

Luke Amos

Registration number 100193308

2023

Loop Modelling Server

Supervised by Steven Hayward



University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

The project's aim was to create a website, for a user to submit jobs to a Linear Inverse Kinematic loop modelling method, given to me by the project supervisor, and visualize the results. The project uses NodeJS to host the API, and generated files related to the project. The generated files are viewable from a front end implemented as client side assets which is also hosted by the Backend.

Acknowledgements

I'd like to thank my advisor for the help and providing me with the algorithm this project show cases. The conversations we had have sparked my curiosity in the subject.

Contents

1	Introduction	5
2	Background	6
2.1	Proteins	6
2.2	Loops	8
2.3	Inverse kinematic approach	9
2.4	Other Loop modelling webserver	12
3	Methodology	13
3.1	Initial plan	13
3.2	Software	15
3.3	Backend	16
3.4	Frontend	18
4	Implementation	19
4.1	Sprint 1	20
4.2	Frontend objectives	20
4.3	Backend objectives	23
4.4	Mid Project Progress Reflection	24
4.5	Sprint 2	25
4.6	Frontend objectives	25
4.7	Backend objectives	29
5	Results	30
6	Conclusion	30
7	Evaluation	31
8	Future work	31
	References	32

List of Figures

1	Condensation reaction of amino acids	6
2	Isomers of amnio acid (right L, left D)	7
3	Protein backbone	10
4	High level overview	15
5	Endpoint 1 JSON Payload	33
6	Endpoint 1 Activity diagram	34
7	Endpoint 2 Activity diagram	35
8	Endpoint 3 Activity diagram	36
9	Endpoint 1 Sequence diagram	36
10	Endpoint 2 and 3 Sequence diagram	37
11	screen shot of submit job page	38
12	screen shot of view structure wide page	39
13	screen shot of view structure narrow page	40
14	Use Case Diagram	41
15	Code that invokes MATLAB	41
16	Code that generates standard file name	42
17	example command line on a windows system	43
18	submit job input key	44
19	view_structure input key	45
20	Revised project Gantt chart	46

1 Introduction

The objective of this project is to develop a web server that provides a graphical user interface for the linear inverse kinematic loop modelling algorithm provided as a MATLAB program. The primary purpose of the website is to facilitate easy testing of different input combinations and allow users to view the corresponding results.

The algorithm takes as input: the four character PDB code; an output file name; the chain identifier; start and end amino acid index; two matrices where each row comprises of an amino acid index and a target angle; two vectors containing the indexes of amino acids where the torsion angles should be constrained; and finally the number of iterations to perform. These inputs need to be collected from the user from the web page and submitted on the server.

A potential solution for this, would be to use the browser's fetch API to POST these inputs to a HTTP endpoint as a JSON object. This would require a web server running a POST endpoint, that checks the submitted object for invalid inputs, then checks that a file corresponding to that set of inputs already exists. Finally, if the file doesn't already exist, it needs to call MATLAB with the set of inputs, and, if the file was generated successfully, provide the link to the file on the server back to the client side code. It also needs to present an endpoint where the user can submit a query and the server will respond with PDB files that exist and match the query. This is so the client side code can load in the generated files to JSmol applets on page.

The client side can be split into code that handles the submission of jobs to the server, and code that handles viewing of PDB files stored on the server. The `submit_job` page can use the submit endpoint in order to create new processed files from a set of inputs. There should be feedback to the user indicating whether the file was created successfully or not. The `View_structure` page provides a search feature to find processed files in the database and load them into an applet for viewing.

To validate the inputs, the server should only accept JSON formatted data, or query string parameters, depending on the endpoint. The JSON object should be checked against a schema to ensure all the fields are of the correct type and all required fields are

present. Finally, shell escape chars should be removed to prevent script injection before user input is transformed into the corresponding command line string and executed in NodeJS/MATLAB. If a file was not successfully created a record should not be inserted into the database for it. When a file is generated by MATLAB, it should be moved to a statically hosted directory, and its details added to the database.

The client side code will contain a form for submitting jobs, which checks a user's inputs and gives some feedback about whether they are valid. The submit job page will also display a message to the user if the file has already been created. The view_structure page will rely on the query endpoint in order to complete a user's search of the database, and facilitate loading the hosted PDB files into JSmol. It should also provide controls linked to JSmol scripts for interacting with the structures loaded in the JSmol applet.

2 Background

2.1 Proteins

Proteins are essential to all biological processes, and are often termed the workhorses of biology. For this reason they are the subject of much research. For example, a functional understanding of proteins is essential for understanding the pathology of disease.

Proteins are polymers, formed by the condensation reaction of amino acid monomers forming peptide bonds between them to form a backbone. see Fig1. Amino acids are organic molecules in which a carbon atom, termed the alpha carbon, is bonded to an amine group (-NH₂), a carboxylic acid group (-COOH), a Hydrogen atom and an R group.

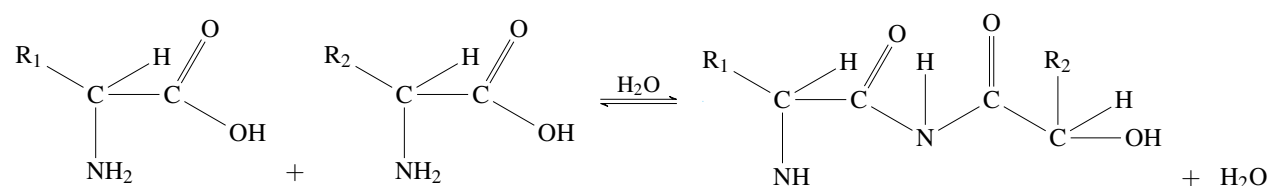


Figure 1: Condensation reaction of amino acids

The structure of the R group determines the type of amino acid, for example Alanine has a methyl R group. The interactions of the R groups, in standard conditions, result in the protein folding into a specific shape that confers its function. The R group of a particular amino acid will interact with the R group of the preceding and successive amino acids in a way that can introduce a bend in the backbone. The cumulative effect of these interactions, combined with hydrogen bonding, disulphide bridge and solvent interactions, result in a specific structure appearing out of an intractably large number of possible folded conformations. This typically happens in a time scale of microseconds.

The alpha carbon of a amino acid is chiral meaning it can have two non-superimposeable mirror images, levo (L) or dextro (D). Fig 2. The number of possible enantiomers of a molecule containing n chiral centers is 2^n . With the exception of a few bacteria, all organisms use the L isomer of the amino acids in their proteins. Additionally the vast majority of organisms metabolise the D version of chiral carbohydrate molecules. Levo and Dextro get their names from the Latin for left and right, and refer to the direction the enantiomer rotates plane polarized light.

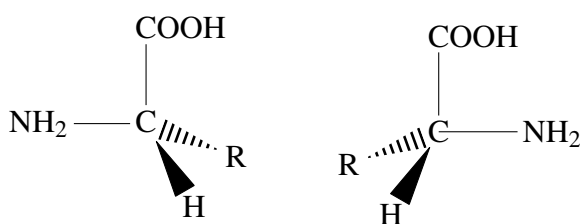


Figure 2: Isomers of amino acid (right L, left D)

Proteins have a complex free energy landscape, due to "their rich typology and the nature of their non-bonded interactions". The free energy landscape of a protein can be thought of as a "rugged funnel" that approaches the conformational energy of the native fold. (Tavernelli et al.) Proteins that misfold occupy a local energy minima in the free energy landscape. Misfolded proteins have been linked to neurodegenerative disease such as Parkinson's and Alzheimer's.

Information about protein structures are stored in PDB files. The PDB file contains a number of ATOM records, that specify the 3d atom coordinates of an atom in a protein. The file also contains information about how the results were determined (e.g

experimentally or analytically) as well as the primary structure. Regions denoting the type of secondary structure relating to different regions of the protein are also included. The atom and residue numbers are indexed. The PDB file can also contain a series of MODEL, ENDMDL directives surrounding ATOM directives, this allows structures to be animated. In the context of proteins, RMSD, or root mean squared deviation, is a measure used to quantify how close a given structure is to another structure. It is used to assess the accuracy of a predicted structure by taking the root mean square of all the position components of atoms in one structure, subtracted from all the position components of the atoms in another.

2.2 Loops

Loops are an important area of study as loops are often the drivers of protein interactions and function. A loop is an irregular region of a protein backbone, that is often terminated by secondary structural regions, such as alpha helices or beta pelted sheets. Many different approaches have been used to make predictions about loop structure from their sequence. These can be broadly divided into three groups: ab initio; comparative; and hybrid modelling. Such techniques lack accuracy across a broad range of protein structures and with increased loop length. (Choi et al.) This is because loops often have no discernible pattern in their sequence, unlike alpha helices or beta pelted sheets, and because increases in loop length leads to an exponential increase in the number of potential configurations.

AlphaFold is a program, which uses a machine learning approach to tackle the problem of determining structure from amino acid sequence. It has been shown it can predict the structure of loop regions with sub angstrom accuracy. This has had a impact on the loop modelling community, as AlphaFold out performs other methods across a wide range of proteins in predicting loop structures (Alquraishi). Despite this, techniques invented and adopted in the field will still be useful for a wide range of other tasks, as well as providing useful intuitions about protein dynamics and the functions of particular loops in proteins. This has potential for drug design as drugs often target flexible loops in proteins.

The protein backbone consists of "repeating $-N-C_{\alpha}-C-$ units" (Noonan et al.). All the bonded atoms connecting a given carbon alpha to the next carbon alpha, can be considered a "peptide unit", "The bonds of the peptide units are generally considered to be planar and of fixed length" (Noonan et al.).

