concepts to be learnt. The project's web server is going to be quite simple, and therefore I assume that the so-called "callback hell" will not plague the code. However, this may require some coding standards to be applied.

Discussion: Node.js After careful deliberation, I have kept my original decision to use Node.js as the back-end technology of the server. Pros of Node.js and some framework with it are:

- rapid prototyping.
- lack of concurrency issues.
- popularity, active community and big code base.
- nativity of JSON[7] object format, and therefore interoperability with client-side JavaScript and back-end database storage if JSON is to be used.

Cons of Node.js are:

- lack of coding standards.
- poor error handling.

[7] JavaScript Object Notation *http:// www.json.org/*

- constantly and rapidly moving ecosystem.

Lack of coding standards are to be managed by introducing some coding standards while writing application code; so is the poor error handling. The rapidly moving ecosystem can be avoided by only using big, stable and popular packages available on the Node.js's NPM package manager.

See Aimonetti (2013) for a general overlook of technologies that start-ups might use. This kind of information is very close to the aim of this section: to choose a technology stack to use.

Framework for Node.js By reading Glock (nd), abhishek (2015), Arora (2016) and Yang (2016), it became apparent to me that Express is the most popular JavaScript framework, however it does not provide sufficient error-checking capabilities, and also tends to produce hard to read code. Even though other frameworks, like Koa and Hapi, provide a better approach to error handling, their relatively low popularity have kept me from using them.

Having also read Goldberg (2016), a conclusion has been made that it would be the most beneficial to use Express framework with the *express-generator* module. Additionally, it is going to be helpful to follow the best practices in error handling and clear code writing found online.

Server Database Engine Since Node.js together with Express are to be used at application's back-end, the best storage choice is to use JSON objects, and propagating them through the whole technological stack for simplicity and code clarity. Thus, a chosen database engine needs to be able to store data in JSON format.

A clear separation of code used in JavaScript could would allow to easily provide a common API to swap out the database engine, and therefore any popular JSON-capable database will fit the project purposes.

Using Wikipedia (nd) to find a list of JSON-capable databases, and also adding MySQL starting with version 5.7[8] to this list, only the most widely used databases have

[8] *https://dev.mysql.com/doc/refman/5.7/en/json.html*

been considered for this round. Having found the most popular JSON-capable databases using DBE (2016), I narrowed the list to these databases (from 10 most popular):

- MySQL
- PostgreSQL
- MongoDB

This choice was trivial. MySQL is the most popular and quite lightweight database that supports rigid SQL schemas. Thus, it is the choice for the project.

3.2.2 Chosen Back-End Components

- Web Server: **Node.js**
- Web Framework: **Express.js**
- Database: **MySQL**

Additional components:

- Database Connector: **Sequelize**[9]. Chosen as the most flexible ORM[10] for SQL databases.
- SASS[11]-parser: **node-sass**. Most popular CSS extension language. Mostly used to change CSS properties as variables.
- Login session manager: **Passport.js**[12]. Widely used authentication middleware for Node.js.

[9] *http://sequelizejs.com/*

[10] Object-relational mapping

[11] *http://sass-lang.com/*

[12] *http://passportjs.org/*

3.2.3 Client-side

AJAX-like JavaScript Framework In order to keep things as simple as possible, I have chosen the framework I am most familiar with and which has the following set of functions:

- Ability to send POST and GET HTTP requests to the server dynamically – without reloading the page.
- Ability to iterate through collections to dynamically create their formatted HTML view containers.
- Ability to bind[13] input forms and images/text labels to objects contained within JavaScript code. This is needed for both forms to work and languages to change without the need to reload the page.

This framework is AngularJS[14]. Although AngularJS is a powerful framework, JQuery[15] is also used to manipulate some of the smaller UI components in order to abstract them out of AngularJS controllers. You can refer to a blog post by Ivanovs (2017) for more details on other frameworks.

Molecule and Protein Viewer Another critical part is the protein molecule viewer. Dr. Hayward has recommended JSmol[16] as a molecule viewer, but it is not the only one I have found. Another one is simply called PV[17], or Protein Viewer.

JSmol is based on Jmol[18], a Java viewer for chemical structures. Jmol has extensive interactive scripting documentation[19], even if it seems bloated. On the other hand, PV

[13] Update the view as soon as underlying data is changed in the associated object or collection.

[14] *https://angularjs.org/*

[15] *https://jquery.com/*

[16] *http://wiki.jmol.org/index.php/JSmol*

[17] *https://biasmv.github.io/pv/*

[18] *http://jmol.sourceforge.net/*

[19] *https://chemapps.stolaf.edu/jmol/docs/*

is lightweight and fast, but after a short period of testing, the conclusion has been that its API lacks a simple way to load and animate a number of protein structures. JSmol handles this operation easily in a single line of script code.

The protein loop needs to be animated in order to show how exactly did MATLAB script fit the loop segment to its position. This is one of the requirements that have been discovered during iterative experimental construction of the website.

Therefore, the preferred and currently used molecule viewer is JSmol.

CSS Framework The final part of choosing software for client-side system is a CSS framework. CSS frameworks are employed to specify a page's layout seamlessly. Additionally, they allow to style the appearance of a website in a consistent manner.

On my whim, I have decided to find a framework that gives me a way to easily create a column-based layout like Twitter's Bootstrap[20], but it should be styled following Google's Material Design guidelines[21]. After reading Ivanovs (2016), I have decided to use Materialize[22]. The reason for that is Materialize's popularity on GitHub[23] (Ivanovs, 2016), and its simple and functional API (discovered through a simple playground test).

Finally, JQuery-UI[24] framework has also been added to the project to simplify the animations of columns laid out using Materialize.

Chosen Client-Side Components

- Framework: **AngularJS**.
- Molecule viewer: **JSmol**.
- CSS Framework: **Materialize**, and also **JQuery-UI**.