Torsion angles are the twisting angles the plane of each peptide unit makes at a carbon alpha atom with the next peptide unit. Unlike the bonds angles in the plane of the peptide bond, the torsion angles "exhibit large variation within proteins"(Noonan et al.). The torsion angle around the $N-C_{\alpha}$ atom is called phi and the torsion angle around $C_{\alpha}-C$ is called psi. Some combinations of phi and psi angles are infeasible while others are characteristic of common structures. These regions can be visualised by plotting the psi angle by the phi angle in a Ramachandran plot. Changes in torsion angle can have large effects in the displacement of other atoms as "torsion angle moves result in movement proportional to the distance of each atom from the perturbation axes" (Coutsias et al.) This makes torsion angles a good candidate for a degree of freedom that contributes to loop structure/motion.

Loops have been an area of focus for the homology modelling community, as the sequence of loop regions is conserved and often unique, making it hard to find similar regions in proteins from the same evolutionary lineage. The secondary structural regions are more forgiving to mutation, so their sequences can display a greater degree of variation, while changes in loop regions are more likely to effect the protein's function. For this reason loop regions display less variation in their sequences across homologous proteins, and so are called conserved. This is why "overall accuracy of homology models tends to be considerably lower in loop regions"(Karami et al.) This suggests that natural selection in proteins is driven in large part by mutations to loop regions.

2.3 Inverse kinematic approach

Inverse kinematic approaches, model loops as rigid sections, connected by rotateable joints, without regards to forces involved. Inverse kinematics provides a means to model the geometric properties of a protein backbone. Two mathematical models are introduced in (Go⁺ and Scheraga), that can be used, to either find a set of dihedral angles that close a chain molecule, or to perform "local deformation" for a section of a chain molecule. The first treats bond lengths, bond angles and dihedral angles, or the Carte-

sian coordinates of atoms as variables. The second focuses on only the dihedral angles within the chain. While method one is more realistic, method two is less computationally expensive, due to the reduced number of independent variables.

The mathematical formulation consists of defining a set of local right hand coordinate systems, one for each atom, where the coordinate system of the i th atom is determined from the coordinate system of the $i-1$ th atom's local coordinate system. An equation relating the $(i-1)$ th atom's position vector to the i th position vector and $(i-1)$ th Cartesian coordinate vector is found. "The equations for transformation of coordinates among these $(n + 1)$ local coordinate systems" (Go[−] and Scheraga) are given by repetitive application of this equation. These equations, in combination with an equation that relates the 0th coordinate systems position vector to the n th coordinate systems position vector, cannot be satisfied for n torsions which corresponds to the fact that "deformation cannot be confined to a local region". (Go[−] and Scheraga) To solve this a system of equations is found that interrelate the n torsion angles. The result is a set of equations that can be used to determine the 6 dihedral angles based on the given values for the other torsion angles in the chain such that the deformation is confined. This "nonlinear method finds all possible bridging loop conformations" (6) The paper (Go[−] and Scheraga) also treats the case where the chain is a poly-peptide.

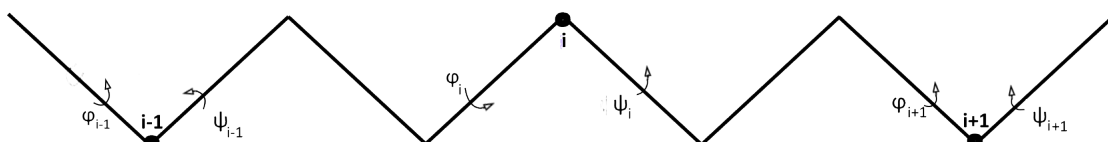


Figure 3: Protein backbone

The method presented in (5) uses a conceptually similar model to the one formulated in (Go[−] and Scheraga) to investigate the effect of constraining end groups on the motion of a functional loop in horse liver alcohol dehydrogenase (LADH) and lactate dehydrogenase (LADHase).

In LADH, the loop comprising residues 290-300, "acts as a blocker to domain closure in the absence of NAD⁺, " (5) due to contact between the loops Pro296 residue and His51, Thr56, Leu57. The author propose that the Val294 sidechain rotates to contact

NAD⁺ resulting in changes of torsion angles along the chain that move Pro296 out of the way. The loop has a "Pro-Pro motif" on (residue 295 and 296), because "proline will inhibit not only rotation about its own phi torsion axis but also rotation about the phi torsion axis of the preceding residue" The torsion angles psi294, phi295, psi295 and phi296, are fixed and create a "rigid arm". Starting from the open structure of the loop in 1ADG, constraining angles psi294, phi295, psi295 and phi296 and targeting phi291, psi291, phi292, psi292, phi293, psi293, psi294 to their values in the closed loop structure 20HX, and extending the segment to include residue 301, they were able to model conformations from the open to the closed loop conformation. It was then investigated whether the Pro-Pro motif is needed for the loop to act as a block to domain closure. This was done by selectively constraining the torsion angles of residue 295,296 effectively simulating Pro-NonPro, NonPro-NonPro and NonPro-Pro mutants. They show the algorithm was unable to reach the closed conformation without the Pro-Pro motif, supporting evidence that the Pro-Pro motif is instrumental in the loop's ability to act as a "NAD⁺ activated switch for domain closure."

In LADHase, a loop consisting of residues 98-110, undergoes conformation change when NAD binds to the coenzyme binding domain. The Open structure of the loop is given in 6LDH, while the closed structure is given in 1LDM. Residues 97,98 and 110,111 are hinge bending residues. It was shown that the algorithm progressively targeted the phi psi angles to those of the target loop until the targeted phi psi angles were unable to reach their target values. This process reached an RMSD of 1.5 between the experimentally generated structure and the observed structure.

In (6) a method is introduced, that uses a linear inverse kinematic technique to simulate loop closure and investigate whether, "a functional loop movement requires conformational change in the rest of the molecule", and whether "domains be conceptually viewed as rigid." Researches in Robotics have shown that the non linear approach, developed in (Go⁻ and Scheraga) "amounts to solving a polynomial of degree 16" where each root corresponds to a "non-interconvertible conformation of the loop". The linear method used in this paper uses "the determination of the null space or inverse of a Jacobian". It works by "small incremental changes in internal variables of the loop" In addition to this, bond angle variation of the backbone, is also performed during closure as "it's inclusion is important for bridging short gaps in proteins." It first defines a "linear

approximation of the rotation and displacement of a coordinate system at the end of a segment relative to the coordinate system at the beginning of the segment". First the Nth atom's displacement and rotation is expressed as a linear combination of the displacement and rotation of the atoms preceding it. Next a matrix equation involving a rotation matrix and displacement vector allow for converting the Cartesian coordinates into the internal local coordinate system. One equation is in the dihedral angle tau, and relates Cartesian coordinates of a point to the unit vector of rotation for the ith dihedral angle in a one to one mapping. A similar process is done for the bond angles theta. A Jacobian is set up that allows finding simultaneous solutions to loop and proline ring closure. This is done each time Monte Carlo is used to sample a random point in (PHI,PSI,OMEGA) angle space. The linear approach introduces a drift in the end groups. This drift is accounted for by resetting the position of the end groups, if they move from their initial position by more than a threshold value. (6)

This method is then used to investigate the loop (167-179) in 5TIM. It was found that "Pro168 has an unusually flat conformation in the closed structure" and is highly contorted in the open structure. The "relaxation" of this flat proline in the closed loop "drives loop opening". Through simulation they were able to show that the "relaxation of strain in the Pro168 ring" is not a factor that drives the loop from the open loop into the closed conformation. (6)

2.4 Other Loop modelling webserver

Web servers are often created for loop and protein modelling. They are also used to aid structural biologists and students in the visualisation of structures. And to perform functions on the protein structures, such as calculating energy or RMSD. For example, the "RCD+ fast loop modelling server" (López-Blanco et al.) uses an ab initio technique, that optimises randomly selected bonds. The combinatorial space of bond angles, lengths, etc can be reduced using a "geometric filter" that prevents "clashes between loop backbone atoms and the local protein surroundings" In this way it is able to generate "ensembles of closed loops". The generated loops are scored using a knowledge based energy function called "ICOSA" . They present a web service for submitting regions of a protein to model the loop for and for viewing the results in a JSmol applet. If the PDB for a native loop structure is provided, the RMSD is also displayed for each

loop on the results page. (López-Blanco et al.)

The "FALC-Loop web server" uses an *ab initio* technique "Fragment assembly and analytical loop closure" in combination with a database of fragment structures, where fragments are poly-peptides, 5 or 7 residues in length. The use of "fragment assembly" cuts down the combinatorial space while a "knowledge based potential allows fast scoring of generated conformers." The fragments are chosen based on "sequence identity" and assembled randomly. Then the analytical loop closure algorithm is used to "fit the candidate structure" by "rotating the six backbone torsion of randomly chosen three residues." The results page contains JSmol applet for viewing the structures of the generated conformers as well as DFIRE and RMSD scores. (Ko et al.)

The "DaReUS-Loop" web server is a loop modelling web server for modelling multiple loops in homology models. The loops are modeled using a database based approach, and loop candidates are ranked using the GROMACS energy function. The server takes as input either a "gapped" PDB file (where gapped refers to the fact atoms comprising loop regions to be modelled have been removed) and a sequence in FASTA format, or a gapped FASTA sequence with a non gapped protein structure provided in PDB file. The algorithm models the gapped loops simultaneously. With these inputs, it can perform either Modelling, where a gapped PDB and full sequence is used to complete the loops, or Remodelling where the server takes a homology model and a gapped sequence, and remodels loops based upon the gaps in the sequence. Advance Modelling uses the same inputs as Modelling but models the loops independently, which improves accuracy. A limitation of this server is that it does not provide a ranking for the generated final protein structures (Karami et al.)

3 Methodology

3.1 Initial plan

The application uses a client server model and is written using an iterative development paradigm, with use case testing during and at the end of each sprint. The Frontend client is written in HTML, CSS and Javascript, while the Backend is written in Javascript

(NodeJS). The Frontend code is responsible for validating user input, submitting the inputs to the server over a HTTP POST endpoint, allowing the user to search the database, and facilitate visualisation of the results in JSmol. The Backend code is responsible for validating the job requests it receives, checking if the job has been submitted previously, processing the input with MATLAB and hosting the resulting file so the client side code can use it. MongoDB, MATLAB and NodeJS were installed on the server. Figure 4 shows a plan for the architecture. The Backend code needs three endpoints to achieve this. Submit_job uses a POST request to endpoint 1 in a JSON body format, an example request is shown in appendix fig 5. Endpoint 2 is used to retrieve a list of processed files for a given PDB id. endpoint 3 is used to check if a given file exists on the server and returns a link to download it if it does.

The table below shows the endpoints used.

Endpoint	method	path	payload	function
1	POST	/api/v1/pdbs/	JSON	Produces PDB files from payload
2	GET	/api/v1/pdbs/?query	Query string	search for pdb files
3	POST	/api/v1/pdbs/1	JSON	Check if PDB file exists

The string following the '?' in endpoint 2 is a query string. It consists key=value pairs, separated by symbols. A GET to this endpoint where the PDB ID query parameter is set, returns an array of JSON objects representing PDB files that correspond to the PDB ID given in the query string. It should return an empty list if no such PDB files exist. The query parameters supported are PDB ID, segbeg and segend.

This diagram shows the simplest configuration. However, in deployment other software may be in use, such as Nginx acting as a reverse proxy, certbot for providing SSL or fail2ban for rate limiting. The deployment was done using a cloud provider hosting a VM instance. The instance needs a minimum of 8GB of RAM to meet the minimum requirements of MongoDB and Matlab. Nginx acts as a buffer between the NodeJS applications and incoming requests, this improves the security of the NodeJS application. The NodeJS application needs to function on Windows and Linux system where MongoDB and MATLAB are installed.

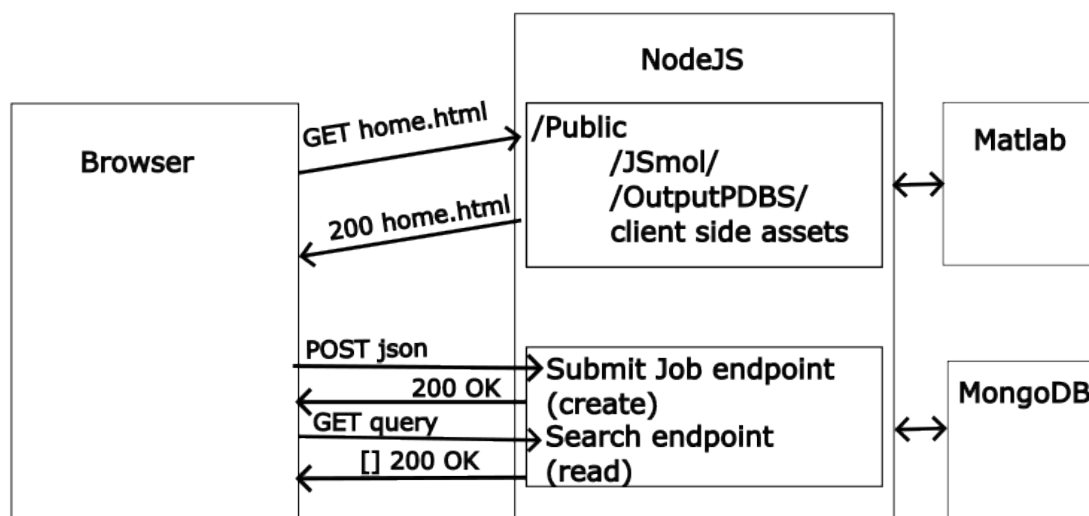


Figure 4: High level overview

3.2 Software

MongoDB was chosen because it's the default used by the boilerplate generator.¹ MongoDB differs from SQL databases in that it uses a collection of documents, where each document represents an object, while a SQL database stores data in tables where a row represents an object. You can search the collection by specifying a pattern in the document that matches the desired subset of documents. MongoDB is used to store information related to successfully created PDB files. Because no use cases require information relating the different input parameters / generated files / document objects together, a non-relational database is a good fit, as it removes the need for designing a table to store the information. Instead, a JSON object representing a given set of inputs can be inserted directly into the DB as a document.²

MATLAB is used to process a given set of inputs and generate the output PDB file. The server needs to tolerate MATLAB failing to produce an output PDB, or crashing. The MATLAB coder toolbox allows exporting code as transpiled C or C++. The transpiled code can be compiled with the MATLAB SDK however, as this was not the aim of the project an installed version of MATLAB was used with a crafted command line

¹ <https://github.com/w3cj/create-express-API>

² <https://www.mongodb.com/>

string instead. To facilitate this a wrapper function was created in MATLAB. This allows the user to call the entry point MATLAB script as a function from the command line passing the inputs as command line arguments.³

Initially, PHP was used to host the JSmol library and client side files. This was because the JSmol documentation suggested that certain functionalities would only work when hosted on a PHP server. However, once the functionalities required for the project were confirmed to work (loading in and playing animated PDB files) on a NodeJS Backend, NodeJS was selected instead. NodeJS is a server side Javascript context, built on top of the v8 Javascript run time. It supports non-blocking IO meaning it can handle requests while also performing IO operations. There is a large ecosystem of libraries, documentation and boilerplate generators available for NodeJS.^{4 5}

The boilerplate generators set up the basic project structure and make sensible choices about what middlewares and defaults to use, by generating boilerplate code. Using them saves time and can improve security. I used the create-express-API boilerplate generator created by CJ from the coding garden.⁶

3.3 Backend

The Backend was initially generated using a boilerplate generator that produces boilerplate code for a CRUD API using express for the endpoints and MongoDB for the database. A CRUD API is a common API paradigm that provides endpoints for Creating, Reading, Updating and Deleting entries in a database. The command below was used to generate the CRUD API boilerplate using the express package for NodeJS.

```
npx create-express-API -d my-server
```

This Generates the following file structure

```
my-server/  
  src/
```

³ <https://uk.mathworks.com/products/matlab.html>

⁴ <https://nodejs.org/en>

⁵ <https://jmol.sourceforge.net/>

⁶ <https://www.youtube.com/watch?v=EzNcBhSv1Wo>


```
API/  
    index.js  
app.js  
index.js  
middleware.js  
.env  
package.JSON
```

The boilerplate can be run by changing to the 'my-server' directory and running `npm start` or changing to the 'my-server/src' directory and running `'node index.js'`. The 'package.JSON' file contains information about production and development dependencies and start scripts. Running `'npm -i'` in a directory with a 'package.JSON' creates a 'node_modules' folder which contains the dependency code. The .env file contains environment variables that are available to NodeJS through the process.env object.

The 'my-server/src/api' directory, stores Javascript files containing express endpoints. The 'my-server/src/api/index.js' file contains the code that mounts the API's specified in this directory. 'my-server/src/app.js' contains code related to setting up middlewares for the server and mounts the API Javascript files in the API directory onto the route '/api/v1'.

A middleware is a function that takes a request object, response object and a function to be called next. An example is the CORS middleware which sets HTTP headers to allow cross origin resource sharing. The middlewares in use by default are Morgan, Helmet, cors, and express.JSON. Morgan is a dev dependency and allows for coloured output in the server's shell, and error debug output on the endpoint, which is useful for testing. Helmet sets sensible defaults for HTTP headers to make the application more secure. Express.JSON, parses payload strings into JSON objects. 'my-server/src/index.js' is the entry point, it loads in app.js as a module and sets it listening on a port.

From this boilerplate, the express static middleware is used to host a directory containing the JSmol library and the client side documents. A file, 'pdbc.js', was added to the 'my-server/src/api' directory and contains code for the API endpoints for the project. 'pdbc.js' provides the three API endpoints that allow client side code to create

and read from the database. The directory, 'my-server/src/public', was added to hold the extracted JSmol library files. Browsing to

`http://localhost:5123/public/JSmol/simple.htm`

reveals that JSmol is successfully loaded from the server. JSmols functionality is demonstrated by the large collection of examples pages that the library comes with, they are also hosted at chemapps.stolaf.edu.⁷

The API endpoints won't be authenticated, meaning any application could make requests to the endpoint. Ideally the API would be authenticated, but it was not a preliminary goal of this project. In addition to this, the PDB files were statically hosted in the 'my-server/src/public/OutputPDBS/' directory. This directory could be scraped by automatic web scrapers and, if it became too large, could cause more expensive web hosting fees. To solve this API authentication could be used, but the effect can be mitigated using rate limiting. In addition there won't be any user accounts or user state, which simplifies the design of the Backend.

Backend Sprint 1 Aims An initial aim for the Backend was the creation of a POST route for endpoint 1. It needed to accept the submitted JSON object and parse its fields into the corresponding command line string to be executed.

3.4 Frontend

The Frontend code consists of two main pages, 'submit_job.html' and 'view_structure.html'. These files have corresponding CSS and JS files of the same name and are served statically from the 'my_server/src/public' directory. The JSmol library files are hosted in the same way. The HTML boilerplate for 'submit_job.html' and 'view_structure.html' is generated using emmet VS code extension.

8

The submit job page contains input elements that allow the user to submit the set of 10 inputs necessary for the MATLAB script. The page communicates with the server

⁷ <https://chemapps.stolaf.edu/jmol/docs/examples-11/simple.htm>

⁸ <https://code.visualstudio.com/docs/editor/emmet>

using the browser's fetch API, a successor to AJAX. The JSmol applet uses AJAX to communicate with RCSB and retrieve PDB files so CORS headers should be set. To do this the 'contentSecurityPolicy: false' property was set on the Helmet middleware. The page should also provide some way to view initial torsion angles for a given loop section so the user can make an informed choice for the values of target angles. The page should display a message if a user has input an invalid type for a particular input field. The page should also provide feedback about whether a job completed, is running, or resulted in an error. Ideally the page would only accept inputs which are valid for the algorithm to avoid MATLAB crashing or failing to produce an output, however this is not a necessity.

Frontend sprint 1 Aims An initial aim for the 'submit_job' tab would be to collect the users inputs and submit them in a POST request to endpoint 1.