[20] *http://getbootstrap.com/*

[21] *https://material.io/guidelines/material-design/introduction.html*

[22] *http://materializecss.com/*

[23] *https://github.com/Dogfalo/materialize*

[24] *https://jqueryui.com/*

3.3 Development Methodology

Even though I had had some experience in writing Node.js and AngularJS code, numerous parts of the system were unknown to me. According to, Noppen (2014) "You only know how to do something after you have done it," so my approach has been highly chaotic, agile, and relied on weekly feedback from Dr. Hayward.

The development time line has never been rigid, and there have been few milestones and deadlines. As such, a Gantt-chart of the project is not present in this report. Instead, a short list of most important events is presented below.

- (26th October 2016) Literature Review submitted.
- (12 December 2016) Showcase of minimal working prototype.
- (27 April 2017) Live website deployment.

On the technical side, classes in neither Node.js nor client-side JavaScript were used, since the use of classes would not have made the code any more readable. API routes in Node.js code and hard-coded objects in AngularJS controllers are enough to keep a moderately-sized code base clear and consistent.

3.4 UI Design Framework

Simplicity in interaction has been the goal from the very , and so I decided to create a single-paged application for the tool itself. The application would contain three distinct "stages," which would have the following functionality:

1. First stage (cut form): form for a user to details of the loop segment to extract.
2. Second stage (segment): view the loop segment and choose which torsions to target or constrain.
3. Third stage (transform): view the animated loop segment, going from the initial segment to transformed[25] structure, and view the results.

[25] Resulting structure after targeting and constraining the torsions.

Also, a navigation bar is to be present on all of the pages, which has the following functionality:

- A sidebar that can be opened or closed, which is to contain all user-related data: user's previous segment and transforms, and also the "log in" and "log out" buttons.
- A set of navigation buttons to transition between the stages of the application (cut form – segment – transform).
- A button to change the application's language.
- A button to navigate to the homepage.

Design of the segment and transform stages is to be almost identical. A stage should be able to exist in two states:

1. "Open" state: the left hand side of the screen is taken by the molecular viewer and its controls. The right hand side should be taken by a form or result data.
2. "Closed" state: the whole screen should be taken up by the molecular viewer.

Toggling the states should be made possible by clicking a button on the molecular viewer's side. This design has been inspired by the tabbed window design in Xcode IDE on MacOS on smaller screens. The navigation bar, on the other hand, should look like a typical Material Design side menu, as found on modern Android OS devices.

Refer to the results – subsection 4 on page 35 – to see what the resulting design looks like.

3.5 Database Design

Since the infrastructure is very simple, the database only has 4 tables presented in figure 7 on page 30.

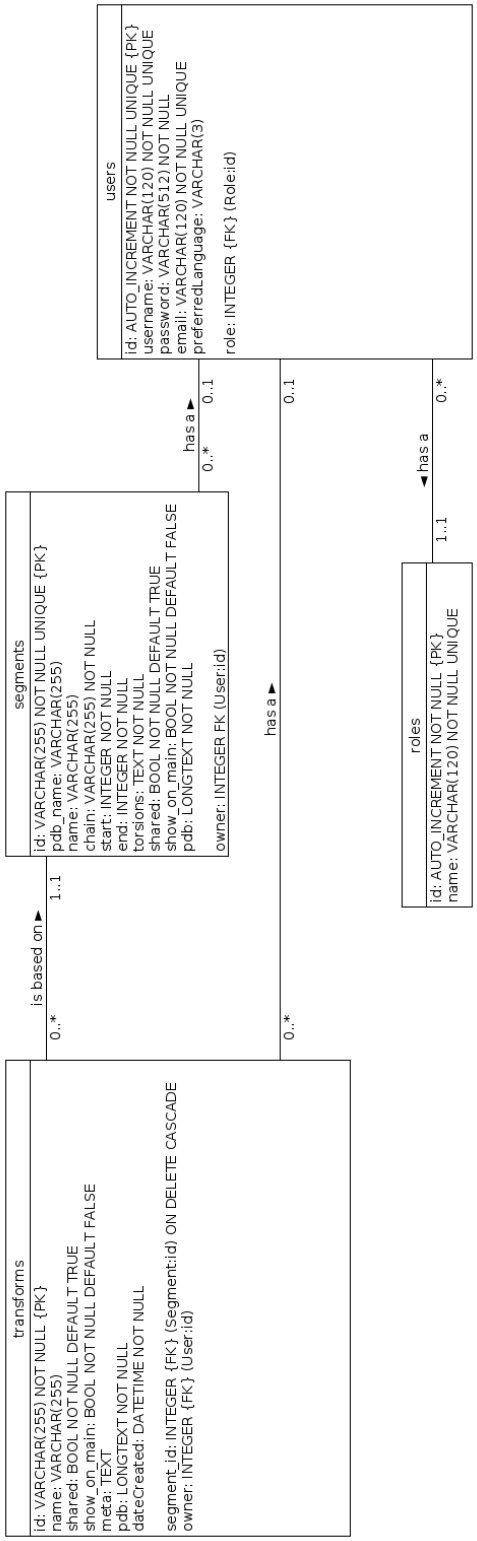

Figure 7: UML Entity-relationship diagram of the database schema.

Users Table The *users* table consists of the following fields:

- **id** – used to uniquely identify a user within the system.
- **username** – currently not used, and email is simply copied into it. Could be adapted in the future.
- **password** – user's hashed and salted password.
- **email** – user's email address. Used as unique identifier when logging in.
- **preferredLanguage** – user's preferred language as an ISO-639-1 code[26].
- **role_id** – reference to the *roles* table; used to distinguish users from administrators.

Roles Table The *roles* table is used to simply contain possible roles of registered accounts. Currently only two roles are present: "user" and "admin."

- **id** – used to uniquely identify a role.
- **name** – human-readable name of the role. Only to be used within the system.

Segments Table The *segments* table consists of the following fields:

- **id** – used to uniquely identify a segment. Generated in the server, not in the database.
- **pdb_name** – PDB code in the Protein Data Bank. Might be null if this segment is based on user's own PDB file.
- **name** – name of the protein. This is supplied from a PDB file from Protein Data Bank, or a PDB file uploaded by a user.
- **chain** – name of the protein chain that contains the loop.