The View structure page should contain the JSmol applet and a means to load one of the generated structures into the JSmol applet. There should also be a button to play the animation of the loaded structure. The view structure page will use the fetch API to query the server for a particular file. The page should make use of JSmol's scripting capabilities, in order to create widgets for visualising the structure, for example changing the display style.

Frontend sprint 1 Aims An initial aim for the 'view_structure' page would be a page with a JSmol applet and controls for playing the animated PDB file. Using the JSmol applets right click context menu it was confirmed that the animated PDB files, stored in the 'my_server/src/public/OutputPDBS' directory on the server, could be loaded in by URL.

4 Implementation

The development of the app happened in two stages. The first sprint focused on creating a minimum viable product by fulfilling the initial aims set out in the methodology section, by breaking them down into objectives. The second sprint focused on adding more features and improving the usability of the site. The testing was done alongside each sprint. After completing code for a use case, the use case was tested, and any bugs

fixed as they came up. The primary use case is to allow a user to submit jobs and view the output structure. Use cases are shown in a use case diagram in the appendix fig 14.

4.1 Sprint 1

Sprint 1 Focused on creating a minimum viable product, by fulfilling the initial goals set out in the methodology section in order to create the use cases shown in fig5. The following sub goals were set.

4.2 Frontend objectives

1. Load a JSmol applet into the page and display a protein structure. (view_structure).
2. Load a structure in from the server by URL and JSmol script or JSmol right click context menu. (view_structure).
3. Animation controls for the JSmol applet. (view_structure) .
4. Collect and validate a user's input. (submit_job).
5. Fetch the users input to endpoint 1 and print the result. (submit_job).

A view_structure.html and submit_job.html page was added to the 'my_server/src/public' directory containing a boilerplate HTML5 template. Client side assests were chosen in preference to server side rendering for simplicity. Naviagting to

`http://localhost:5123/public/submit_job.html`

`http://localhost:5123/public/view_structure.html`

confirms that the NodeJS server was correctly hosting the files in the 'my_server/src/public' directory statically. The JSmol library was extracted to the public directory. Navigating to the example pages in the JSmol library showed that JSmol was successfully loaded from the server. HTML Script tags referencing the JSmol library files 'JSmol.min.js', 'JSmolJME.js' and 'jsme.nocache.js' were added to the HTML head block of submit_job.html and view_structure.html pages as well as HTML Script tags to 'submit_job.js' and 'view_structure.js' respectively.

Objective one was achieved by creating a div tag in the body of the view_structure.html page. Code defined within the window.onload function of a client side web page will be run when the page loads. In this function a JSmol applet was created by defining a JSON object, 'JmolInfo', that contained the necessary parameters for JmolApplet's constructor which is available in the Javascript context after importing the library. This included a 'width' and 'height' property which set the initial dimensions of the applet. A 'use' property which was set to the value 'HTML5' which instructs the JSmol applet to use HTML5 for rendering the applet. The 'script' property was set to the string 'load=1adg'. The string passed, as the script property is run as a JSmol script on the JSmol applet when the JSmol applet loads. The HTML for the JSmol applet was created by calling the Jmol.getAppletHtml(name, info_object) with the first parameter as a string which names the JmolApplet and the second parameter is the JmolInfo JSON object. It returns a HTML string corresponding to the created JmolApplet which can then be mapped to the innerHTML property of the div added to the view_structure.html page. After calling Jmol.getAppletHtml, a reference to the created JSmol applet object is available in the client side Javascript scope under the name passed as the first argument to Jmol.getAppletHtml.

Objective two was completed by first placing an animated PDB file, 'LADH_Loop_Movement.pdb', into the server's 'my_server/src/public/' directory. Using the JSmol applets right click context menu, the PDB file was loaded from the server into the applet by URL. It was found that, to successfully load a PDB from the server, the URL must contain the www and http/https parts of the URL.

Objective three The JSmol scripting documentation shows the 'anim play' and 'anim off' JSmol scripts can play an animated PDB file. Using the web developer console, JSmol scripts can be run on a given JSmol object using the Jmol.script method of the JSmol library. For example the following code

```
JSmol.script(jmolappletname, 'anim play')
```

Starts the animation from the currently displayed Frame for the PDB loaded in the JmolApplet jmolappletname. HTML5 Buttons were then added to the body of the page that called functions which then in turn called 'anim play' / 'anim off' JSmol scripts on

JSmol applet respectively.⁹

Objective four A set of HTML input fields were added to the submit_job.html page. These included text fields for the PDB ID and chain. Numerical fields for the start and end of a loop segment and number of iterations to perform. The algorithm also requires a set of four arrays. The target phi and target psi arrays, contain arrays where each sub array contains 2 numbers, the index of the residue to target the angle to target too. The constrained phi and constrained psi array contain a list of amino acid indexes of residues to constrain the phi or psi angle of respectively. To collect these inputs, two number input fields and two buttons were added to the page, to collect the target phi, and the same for the target psi. To collect the constrained residue index's, one number input field and 2 buttons were added, for constrained phi and constrained psi inputs respectively. Four arrays were defined in the global scope of the 'submit_job.js' file. When the user enters a given residue index and torsion angle target, their can press the 'add' button, which appends an array containing the index and target to there respective global array. The contents of the array is then shown in 'a' Tag below. For the constrained residues, the indexes of residues are appended directly to the respective global Javascript array. Each set of inputs has an add and remove button, which adds a set of inputs to the global Javascript array, or removes the last item added respectively, and updates the view below. A validate function was written, that checks the required inputs are populated, of correct type, and that no targeted or constrained residues lie outside the region between loop start and loop end. It shows a message to the user showing which of these checks failed, if any.

Objective five The inputs are assembled into a JSON object, then the fetch API is use to POST the object to endpoint 1 on the server. An example of fetch is shown in the code snippet 6. The fetch returns a promise which can execute a different callback, based on whether the POST request was successful or not. The fetch command returns a promise. The .then() function of the promise object is then used to parse the servers response to a JSON object and then to access a property of the JSON object. The property will have a value of the PDB file name if the file was created successfully or false otherwise.

⁹<https://chemapps.stolaf.edu/jmol/docs/>

4.3 Backend objectives

To the pdbs.js file, a post endpoint on the path '/' , was created and the express.JSON middleware was used to parse incoming payloads strings to JSON objects. The post endpoint was set up to return any JSON object it was called with back to the calling code using the following line 'res.JSON(req.body)' res.JSON makes the response a JSON object, req.body is the payload of the request object. The request object comes from the browser which make the request, the response object determines the content and type of response made by the endpoint. The initial sub objectives for endpoint 1 were as follows.

1. Detect the OS of the server (for using the correct move command)
2. Parsing the req.body JSON object into a crafted command line string
3. Look in the database for a file corresponding to the req.body JSON object and return it's name if found.
4. If no relevant file was found in the DB it should generate the output PDB with the crafted command line string and move the resulting file to the OutputPDBS directory.
5. Insert a document corresponding to the generated file into MongoDB. Finally it should return the filename of the newly generated file or an error string if MATLAB crashed.

Objective one was achieved using the node environments 'process.platform' global which will be 'win32' on a Windows based system. If not on a Windows based system the server will assume the server is running on Linux. The only practical difference is the 'move' command on Windows is 'mv' on Linux which is used to move the file generated by MATLAB to the 'my_server/src/public/OutputPDBS/' directory.

Objective two The API was tested in Insomnia, an open source API development platform. Submitting a POST request to endpoint1 with a JSON object shows that the endpoints reflects any JSON object sent by the browser back to the browser.¹⁰

¹⁰<https://insomnia.rest/>

Objective three was achieved by using a function 'genStandardFileName' which takes a JSON object like that shown in fig5. From this is constructed a string for the file name based on the JSON objects fields (PDB id, chain, segment start, segment end, target phi angles, target psi angles, constrained phi angles , constrained psi angles, and iterations). It is constructed in such a way that two identical sets of inputs yield the same string. The JSON object's 'fname' field is then set to the generated standard file name. The Database is searched for any document where the 'fname' is the generated standard file name. If a corresponding document is found it returns the file name in a JSON object back to the client side code.

Objective four happens in the case when no document matching the generated standard file name from request objects fields was found in the MongoDB. The JSON objects fields are parsed into a command line string like that shown in appendix figure 17. This is then executed. If the file was not created successfully 'false' is returned and the request completes. The client side code receiving a JSON object where the only field value is false indicates that the server failed to create the file.

Objective five happens in the case the file was created successfully generated by MATLAB. First the file is copied to the 'my_server/src/public/OutputPDBS' directory then a document corresponding to the file is inserted into the MongoDB. Finally a response is made to the client side code consisting of a JSON object with the file name of the newly generated file.

4.4 Mid Project Progress Reflection

At this point current progress was assessed and use cases were tested. It was found that the post endpoint was vulnerable to a script injection attack. Setting the chain parameter to a particular string like the one shown in the JSON snippet below

```
"chain" : "';)\\" & notepad & "
```

could start a process on the server. To solve this the Joi object schema was designed around the valid fields for the JSON payload. The strict() option in the Joi schema prevents automatic type casting while the required() option ensures that field is present. This ensured that the necessary fields are present in the JSON payload and that they

have the correct type. This prevented the numerical and array typed fields being used for command injection. A function 'sanitizeInputString' which takes a string and returns a copy of the string with special chars removed was created. This function was used to create a function 'sanitizeInputObj' which applies 'sanitizeInputString' to each string field of the request object. The result is a object with fields that have had special chars removed. It is then safe to craft a command line string from this object. Two lofi sketches were created to guide how the site should look after adding CSS. A home.html page was added to the public directory. The home page contained a user guide for the submit_job and view_structure pages.

4.5 Sprint 2

Sprint 2 Focused on adding extra features and improving usability. The features added included, the ability to view torsion angle change in the loop animation, the ability to align the left and right applet views, highlight the loop segment in the protein structure, sync the loop and protein views and load in the alternative version of the PDB file. Navigation bar and CSS themes were also added, to improve the usability of the site.

4.6 Frontend objectives

1. Decide a layout for the view_structure.html and submit_job.html pages and add CSS to implement it.
2. A feature that allows the user to see a residues phi/psi angle in the initial configuration for the submit_job page.
3. A feature that allows the orientation of the left and right applets to be synced in the view_structure page.
4. A feature that highlights the loop currently shown in the Loop view applet in the protein view applet
5. A feature to change the display style, color and background color of the applet
6. A feature to show the header information for the protein loaded into the protein view.

7. A feature to show the torsion angle changes in the loop as the animation plays.
8. A feature to run a given JSmol scripts on a given applet
9. A feature to search for structures by PDB id and load them into the loop view applet.
10. A feature that loads an alternative version of the PDB file where the ATOM directives before the loop start and loop end aren't removed, so the motion of the loop region can be shown in surrounding protein structure

Objective One This was implemented by designing two layouts for the view_structure page, one for wide views and one for narrow views. (screen shots in appendix figure 12 and 13). A function was assigned to the 'window.onresize' handler to change between the narrow and wide views when the page width was less than 800px. To the view structure page was added two divs for the JSmol applets with two divs containing input elements that would control their respective JSmol applet. A fifth div that would contain controls effecting both JSmol applets was also included. These divs were arranged using inline CSS that defined the absolute positions, dimensions, border styles and background colours. Two JSmol applets were chosen so the user could view the loop structure next to the full structure. This was done to aid the user in imaging the interactions the loop might have with other parts of the protein structure close to the loop.

A Nav bar was added to the home, view_structre and submit_job page using the Bootstrap5 CSS framework. Two colour pallets were used to create themes. One came from the colorhunt website ¹¹ and the other is the solarised dark color pallet ¹² . A change theme Javascript function was added to the view_structure and submit_job page that applied these color pallets to various elements on the page. A dropdown menu was added to the nav-bar to enable theme selection.

To the submit_job page three divs were added. The first contained the PDB ID , chain , segment start, segment end input elements and buttons for loading and viewing initial torsion angles for the loop region. The second div was reserved space for loading a

¹¹ <https://colorhunt.co/palette/fbfacddebaceba94d17f669d>

¹² <https://ethanschoonover.com/solarized/>

JSmol applet used to calculate the initial torsion angles. The third div contains the elements necessary for collecting the rest of the inputs as well as a submit button, and a P tag to show the result of the job to the user. A Bootstrap5 spinner¹³ was also added to be set visible when the submit button is pressed and set hidden when the .then() execution context of the fetch call is reached.

Objective Two this was achieved by adding three buttons and, a number input field to the submit_job page. The 'show initial torsion's button loads the PDB ID specified by the first input into a JSmol applet in the second div of the page. It then prints the torsion angles between the Start and End amino acid index's on the page. The 'Get' button gets the Phi and Psi angle of the residue at the amino acid index provided in the preceding number input field and adds it to the page below. The Clear buttons clears this output.

Objective Three This was achieved by adding a checkbox 'sync' to the joint controls div of the view_structure page. When the checkbox is checked, a JSmol script is called on the protein view JSmol applet that selects all the JSmolApplets on the page, and sets the 'syncMouse' directive to true, otherwise sync is set off. To each of the individual control divs were added 'Jmol.JmolMenu' inputs which are defined in the window.onload function and appended to a control div's 'innerHTML' property. The Jmol inputs 'JmolMenu' and 'JmolBr' are used. Each returns a HTML string corresponding to the input element. Using JSmol's built in input elements results in less code than having vanilla HTML5 buttons that call JSmol scripts via the 'Jmol.Script' method directly. The JmolMenus added to each div contained six options (Front, Back, Left, Right, Up, Down) and use the 'moveTo' JSmol script to change the orientation. By setting each view to the same orientation the views can be synced, such that the orientation of the loop in the loop view applet is the same as the orientation of the loop in the protein view applet.

Objective four This was achieved by adding a checkbox 'highlight loop' to the joined controls div and a global variable 'current_loop' was defined that contains a string such as '290-310'. When a new loop is loaded a regex is used to extract the segment start and segment end and reassigns the current_loop variable appropriately. When the 'highlight loop' checkbox is checked the loop region held in the current_loop variable is selected

¹³ <https://getbootstrap.com/docs/5.0/getting-started/introduction/>

and coloured green. Additionally a 'JMolRadialGroup' was added to each of the individual controls that allow a user to set the background color of an applet to black or white.

Objective five To achieve objective five, two 'JMol.Menu' inputs were added to the loop view's control div and protein view's control div. The first changes the style of the current display, the options used are 'ball and stick', 'stick', 'cartoon' and 'van der walls'. The second changes the colors used by the JSmol applet, the options used are 'cpk', 'group', 'amino', 'structure' and 'chain'

Objective six this was implemented by adding a div to the view_structure page above the other elements which is initially hidden by CSS. When the 'show_header' button is pressed the header information is retrieved from the protein view's JSmol applet and set to the div's 'innerText' property. The divs display style is then set visible.

Objective seven This was implemented by adding a checkbox to the loop view controls div that, when checked, sets a global Javascript variable 'showing_labels' to true. In the 'window.onload' function a JSmol script 'set animFrameCallback "label_phi_psi"' is run on the loop view applet that sets a call back function to be called each time the frame changes. When showing_labels is true, the label_phi_psi function runs a JSmol script on the loop view applet, selects the alpha carbon atoms between the loop start and end, and labels them with the phi and psi angle. When showing_labels is False the atoms are labeled with nothing to hide the labels. This function is called from the JSmol applet as a call back every time the frame changes, so toggling the show_labels checkbox while the animation isn't playing won't do anything. In addition to this, further buttons to control the animation were added to the loop view controls. There are buttons to play once, pause, show the first frame, show the previous frame, show the next frame, show the last frame and play in a loop.

Objective Eight this was implemented by adding two checkboxes, a text input field, an output div and a button to the joint controls div. The checkboxes set which applet the JSmol script typed into the text input field should be run upon. They are labelled left and right in the wide view, and top and bottom in the narrow view. The history of what commands have been run is shown on the output div. The currently selected applet is

stored as a global Javascript variable. When the user hits the 'run cmd' button the text in the text input field is run on the selected applet using the Jmol.script method.

Objective Nine This was implemented by adding a text input field, two number inputs fields, a button and a output div to the loop view controls div. The text input field takes a PDB id and is a required parameter. The two numerical fields take the loop start and end index's and are optional. When the user clicks the search button the inputs are parsed into a query string that is sent to endpoint 2 in a GET request. This returns a JSON object array, where each object corresponds to a PDB file that exists on the server matching the user provided inputs for PDB ID , loop start and loop end. For each of these JSON objects a div and a tag is created. The 'a' tags 'innerText' property is set to a combination of the JSON objects parameters such that the user can identify the combination of parameters that were used to generate the file to which the a tag refers. The 'a' tags 'onclick' handler is set to a function 'check_exists' which queries endpoint 3 to determine the URL of a particular file on the server. Then the PDB file corresponding to the clicked a tag is loaded into the applet using the 'load' JSmol script and the URL of the PDB file.

Objective Ten This required modification of the Backend code that handles endpoint 1 so that it produces an alternative version of the PDB file where the name is appended with string 'ALT'. Pressing this button loads the ALT PDB file of the PDB file currently loaded in the loop view.

4.7 Backend objectives

1. An endpoint that allows a user to check whether a file exists
2. A function that can process a successfully created file in order to create the alternative PDB file.

Objective One This was implemented by adding a new endpoint that accepts a JSON object as payload with a single property 'fname' that is the file name of the file to check. It uses the NodeJS 'fs' module to check if the given file exists. If the file exists it returns a JSON object with a single parameter 'redirectStr' which is set to false if no file exists and set to the URL of the file if it does exist.

Objective Two This was implemented by reprocessing the animated PDB file and the original PDB file into a new PDB file, that contains the same models as the animated PDB file, but with the ATOM directives of residues before and after the loop inserted either side of the loop region. The result is a animated PDB file that displays the full structure but can be played and shows the loop animation within the protein's superstructure. The original PDB file is downloaded by PDB id from RCSB ¹⁴ to a temporary folder. The output file is generated, then copied to the 'OutputPDBs' directory with the string 'ALT' appended to the end of the file name. The file is generated, after the animated PDB is generated and before the reference to the generated file is inserted into the database.

5 Results

JSmol provides a wide feature set and scripting capabilities for the creation of interactive web pages that aid in the visualisation of protein molecules and other chemical systems. It is easy to use and there is a large amount of documentation and examples available on the web. In this project JSmol was used to provide a Frontend to a linear inverse kinematic loop modelling algorithm. JSmol is a good choice because its simple enough to be deployed easily and has a feature set which covers most common visualisation tasks. There is also a large amount of documentation available on the web. The side by side view ended up cluttering the page and increased the page load time while the load alt PDB feature made it redundant. A screen shot of the submit job page can be seen in figure 11, with a key for the inputs shown in figure 18 of the appendix. A screen shot of the wide and narrow views for the view structure page can be seen in figure 12 and 13, with a key for the inputs shown in figure 19 of the appendix.

6 Conclusion

The project fulfills the two overarching aims set out in the introduction. The submit_job page allows the user to submit a set of inputs. The server can then process these inputs into a file using MATLAB. The server also provides a means to search for and view the generated PDB files.

¹⁴<https://www.rcsb.org/>

7 Evaluation

The project fulfills the initial aims set out in the methodology to provides a means for the user to submit jobs to the server and view the resulting structures. The decision not to use authentication or user state simplified the development as it meant security was not a primary concern. The styling of the page is clunky and dated but reasonably straight forward to use. The server does little to make the process of picking good target angles for the algorithm easier. Hosting all the content statically simplifies the server code as it eliminates the need for server side rendering, however it limits the scalability of the site and potentially makes the client side code more complicated.

8 Future work

Future work could include fixing a number of bugs that were discovered during testing.