[26] *https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes*

- **start** – First amino acid number of the loop segment.
- **end** – Last amino acid number of the loop segment.
- **torsions** – JSON converted to string that contains initial torsion angle values of this segment.
- **shared** – flag to show whether this segment can be accessed by link by anyone but the owner. Currently not used.
- **show_on_main** – flag to determine whether to display any of the transforms derived from this segment on the main page.
- **pdb** – PDB data of the segment. Contained in the database in order to couple the meta and the data instead of storing the data in a single file which could be lost.
- **owner** – reference to the *users* table; used to determine who created the segment. Might be null if a user was not logged in when creating it.

Transforms Table The *transforms* table consists of the following fields:

- **id** – used to uniquely identify a transform. Generated in the server, not in the database.
- **name** – name of the transform. Currently not used.
- **shared** – flag to show whether this transform can be accessed by link by anyone but the owner. Currently not used.
- **show_on_main** – flag to determine whether to display this transform on the homepage.
- **meta** – meta data; currently used to store meta (resulting values) in a JSON object converted to a string.
- **dateCreated** – date when this transform has been created.

- **owner** – reference to the *users* table; used to determine who created the transform. Might be null if a user was not logged in when creating it.
- **segment_id** – reference to the *segments* table; used to determine which segment this transform is based on.

Please note that tables are created by Sequelize ORM framework, and thus there is no need to write SQL queries to create the tables.

3.6 Error Codes and Languages

I have developed a simple system to propagate error codes all the way to the client browser in case one happens. Every API that returns a JSON object always contains a "_meta" object, which in turn contains a "code" key. The code can be used to show the user what exactly has gone wrong in the system. Table 1 on page 34 contains all of the possible codes and their explanations in English as of May 2nd 2017.

The error codes are grouped as follows:

- 0-99 Generic errors.
- 100-199 MATLAB errors (written into a file by MATLAB scripts).
- 200-299 API data integrity errors.
- 300-399 Authentication and registration errors.

The languages are stored as JSON files in a public directory. They can be accessed by JavaScript code within the user's browser to load an new version with required language from the server. This allows to change the language without reloading the page. The language preference is then to be stored in browser's local storage and in user's profile, if he or she is logged in. This way, the next time the page is refreshed, it still is shown in the selected language. Default language is English, and an English JSON is to be served with every page of the website as "_lang" variable in JavaScript global namespace.

Table 1: Error codes and their corresponding meanings in English.

Code	Meaning	Notes
0	Unknown server error	Sent when the cause of error is unknown or should not be shown to the user.
10	Success	Sent with any successful API call.
100	No torsions are being targeted	
101	Targeting and constraining the same Phi torsion	
102	Targeting and constraining the same Psi torsion	
103	Zero degrees of freedom, cannot target	
110	There may be a break in chain	Usually returned when the chain name was incorrect.
200	No PDB code or own PDB file provided	
201	No chain name provided	
202	Start and end of a segment must be positive integers with start being a smaller number than the end	
203	No segment ID provided	
204	No transform ID provided	
205	No segment with this ID exists	
206	No transform with this ID exists	
207	Target torsions are in incorrect format	Refer to source code for format.
208	Constrained torsions are in incorrect format	Refer to source code for format.
209	Provided PDB file is in wrong format	Returned if there are no atoms or no TITLE header.
300	Wrong credentials	
301	Authentication required	
310	Wrong email format	

311	Weak password	8 symbols and at least 1 number required.
312	User with this email already exists	
350	Administrator privileges required	

4 Results: Loop Torsion Driving Tool

The resulting tool is titled the "Loop Torsion Driving Tool", or "TorsDrive." It can be accessed at *http:// torsdrive.cmp.uea.ac.uk/*. This section is a very short version of the tutorial[27]. The purpose of this section is to introduce the tool's resulting functionality.

The first stage of protein transform process is the Cut Form. Here you can choose whether to upload your own PDB file or simply specify a PDB code as in the Protein Data Bank. This scenario shows how to use a PDB code.

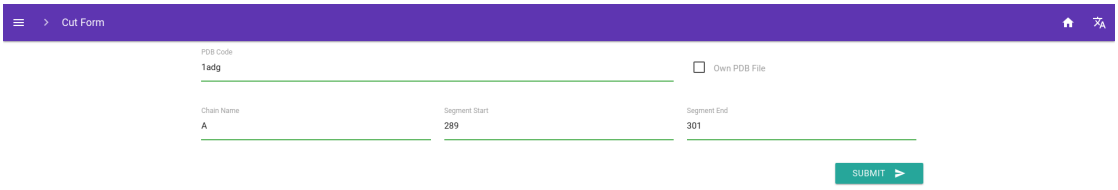

Figure 8: The initial stage: Cut Form.

The **chain name** defaults to "A". If you are not sure or if he protein only contains a

[27] *http:// torsdrive.cmp.uea.ac.uk/ tutorial*

single chain, leave it as-is.

Segment start is the ordinal of the first amino acid of the loop. **Segment end** is the ordinal of the last amino acid of the loop. Note that the minimum range is 6 residues, and the maximum range is 40. Also, is might take a long time to work with a segment that is more than 20 amino acids long.

After you click the *Submit* button (or press the "enter" key), you will get to the Segment View.