- The label ϕ ψ angle checkbox in the loop view controls defaults to checked when it should default to unchecked.
- The generated alternative PDB file contains artifacts for the first frame on the animation, most likely due to a bug in the function that produces the alternative file.
- The box showing search results on the view structure page contains sometimes shows NaN for the constr ϕ title

In addition there are some additional features and optimisations that would improve the site such as:

- Rewrite the API endpoint code in a more streamlined way by refactoring app logic into custom middlewares. This will make the code for the endpoints more readable and reuseable.
- Streamline the Javascript imports and 'window.onload' function for the view structure page, to reduce load times.
- Streamline the view_structre page, to use one JSmol applet. This is because the ability to view the structures overlayed and individually would eliminated the

need for the side by side view. This functionality was added in the 'load alt file' feature

- Find a better way to view initial torsion angles for a loop segment on the submit_job page. This would allow the user to make a better informed choice for the target values.
- Find a way to view other biological molecules, like co-enzymes on the loop view. facilitate superposing loop structure with that of coenzymes inserted into the view. This would aid in imagining how certain loops and co-enzymes may interact.
- Include a way to score the candidate loop structures conformational energy, to provide a way of determining which is most energetically probable.

References

- [Alquraishi] Alquraishi, M. AlphaFold at CASP13. 35(22):4862–4865. Number: 22.
- [Choi et al.] Choi, Y., Agarwal, S., and Deane, C. M. How long is a piece of loop? 1:e1.
- [Coutsias et al.] Coutsiass, E. A., Seok, C., Jacobson, M. P., and Dill, K. A. A kinematic view of loop closure. 25(4):510–528. Number: 4.
- [Go[−] and Scheraga] Go[−], N. and Scheraga, H. A. Ring closure and local conformational deformations of chain molecules. 3(2):178–187. Number: 2.
- [5] Hayward, S. and Kitao, A. The effect of end constraints on protein loop kinematics. 98(9):1976–1985. Number: 9.
- [6] Hayward, S. and Kitao, A. Monte carlo sampling with linear inverse kinematics for simulation of protein flexible regions. 11(8):3895–3905. Number: 8.
- [Karami et al.] Karami, Y., Rey, J., Postic, G., Murail, S., Tufféry, P., and de Vries, S. J. DaReUS-loop: a web server to model multiple loops in homology models. 47:W423–W428. Number: W1.

[Ko et al.] Ko, J., Lee, D., Park, H., Coutsiar, E. A., Lee, J., and Seok, C. The FALC-loop web server for protein loop modeling. 39:W210–W214. Number: suppl.

[López-Blanco et al.] López-Blanco, J. R., Canosa-Valls, A. J., Li, Y., and Chacón, P. RCD+: Fast loop modeling server. 44:W395–W400. Number: W1.

[Noonan et al.] Noonan, K., O’Brien, D., and Snoeyink, J. Probik: Protein backbone motion by inverse kinematics. 24(11):971–982. Number: 11.

[Tavernelli et al.] Tavernelli, I., Cotesta, S., and Di Iorio, E. E. Protein dynamics, thermal stability, and free-energy landscapes: A molecular dynamics investigation. 85(4):2641–2649. Number: 4 Publisher: Elsevier.

```
{
  "pdb_id": "1ADG",
  "fname" : "LADH_loopmovement.pdb",
  "chain": "A",
  "segbeg": 290,
  "segend": 301,
  "target_residues_phi": [ [291,-90],
                           [292,-110],
                           [293,-64],
                           [294,-90] ],
  "target_residues_psi": [ [291,122],
                           [292,-35],
                           [293,147] ],
  "constr_residues_phi": [295,296],
  "constr_residues_psi": [295,296],
  "iterations": 10000
}
```