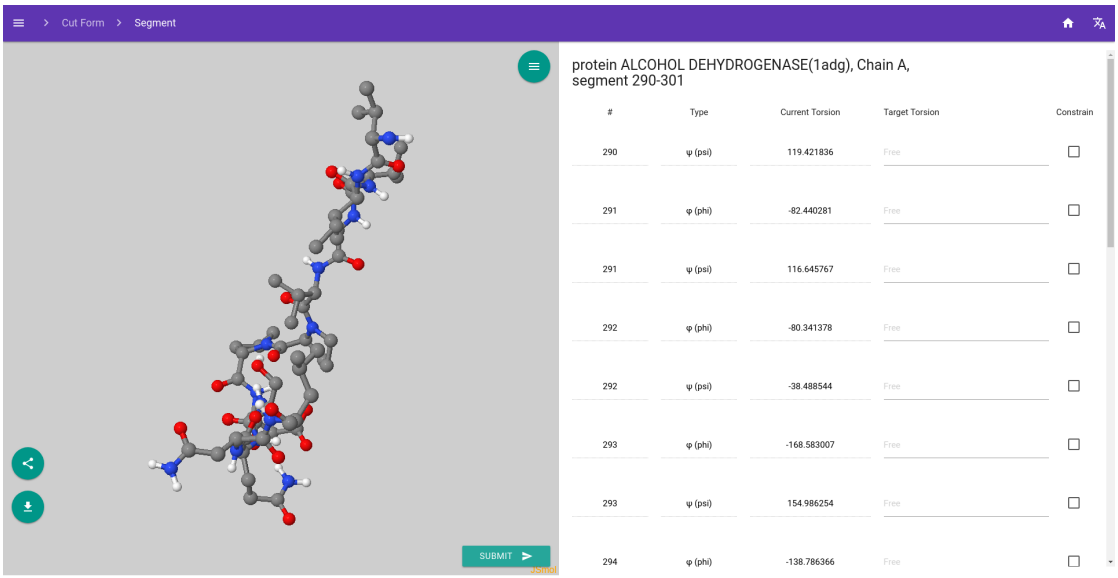

Figure 9: The second stage: Segment View. Note that you can now navigate between different stages of the process by using the navigation bar. Additionally, share, download and menu toggle buttons appear on the left and right-hand sides of the JSmol container.

The Segment View contains a JSmol view on the left, and the transform form on the right. The transform form allows you to enter target values for all torsions of the segment. You can constrain any torsion as well, effect of which will be identical to that of copying the current torsion value into the target torsion box. The torsion values must be numbers between 180 and -180.

As soon as you are done entering the values, click the submit button to transform the loop segment and process to the Transform View.

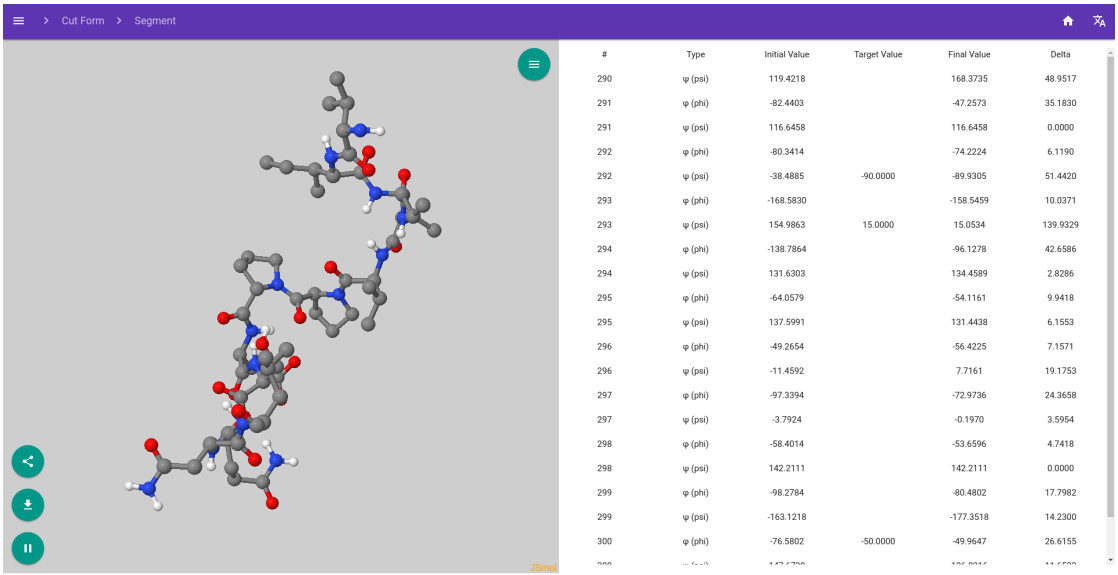

Figure 10: The third and final stage: Transform View. Note the addition of play/pause button in the bottom left of the JSmol container.

The Transform View allows you to see both the initial segment and the resulting transformed segment. In order to see the transformed segment, pause the animation and move the range selector to the rightmost position.

The results are displayed on the right-side panel, where the transform values form used to reside. The column names should be self-explanatory, albeit a single detail is worth mentioning: if the **Delta** value is 0.0000, it means that the torsion has been constrained.

If you were to click the button in the left top corner of the screen, you would open up the user segment tab, which contains your previous segments and transforms. Note that if you are not logged in, the segment you currently have saved are going to be lost. If you managed to create a transform of the segment and did not use your own PDB file, it will have appeared on the homepage by now.

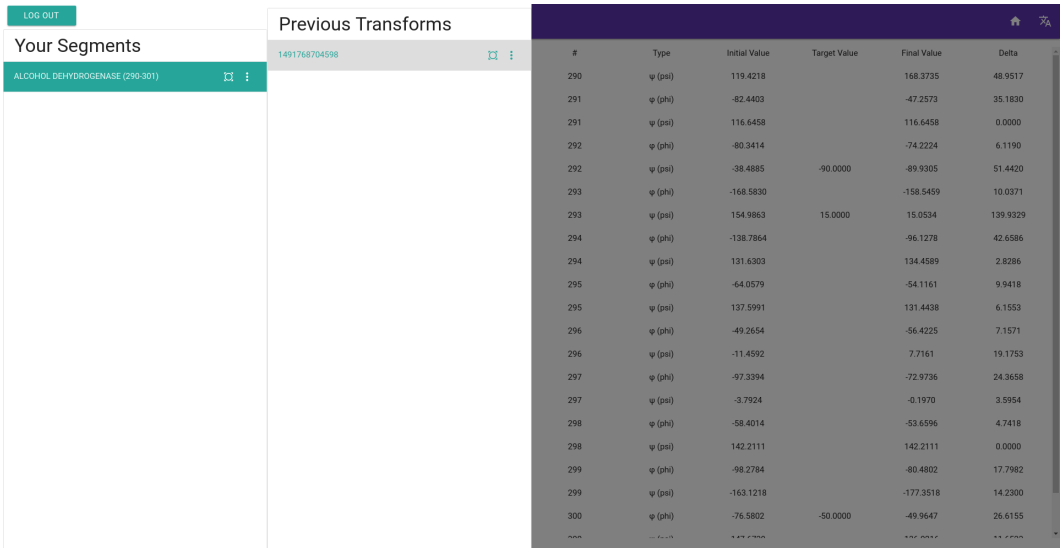

Figure 11: User tab. Note that a user is currently logged in. If that is not the case, sign up and log in buttons would be present in the top part of the tab. Clicking on the view button is going to open the selected segment/transform, while the context button allows you to share, download or delete it. You cannot delete a structure if you have not been logged in when creating it.