Figure 5: Endpoint 1 JSON Payload

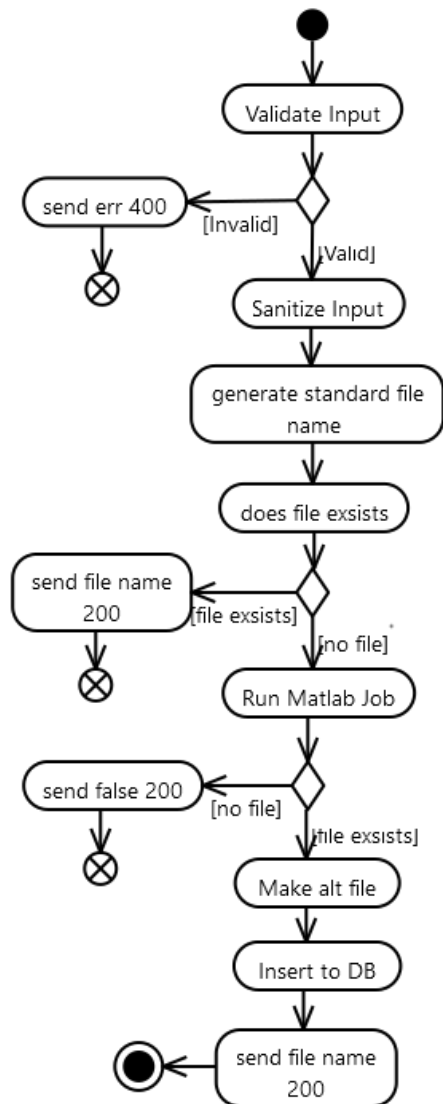


Figure 6: Endpoint 1 Activity diagram

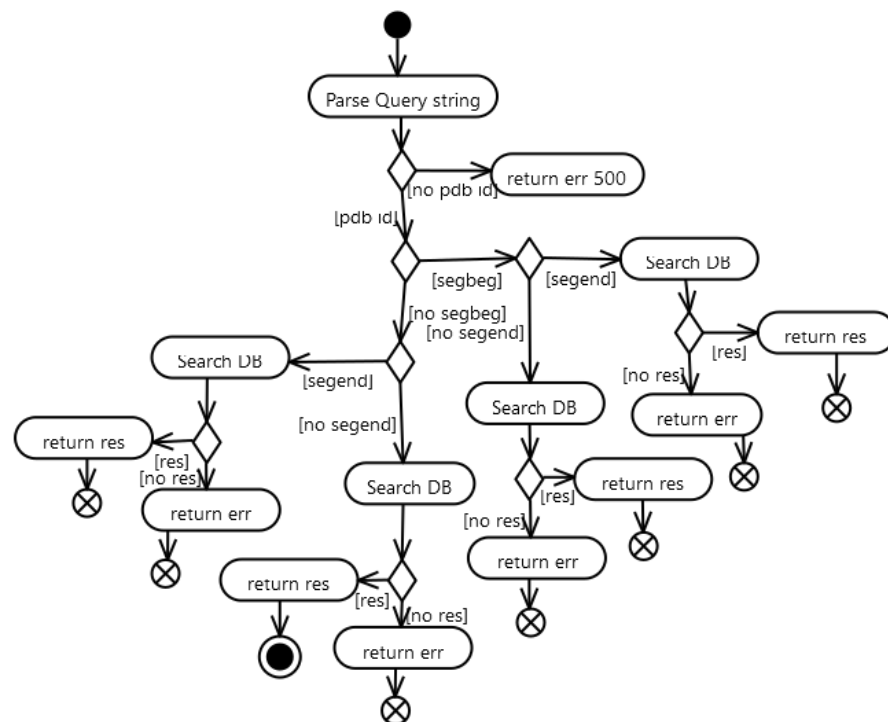


Figure 7: Endpoint 2 Activity diagram

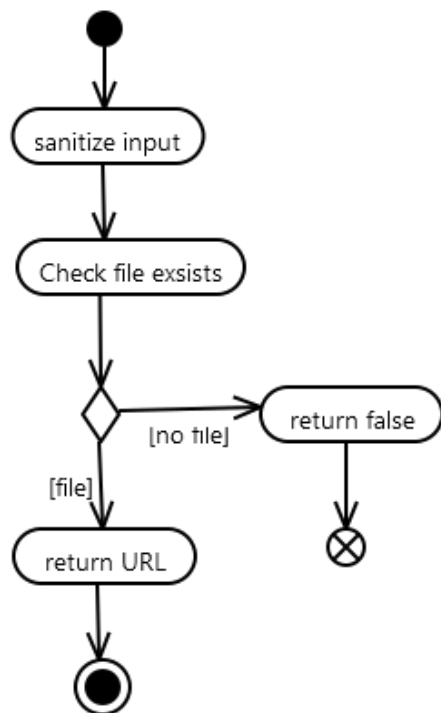


Figure 8: Endpoint 3 Activity diagram

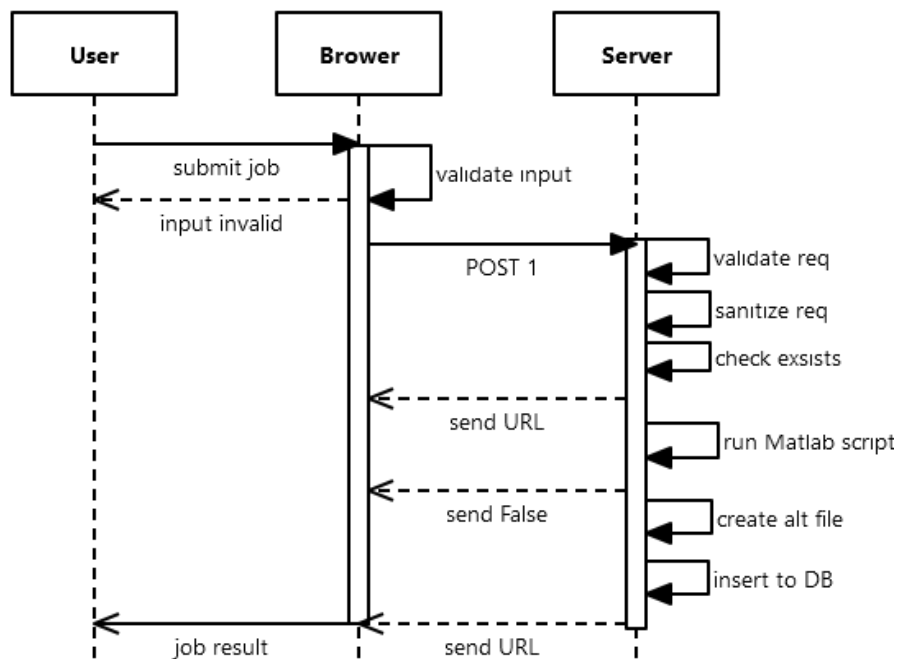


Figure 9: Endpoint 1 Sequence diagram

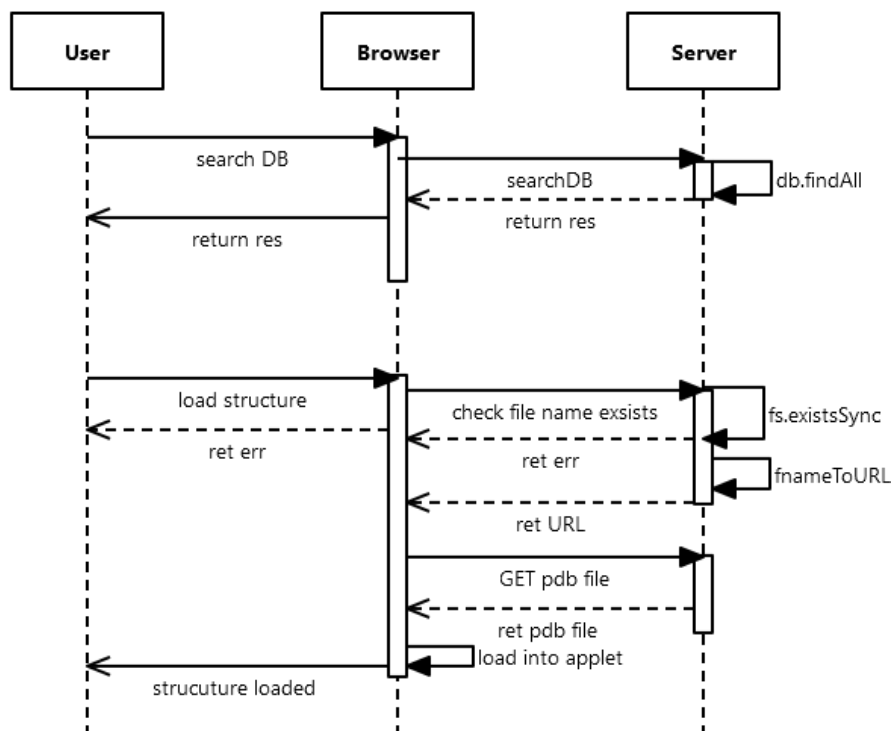


Figure 10: Endpoint 2 and 3 Sequence diagram

Home Submit Job View Structure Theme ▾

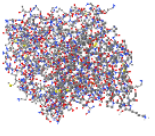
1ADG 1 A 2 290 3 301 4 Show Initial Torsions 5

Combined torsions

-82.44028359106635, 116.64577076406886, -80.34138058523655, -38.48854538204105, -168.58301207131993, 154.98625821088604, -138.7863696704609, 131.6303449911795, -64.05790329489498, 137.59915126398872, -49.2654447160971, -11.459217879867738, -97.33936036526714, -3.7924484117495174, -58.4013861569042, 142.2110698487128, -98.27841464147762, -163.12183070414372, -76.58017902711289, 147.67389383545375, -112.85843282055153, 134.48119589359854, -74.00003155115742, 126.42699092629826

290 24 find torsion Get Clear 25

seq 290, ϕ -82.44028359, ψ 116.64577076 26

 JSmol

7 8 add target phi remove target phi

291 -90, 292 -110, 293 -64, 294 -90, 9 10

11 12 add target psi remove target psi

291 122, 292 -35, 293 147, 13 14

15 add constrain phi remove constrain phi

295, 296, 16 17

18 add constrain psi remove constrain psi

295, 296, 19 20

10200 21 submit job 22 23

The file was created, try search for it on the view_structure page. Or download here : [link](#)