Finally, to pick another language click the button on the top-right corner of the screen.

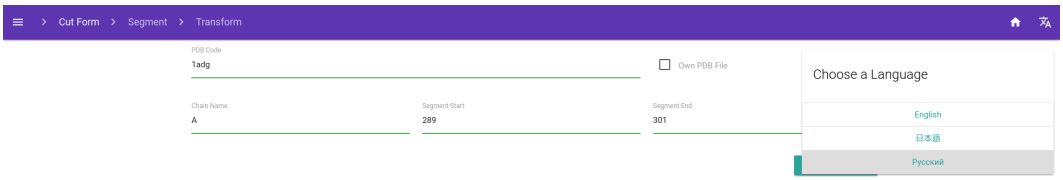

Figure 12: Language selection box.

4.1 Implementation Details

This section describes a small subset of technical problems encountered during the implementation stage of the project.

4.1.1 MATLAB Script

Since the original MATLAB script by Dr. Hayward had only a single point of entry for the whole process, I had to split it into a set of scripts. The scripts can be found in directory */tosdrive/matlab*. The main entry scripts are as following:

1. **Segment_cut.m** retrieves a PDB file from the Protein Data Bank using the PDB code, and outputs the results (segment and some meta data) in a set of files. See comments inside the file for more details.

2. **Segment_cut_own_pdb.m** uses a provided PDB file, and outputs the results (segment and some meta data) in a set of files. See comments inside the file for more details.

3. **Loop_Modeller_Transform.m** performs a transform while given a segment in a PDB file. Outputs the results (transform and some meta data) in a set of files. See comments inside the file for more details.

4.1.2 MATLAB Interaction

For underlying scripts to work, MATLAB has to be launched in a non-interactive mode. I have achieved this by wrapping a MATLAB command around an additional set of Bash and MATLAB code, which allows to exit the MATLAB prompt as soon as the command has finished executing. A typical command looks like this:

```
cd ./matlab && matlab -nodisplay -noshplash -r "try; SOME_COMMAND;
catch exception; disp('exception'); end; exit;"
```

For more details, see file *torsdrive/strings.js* and APIs in *torsdrive/routes/index.js* file.

4.1.3 Authentication

Passport.js middleware is used for user authentication. The user data is simply stored in the `users` table in the database. Passwords are hashed using SHA-256 algorithm and then salted using a randomly generated string, which is stored in the database alongside the password hash itself.

For more details, see files in *torsdrive/auth* directory and APIs in *torsdrive/routes/users.js* file.

4.1.4 JSmol, UI and AngularJS

All of the JSmol-related code is contained within files in directory *torsdrive/public/js*, namely *loop_controller.js* (animation) and *view_functions.js* (initialisation).

JSmol's loading happens in *view_functions.js*, which also controls all of the UI stage transitions.

```
/**
 * Initialises JSMol if it has not been initialised yet.
 * If it has been initialised, simply loads the data from the URL.
 *
 * @param {String} url URL to load, pointing to PDB data.
 * In form '/getPdb...': without the host.
 */
view.initJsmol = function (url) {

  if (!vars.jsmolLoaded) {
    // JSMol init info.
    var info = {
    color: "#CECECE",                // Background colour.
    height: "100%",                  // Size of the object.
    width: "100%",
    use: "HTML5",
    j2sPath: "js/jsmol/j2s",         // JSMol functions path.
```

```
    src: url,                          // Source of the model.
    serverURL: window.location.host, // Current URL.
    disableInitialConsole: true
  }

    // Initialise JSMol.
    select.jsmol().html(Jmol.getAppletHtml("jsmolApp", info));
    vars.jsmolLoaded = true;
  } else {
    Jmol.script(jsmolApp, "load '" + url + "'");
  }
};
```

On the other hand, asynchronous loading of JSmol animated models, range selector functionality and play/pause function are all handled in AngularJS controller in file *loop_controller.js*. See functions *toggleJsmolPlay*, *detectJsmolRangeChange* and *segmentTransform* for more details.

4.1.5 Languages

Language change in real time without page reloading is made possible by using AngularJS and LocalStorage object at the same time. As soon as the new language is requested (see *loadLanuage* function in *torsdrive/public/js/loop_controller.js*), it gets loaded from a public server location. Then the *$scope.lang* variable gets replaced by the retrieved language, and *localStorage.preferredLanguage* is set to match the current language's ID. As soon as the page is reloaded, it tries to download the stored language, thus automatically translating the page.

When *$scope.lang* variable gets replaced, it changes all of the language-bound values on the page by utilising AnglularJS' data binding[28], which requires no action on the part of the programmer except for binding the values in HTML beforehand. See file *torsdrive/views/home.ejs* for an example of data binding (look for *aVars.lang.XXX*

[28] *https://www.w3schools.com/angular/angular_databinding.asp*

bindings).

4.1.6 Error Propagation and Handling

Error codes displayed in a table 1 on page 34 are used to propagate errors through the whole server-client stack. Usage of error codes is trivial. The whole process of handling an error can be summed up as follows:

1. Make a request to some of the server's APIs.
2. Check the returned *_meta* object. It contains a code that is other than success code (10).
3. Retrieve the error string for current language by accessing *aVars.lang.errors[code]*.
4. Show the error as a toast[29] message to the user.
5. Code logic handles the internal changes needed to handle the error (e.g. stop loading screen without doing anything).