Figure 11: screen shot of submit job page

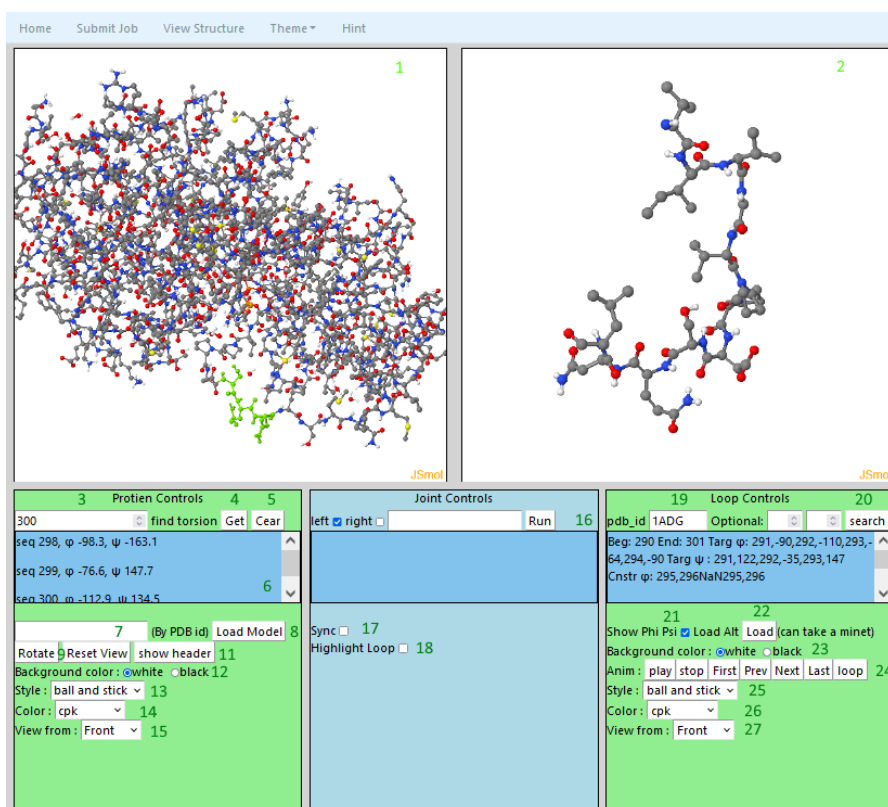


Figure 12: screen shot of view structure wide page

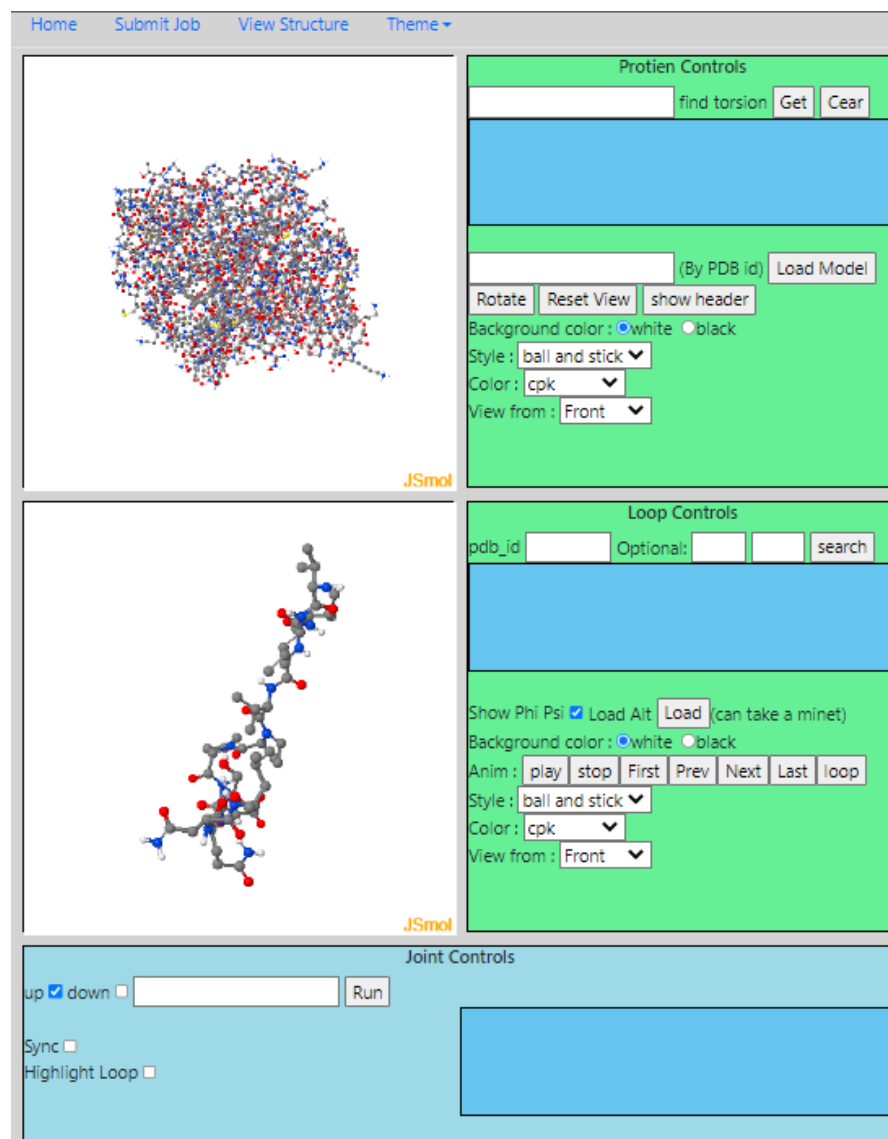


Figure 13: screen shot of view structure narrow page

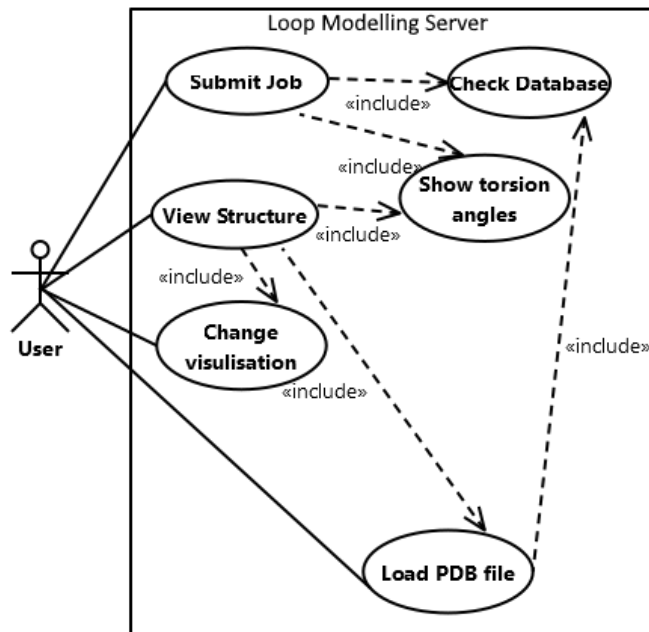


Figure 14: Use Case Diagram

```
function runMatlabScript1(dir1, scriptName, fname1, dir2) {
  return new Promise((resolve, reject) => {
    var matlabCommand = `matlab -batch ${scriptName}`;
    var moveCommand = `move ${dir1 + fname1} ${dir2}`;
    try{
      var proc = exec(`cd ${dir1} && ${matlabCommand} && ${moveCommand}`,
        (error, stdout, stderr) => {
          if (error) {
            reject(error);
          } else {
            resolve(true);
          }
        });
    } catch(error) {console.log(error)}
  });
}
```

Figure 15: Code that invokes MATLAB

```
function genStandardFileName(json_obj) {
  res="PDBID_"
  res=res+(json_obj.pdb_id).toUpperCase()
  res=res+'_CHAIN_${json_obj.chain}\'
  res=res+"_BEG_"
  res=res+json_obj.segbeg
  res=res+"_END_"
  res=res+json_obj.segend
  res=res+"_PHITARGS"

  for(var i=0;i<json_obj.target_residues_phi.length;i++){
    res=res+"_"+json_obj.target_residues_phi[i][0]
    res=res+"_"+json_obj.target_residues_phi[i][1]
  }
  res=res+"_PSITARGS"
  for(var i=0;i<json_obj.target_residues_psi.length;i++){
    res=res+"_"+json_obj.target_residues_psi[i][0]
    res=res+"_"+json_obj.target_residues_psi[i][1]
  }
  res=res+"_PHICONSTR"
  for(var i=0;i<json_obj.constr_residues_phi.length;i++){
    res=res+"_"+json_obj.constr_residues_phi[i]
  }
  res=res+"_PSICONSTR"
  for(var i=0;i<json_obj.constr_residues_psi.length;i++){
    res=res+"_"+json_obj.constr_residues_psi[i]
  }

  res=res+"_ITTR_"+json_obj.ititerations
  res=res+".pdb"
  return res
}
```

Figure 16: Code that generates standard file name

```
cd C:\Users\FYP\TAT_Matlabcode\ && matlab -batch  
"wrapper_loop_modeller2('1ADG','PDBID_1ADG_CHAIN_  
A_BEG_290_END_301_PHITARGS_291_-90_292_-110_293_-64  
_PSITARGS_291_122_292_-35_293_147  
_PHICONSTR_295_296_PSICONSTR_295_296  
_ITTR_10000.pdb','A',290,301,[[ 291 -90];  
[ 292 -110];[ 293 -64]],[[ 291 122];  
[ 292 -35];[ 293 147]],[ 295 296],[ 295 296],10000);  
exit;" && move C:\Users\FYP\TAT_Matlabcode\  
PDBID_1ADG_CHAIN_A_BEG_290_END_301_PHITARGS_291  
_-90_292_-110_293_-64_PSITARGS_291_122_292_  
-35_293_147_PHICONSTR_295_296_PSICONSTR  
_295_296_ITTR_10000.pdb  
C:\Users\FYP\server\src\public\OutputPDBS\
```

Figure 17: example command line on a windows system

1. A text input that takes the 4 character PDB ID
2. A text input that takes a single character representing the chain
3. A number input field that takes the start residue index
4. A number input field that takes the end residue index
5. A Button that loads a list of torsion angles into the space below the first row on inputs for the segment specified by start and end. To do this, it loads a JSmol applet in the section below (6)
6. Space for a JSmol applet to load
7. A number input that takes the amino acid index to set a target phi angle for
8. A number input that takes the target phi angle for the index in 7
9. A button to add the pair of inputs 7,8 to a stack displayed below
10. A button to remove the last index, target phi angle pair
11. A number input that takes the amino acid index to set a target psi angle for
12. A number input that takes the target angle for the index in 11
13. A button to add the pair of inputs 11,12 to a stack displayed below
14. A button to remove the last index, target psi angle pair from a stack displayed below
15. A number input field that takes the amino acid index to constrain the phi index of
16. A Button to add an amino acid index to constrain the phi angle of
17. A Button to remove the last index to constrain the psi angle of
18. A number input field that takes the amino acid index to constrain the psi index of
19. A Button to add an amino acid index to constrain the psi angle of
20. A Button to remove the last index to constrain the psi angle of
21. A number input field to set the number of iterations
22. A button to submit the job
23. A message that indicates whether the file was created successfully or issues were encountered

Figure 18: submit job input key

1. A JSmol applet for viewing a protein's structure
2. A JSmol applet for viewing the loops structure
3. A number input field that takes an amino acid index
4. A button that gets the phi psi angle for the residue specified in 3 and adds it to 6
5. A button that clears the phi psi angles shown in 6
6. An output area containing a list of phi, psi angles. Clicking on them highlights the residue in 1
7. A text input field that takes a 4 character PDB code
8. A button that loads the PDB code in 7 into applet 1
9. A button to rotate the model by 30 degrees about the y-axis
10. A button to reset the view
11. A button that shows a popup displaying the PDB file header for the model loaded in 1
12. A Jmol radial group to change the applet's background color
13. A Jmol menu to change the style
14. A Jmol menu to change the color
15. A Jmol menu to change the orientation
16. The left and right checkboxes select the JSmol applet. The text input field takes a JSmol script string. The run button runs the script on the selected applet. The box below shows the command history
17. A checkbox that syncs the view when moved with the mouse
18. A checkbox that highlights the loop displayed in applet 2, in applet 1
19. An input that takes a 4 character PDB code, and two number inputs that take segbeg and segend respectively
20. A button that searches the database for loops matching the properties in 19, and outputs it to the box below. Clicking on the loop description loads it into applet 2
21. A checkbox that shows the phi psi angles for carbon alpha atoms when the loop is animated
22. A button that loads in an alternative PDB file containing the same loop but surrounded by the rest of the structure unchanged. Takes a while
23. A Jmol radial group to change the applet's background color
24. A series of buttons for controlling the animation
25. A Jmol menu to change the style
26. A Jmol menu to change the color
27. A Jmol menu to change the orientation

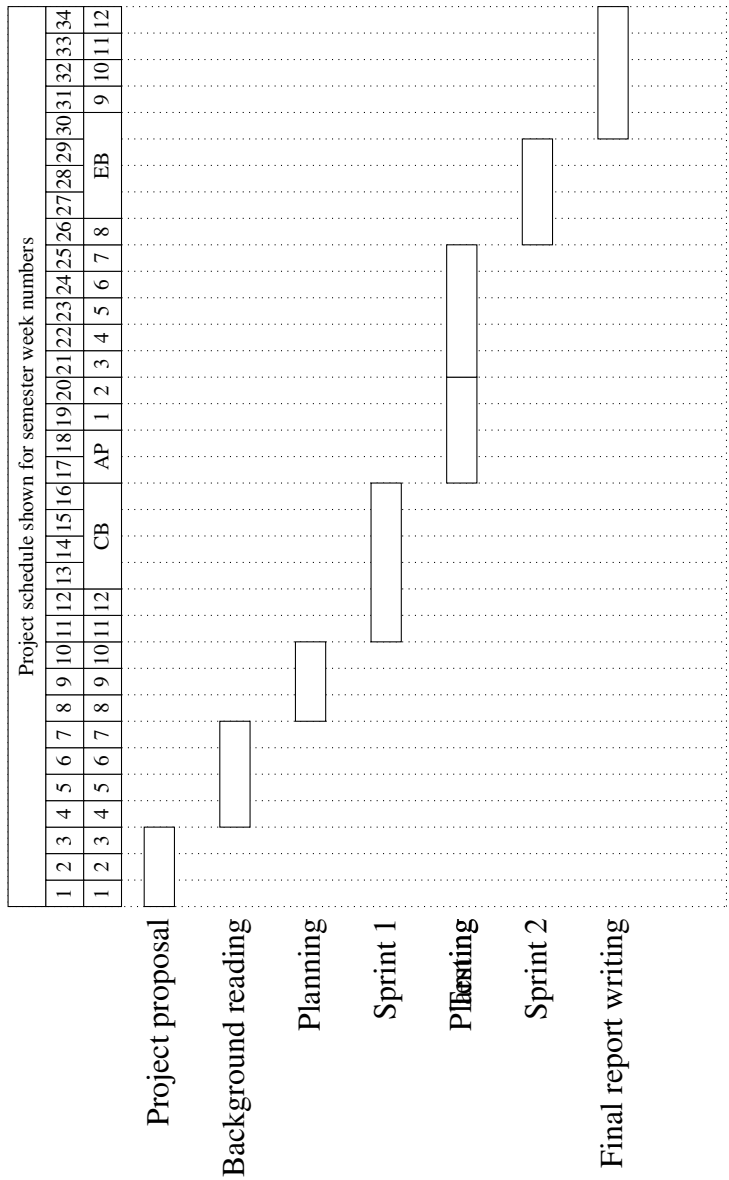


Figure 20: Revised project Gantt chart