5 Technical Manual

The technical manual section describes actions to deploy, edit and manage the website's code base and operation.

5.1 System Requirements

TorsDrive has been built using Node.js, but it requires some system-level interactions. TorsDrive software uses Bourne Shell, or Bash, built into most UNIX-like distributions. As such, the preferred OS are Linux distributions; TorsDrive has been developed under *Fedora 26* and hosted on a machine running *Ubuntu 16.04 LTS*. Ubuntu is the preferred host OS. Other requirements include:

[29] *http:// materializecss.com/ dialogs.html*

- **MySQL server** with a preconfigured database and user. This can be swapped to any other database as long as Sequelize connector is configured properly. Look for file "db.js" in TorsDrive's root directory.
- **MATLAB**, which can be called using "matlab" command from Bash. Bioinformatics toolbox[30] must also be installed.
- **Node.js** and **NPM**.
- **Nginx**, **Apache2** or some other web server for TorsDrive to run behind. This is optional, but recommended.

5.2 Project Folder Structure

Figure 13 on page 45 displays the structure of project's root directory. File and directories listed are:

1. **torsdrive** is the root directory's name.
2. **app.js** is the main driver file of the project. Contains session initialisation data for preserving login sessions, IP restrictions, and various module imports. If you run a server instance autonomously – not behind Nginx or Apache2, – you would need to change the piece of code which restricts access to the local machine.
3. **auth** directory contains files which define login strategies and password handling. This is where credential checking and password hashing happens.
4. **bin** directory contains start up scripts for the server. Files within also control SCSS-building commands.
5. **db.js** is the file that defines and creates the database schema. It also contains credentials for database connections, so protect this file with ownership rights: this file should be executable only for anyone but the owner. Beware that changing the schema will drop all tables.

[30] *https://mathworks.com/help/bioinfo/*

6. **func.js** contains support functions for all of the other files in the project. Functionality like checking object types and other common operations are kept in this file.
7. **license.txt** MIT license[31] text.
8. **logs** directory contains log files.
9. **matlab** directory contains MATLAB scripts that are used in operation of TorsDrive.
10. **node_modules** directory contains Node.js-specific files. You do not need to touch it.
11. **nodemon.json** is used in development to conveniently restart the server on any file change.
12. **package.json** contains NPM dependencies to be installed when deploying the server.
13. **pdb** contains temporary files created by MATLAB. This directory is cleared out after each server restart. Do not use it.
14. **public** contains publicly accessible files, such as static HTML pages, CSS, JavaScript, and language files. Do not put any dynamic pages or sensitive data in it.
15. **routes** contains files that operate all of the API and page access routes. This directory contains the most vital files for the server to work. Access it to see all of the APIs available. There is no API listing as of now.
16. **scss** contains SCSS files compiled into CSS. Do not touch main.css file in the **public** directory, and edit this one instead. SCSS is built on each server restart, overwriting main.css file in the **public** directory.
17. **strings.js** file contains strings and convenient shortcut functions for creating strings, like Bash commands to start up MATLAB.

[31] *https://opensource.org/licenses/MIT*

18. **torsdrive.sh** is a Bash file to start up TorsDrive. The default port is 3000, and can be set in **app.js**.

19. **views** contains page templates that are constructed using EJS[32]. Also contains partial pages that can be embedded within other pages for increased modularity.

```
torsdrive
├── app.js
├── auth
├── bin
├── db.js
├── func.js
├── license.txt
├── logs
├── matlab
├── node_modules
├── nodemon.json
├── package.json
├── pdb
├── public
├── routes
├── scss
├── strings.js
├── torsdrive.sh
└── views
```

Figure 13: Top-level file structure of the project's root directory.

5.3 Deployment

This subsection explains sequence of actions needed to deploy TorsDrive on a server. It implies that a zipped file called *torsdrive.zip* has already been sent to the target server and is readily available. Also, it is assumed that you connect to the server using a SSH-capable terminal software, like a built-in terminal on a UNIX-like system or PuTTY on Windows.

Lines starting with $ (dollar sign) are Bash commands to be entered into Bash prompt. TorsDrive is currently hosted at `bioinfdev1.cmp.uea.ac.uk` *and can only be accessed on UEA internal network or via UEA VPN[33].*

1. SSH into the server and unarchive *torsdrive.zip* into your working directory.
   ```
   $ unzip ./torsdrive.zip
   ```

[32] *http:// ejs.co/*

[33] *https:// vpn.uea.ac.uk/*

2. Descend into *torsdrive* directory and install package dependencies.
   ```
   $ cd ./torsdrive && npm install
   ```

3. If it does not exist, create a *pdb* directory in the *torsdrive* root directory.
   ```
   $ mkdir ./pdb
   ```

4. Edit the *db.js* file using any text editor (terminal-based editor **vim** is used in this case).
   ```
   $ vim ./db.js
   ```

5. Change the following lines (lines marked with *//* are comments, while text in **bold** is what needs to be modified):
   ```
   var database = "NAME_OF_DATABASE"; // Probably "torsdrive".
   var username = "DB_USER"; // Probably "torsdrive_admin".
   var password = "DB_USER_PASSWORD";
   var host = "localhost"; // Leave as-is.  127.0.0.1 should
   also work.
   var dbType = "mysql"; // Refer to Sequelize documentation
   to see what needs to be changed to swap the database.  If
   using MySQL, leave as-is.
   ```

6. Run the tool to create the needed database tables, and turn it off as soon as it has finished (*Ctrl* + *C* keystroke).
   ```
   $ ./torsdrive.sh
   ```
   If the file is not executable, run the following command before trying again:
   ```
   $ chmod +x ./torsdrive.sh
   ```

7. Log into the database (MySQL used in this example) and add the “user” and “admin” roles into the *roles* table. This example uses *torsdrive* as the name of the database, and *torsdrive_admin* as username.
   ```
   $ mysql -u torsrive_admin -p
   ```
   Now type in the password.

8. Type in the following SQL code. Each > sign corresponds to a single line of code.
   ```
   > USE torsdrive;
   ```

```
> INSERT INTO roles (name) VALUES ('admin'), ('user');
> exit;
```

9. Run the server in background.
   `$ nohup ./torsdrive.sh &`

10. Disown the process so that it is not killed when you log out. First list the jobs, and find the UID of the torsdrive process.
    `$ jobs -l`
    Now disown it by UID (e.g. `disown 1`)
    `$ disown && disown UID`

11. Next you need to set up a server in front of TorsDrive. This example shows how to do it for Nginx. Copy the configuration over into the virtual server file (in `sites-enabled`). For example (using vim):
    `$ vim /etc/nginx/sites-enabled/torsdrive`
    Copy the following into the file replacing all of its previous contents (comments marked with #):

```
server {
    listen 80;
    server_name torsdrive.cmp.uea.ac.uk; # That might differ.

    location / {
        # Localhost on port 3000. Adjust as needed.
        proxy_pass http://127.0.0.1:3000;
        proxy_redirect off;
    }
}
```

12. Restart Nginx (you might have been given a different command to do so).
    `$ sudo /usr/local/sbin/torsdrive-nginx`

13. Go to the website (e.g. torsdrive.cmp.uea.ac.uk) and register as a user. Log in into the database again. You will need to give your new user administrator rights.

```
$ mysql -u torsdrive_admin -p
```
Now type in the password.

14. Enter the following SQL code to determine your user ID. Remember it:
```
> SELECT * FROM users WHERE email LIKE
'%SOME_BIT_OF_YOUR_EMAIL%';
```

15. Run the following command on your *user_id* and *role_id*. If your current role ID (see output of command above) is 0, you need to set it to 1. Otherwise if it's 1, set it to 0 (as long as it matches "admin" entry).
```
> UPDATE users
> SET role_id = ROLE_ID
> WHERE id = USER_ID;
```

16. Copy the server's insides (the torsdrive directory) into a dedicated one (`/var/www` in this case):
```
$ cp -r ./torsdrive /var/www
```

17. Add an entry into *crontab* to restart the server automatically on each server reboot:
```
$ crontab -e
```
Add the line (depending on location of your torsdrive.sh file):
```
@reboot /var/www/torsdrive/torsdrive.sh
```

18. Finally, change permissions for *app.js* and *db.js* files so that only your user can actually read and write their insides:
```
$ chmod 711 /var/www/torsdrive/app.js
/var/www/torsdrive/db.js
```

You should now be able to go to `/admin` route of the website to manage other users. Managing other parts of the database would require logging into the database and doing it manually.

5.4 Editing the Tutorial

The tutorial files are located in `torsdrive/public/partials`. You can create tutorial files with designated ISO-639-1 endings for each language that you introduce. In case you want to edit the current English tutorial, you would access the file `torsdrive/public/partials/tutorial_en.html` and edit it's HTML source.

5.5 Language Management

Language files are located in sub-directory `torsdrive/public/lang`, and are to be created in format `{ISO-639-1}.json`, e.g. `en.json`. Each consequent JSON file is just a translated copy of the previous one. For example, the original English for the *languages* object is as follows:

```
"languages": {
  "en": "English",
  "jp": "Japanese",
  "ru": "Russian"
}
```

However, an Estonian version in `et.json` would look like this (note that names of object keys stay the same):

```
"languages": {
  "en": "inglise",
  "jp": "jaapani",
  "ru": "vene",
  "et": "eesti"
}
```

The process of adding a new language is as follows:

1. Create the language's JSON in `torsdrive/public/lang` using English JSON as template.

2. Open the file `torsdrive/routes/index.js` and add the language's code to the `availableLanguages` object.

6 Discussion

The system I have implemented has a number of flaws that have not been addressed.

Firstly, there is no database replication set up. This might lead to data loss as a result of server hardware failure or software corruption.

Secondly, password protection and configuration file separation are almost non-existent. Both the configuration and passwords are stored together with code that uses them, and the best solution I have had so far is to restrict access to them using Linux's user management primitives. Even though this is enough to keep them reasonably secure, it is still not the best practice.

Moreover, there are some inefficiencies while loading languages, PDB structures and other asynchronously fetched files using AngularJS. There are also inefficiencies when moving from one UI stage to another, which might lead to sluggish performance on slower systems.

Additionally, there is a number of inconsistencies and bad practices in the source code. The front-end JavaScript files are not properly separated, with AngularJS controller still hiding or showing some of the elements, while others are manipulated using a separate *view_functions.js* file. In the back-end, API router code is not completely separated from business logic code, so there are a number of functions withing *index.js* route file. Also, some of the comments are insufficient for or missing from random blocks of code in some JavaScript files: both in front-end and back-end.

Finally, MATLAB manipulation using Bash might not always work correctly, as MATLAB is prone to leaving an interactive prompt open, thus locking the process, if some particular error has happened. Malformed input, for example.

All of this leaves room for potential improvement of the system.

7 Conclusion

I have implemented a website with requested functionality to perform a single task of allowing computational biologists to streamline the process of protein homology modelling. The website is hosted on a web server and is live as of mid April 2017. Users of the hosted tool are able to select a protein by either providing a PDB code or their own PDB file, and target or constrain torsions of a loop segment of these proteins. They are also able to share their transforms, or hide them when needed, and view the results of torsion targeting.

The tool performs as expected, except for a set of bugs which are inevitably present in any nontrivial system. A set of instructions provided in this report enable a person with some technical background to deploy and modify the produced website.

This exercise allowed me to not only improve my technical skills, but also to better understand and appreciate another branch of science and the value of close cooperation with others. My attention to detail also stood up to a challenge, as might be seen in the provided set of more technical examples in section 4.1. Experience acquired in the run of this project is to allow me to tackle more complex problems in the future, and possibly share my expertise with others.

With this, I conclude this report.

References

abhishek (2015). 10 best node.js frameworks for developers. *https://www.devsaran.com/blog/10-best-nodejs-frameworks-developers* [Online; accessed 04-November-2016].

Aimonetti, M. (2013). What technology should my startup use? *https://matt.aimonetti.net/posts/2013/08/27/what-technology-should-my-startup-use/* [Online; accessed 04-November-2016].

AlternativeTo (2016). Alternatives to node.js. *http://alternativeto.net/software/node-js/* [Online; accessed 04-November-2016].

Arora, S. (2016). Node.js frameworks: The 10 best for web and apps development. *http://noeticforce.com/best-nodejs-frameworks-for-web-and-app-development* [Online; accessed 04-November-2016].

Bandt, T. (2015). Is node.js better than asp.net? *https://thomasbandt.com/is-nodejs-better-than-aspnet* [Online; accessed 04-November-2016].

Blundell, T., Carney, D., Gardner, S., Hayes, F., Howlin, B., Hubbard, T., Overington, J., Singh, D. A., Sibanda, B. L., and Sutcliffe, M. (1988). Knowledge-based protein modelling and design. *European Journal of Biochemistry*, 172:513–520.

Branden, C. and Tooze, J. (1999). *Introduction to Protein Structure*. Garland Publishing.

Canutescu, A. A. and Dunback, R. L. J. (2003). Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein Science*, 12:963–972.

Carr, S. and community (2016). What are the pros and cons of node.js+express vs ruby on rails? *https://www.quora.com/What-are-the-pros-and-cons-of-Node-js+Express-vs-Ruby-on-Rails* [Online; accessed 04-November-2016].

DBE (2016). Db-engines ranking. *http://db-engines.com/en/ranking* [Online; accessed 04-November-2016].

gawl, Marlow, S., Steward, D., and community (2010). What is the haskell response to node.js? *http://stackoverflow.com/questions/3847108/what-is-the-haskell-response-to-node-js* [Online; accessed 04-November-2016].

Glock, J. (n.d.). Node.js framework comparison: Express vs. koa vs. hapi. *https://www.airpair.com/node.js/posts/nodejs-framework-comparison-express-koa-hapi* [Online; accessed 04-November-2016].

Goldberg, Y. (2016). Checklist: Best practices of node.js error handling. *http://goldbergyoni.com/checklist-best-practices-of-node-js-error-handling/* [Online; accessed 04-November-2016].

Hayward, S. and Kitao, A. (2010). The effect of end constraints on protein loop kinematics. *Biophysical Journal*, 98:1976–1985.

Hayward, S. and Kitao, A. (2015). Monte carlo sampling with linear inverse kinematics for simulation of protein flexible regions. *Journal of Chemical Theory and Computation*, 11:3895–3905.

Ivanovs, A. (2016). Top 21 best free css3 frameworks for web development 2016. *https://colorlib.com/wp/free-css3-frameworks/* [Online; accessed 02-May-2017].

Ivanovs, A. (2017). Top 23 best free javascript frameworks for web developers 2017. *https://colorlib.com/wp/javascript-frameworks/* [Online; accessed 02-May-2017].

Kaplan-Moss, J. (2011). What are the benefits of developing in node.js versus python? *https://www.quora.com/What-are-the-benefits-of-developing-in-Node-js-versus-Python/answer/Jacob-Kaplan-Moss-1?srid=0Ivc* [Online; accessed 04-November-2016].

Kleveros, D. (2015). 600k concurrent websocket connections on aws using node.js. *https://www.jayway.com/2015/04/13/600k-concurrent-websocket-connections-on-aws-using-node-js/* [Online; accessed 04-November-2016].

Kolodny, R., Guibas, L., and Koehl, M. L. P. (2005). Inverse kinematics in biology: The protein loop closure problem. *The International Journal of Robotics Research*, 24(2-3):151–163.

Lesk, A. M. (2002). *Introduction to Bioinformatics*. Oxford University Press.

Mutlu, O., Yakarsonmez, S., Sariyer, E., Danis, O., Yuce-Dursun, B., Topuzogullari, M., Akbulut, E., and Turgut-Balik, D. (2016). Comprehensive structural analysis of the open and closed conformations of *theileriaannulata* enolase by molecular modelling and docking. *Computational Biology and Chemistry*, 64:134–144.

Noppen, J. (2014). 2m02, introduction & module schedule, software engineering i, week 1.

Peng, Z., He, J., and Niemi, A. J. (2015). Clustering and percolation of protein loop structures. *BMC Structural Biology*, 15(22).

Perrenoud, M. (2015). .net vs. mean: Migrating from microsoft to open source. *https://www.airpair.com/mean-stack/posts/developers-moving-dotnet-to-mean* [Online; accessed 04-November-2016].

Petrov, I. (2016). Comparison of asp.net and node.js for backend programming. *https://gist.github.com/ilyaigpetrov/f6df3e6f825ae1b5c7e2* [Online; accessed 04-November-2016].

RT (2016). Ruby web app frameworks. *https://www.ruby-toolbox.com/categories/web_app_frameworks* [Online; accessed 04-November-2016].

Vickery, G. (2016). After a year of using nodejs in production. *https://blog.geekforbrains.com/after-a-year-of-using-nodejs-in-production-78eecef1f65a#.hfgj8e99t* [Online; accessed 04-November-2016].

Wayner, P. (2015a). Java vs. node.js: An epic battle for developer mind share. *http://www.infoworld.com/article/2883328/java/java-vs-nodejs-an-epic-battle-for-developer-mindshare.html* [Online; accessed 04-November-2016].

Wayner, P. (2015b). Php vs. node.js: An epic battle for developer mind share. *http://www.infoworld.com/article/2866712/php/php-vs-node-js-an-epic-battle-for-developer-mind-share.html* [Online; accessed 04-November-2016].

Wikipedia (n.d.). Document-oriented database. *https://en.wikipedia.org/wiki/Document-oriented_database* [Online; accessed 04-November-2016].

Yang, C. (2016). Express, koa, meteor, sails.js: Four frameworks of the apocalypse. *https://www.toptal.com/nodejs/nodejs-frameworks-comparison* [Online; accessed 04-November-2016].