Ashmit Khadka

Registration number 100227483

2021

Web Application: DNA Visualisation (Collaboration with TSL)

Supervised by Dr Jeannette Chin

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

Data visualisation bridges the gap in finding meaning for large, complex datasets. This project collaborates with The Sainsbury Laboratory (TSL) to visualise DNA sequences and interactions of potato-based protein samples with pathogen baits in the fight against plant disease. Using current industry standard technology stack with the MERN architecture (MongoDB, Express, React, Node), a data visualisation web application platform is created for DNA sequences and interactions using several visualisation techniques found in academia. The web application can import data into the database from industry standard file formats, encapsulate data within user account through user authentication and provides additional features such as ability to save visualisation state, exporting to image and linking between visualisations.

Acknowledgements

I would like personally to thank the following people and organisations for their warm contributions to this project:

- Dr. Jeannette Chin, my project supervisor, for providing me with guidance, availability, communication and support throughout the project.
- The Sainsbury Laboratory for providing the dataset to initiate the project.
- Dr. Pierre Chardaire, the final year project module organiser, for being available to answer module and project related queries.

Contents

1. Introduction

1.1. Background

The purpose of this project is to provide researchers tools to explore DNA sequences and pathogen – protein interaction. Exploration of these interactions provide insight into the mechanics of how these pathogens damage plants in the fight against plant diseases. We can refer to 1845 as a case study to understand the impact of such pathogens. In what is now called the Irish Potato Famine, a fungus like organism named Phytophthora destroyed 50% of the Irish potato crops in that year after rapidly spreading across farms in the country. The pathogen is estimated to have destroyed 75% of crops throughout several years. Roughly 1 million people died due to famine with many forced to relocate as refugees before it's end in 1852 (Kinealy, 2006). In a globalised world, impact of such disease today would affect food supply chains for many more people. At least 10% of global food production is lost to plant disease (Strange and Scott, 2005). Furthermore, plant pests and diseases put up to 82% of the attainable yield of cotton alone and over 50% for other major crops at risk of destruction (Oerke, 2006).

In many parts of the developing world today, people really on agriculture to feed themselves and to earn a living. For countries such as Sierra Leone and Somalia, over 50% of Gross Domestic Product (GDP) consist of agriculture alone (CIA, 2020). The potential economic and social devastation of plant disease for these nations and their people are serveer. It is important that we can identify the nature of these pathogens so that we can prevent such harm early as possible.

1.2. The Sainsbury Laboratory (TSL)

This project is initiated with collaboration from The Sainsbury Laboratory (TSL). TSL are a research institute that focuses on the science of plant-microbe interactions. They are based in Norwich Research Park along with the University of East Anglia. For this project, they have provided a dataset containing interactions of potato protein and bait proteins samples in a .csv file along with a .fasta file containing the DNA sequences for the respective potato proteins.

Upon initial inspection of the dataset, it quickly becomes apparent the motivations for creating a visualisation tool. With over 100,000 interactions in the dataset, a method of extracting meaningful data becomes essential in order obtain the analysis required fuel

their research.

2. Litreture Review

When evaluating the problems of the project, creating visualisations to show protein bait interactions and protein sequencing was a critical factor in determining the success of the project itself. A key question that I have used to fuel my research has been: What visualisation techniques can I implement to display the dataset clearly for analysis?

2.1. Dotplot

This is a 2-dimensional graphical method of comparing two DNA sequences. The aim of the Dot plot visualisation is to identify close regions of similarity within biological sequences A and B. Similarities are highlighted with marked dots aligned in the Y and X axis where, the X axis, represents the index value of sequence A and Y the axis, the index value along sequence B. This is a derivative of the recurrence plot, which is used in statistics and chaos theory to show for each moment in time; phase space trajectories that meet in similar regions. Two important concepts to understand for Dot plot are Window Size and Identity Threshold

- Window Size: A subset length of adjacent base pairs within a biological sequence that represent a region for comparison E.g. a sequence of "CGACGTACGAT" with a Window Size of "3" would result in the following subsets "[CGA], [CGT], [ACG]".
- Identity Threshold: The percentage threshold in the similarity of Window Size, N , subsets to be classified as a match. E.g. a subset [CGA] at index, i, of sequence A and subset [TGA] of sequence B with an Identity threshold of "50%" will be a match because they are 66.6% similar.

D-Genies, Cabanettes and Klopp (2018), is a freeware project available under the GNU General Public Licence (GPL) that generates Dot Plot Visualisations. This is a standalone desktop and web application that processes specifically large genomes sequences. The Visualisation technology itself is powered by D3.js frontend with a python-based backend. The option of a web standalone solution makes D-Genies more

accessible compared to fully desktop solutions. However, customisation is limited with no options for window size and identity threshold.

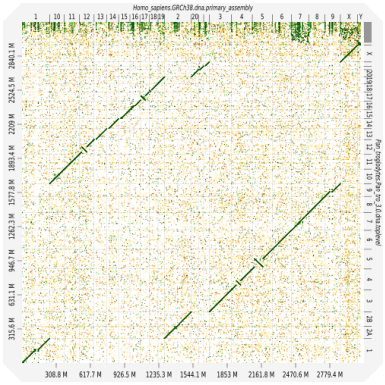


Figure 1: D-Genies Dot Plot visualisation

Flexidot, Seibt et al. (2018), is a cross-platform dot plot visualisation generator. This is a general dot plot freeware that facilitates speciality features in biological sequences. The software is 100% based on python. Flexidots highly customisable UI is fantastic with advanced features such as Mismatch/ambiguity handling and similarity shading not found in other software such as D-Geries. However, the software is built on slightly out dated technology with python 2.7 no longer supported.

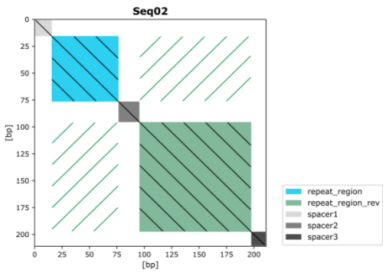


Figure 2: Flexidot Dot Plot visualisation

2.2. Squiggle

Squiggle, Lee (2019), is biological sequence visualisation method created by Benjamin Lee that utilises a Huffman encoding method based on the UCSC .2bit format for storing DNA sequences. Using this method, we can encode at set of 4 base sequences into a

2-bit binary sequence where T -> 00, C -> 01, A -> 10 and D -> 11. Rather than storing the encoded binary sequence into as bytes, we can instead represent a bit 0 encounter as a downward graph vector and 1 as an upward vector to give 4 distinct vectors, Qi et al. (2011).

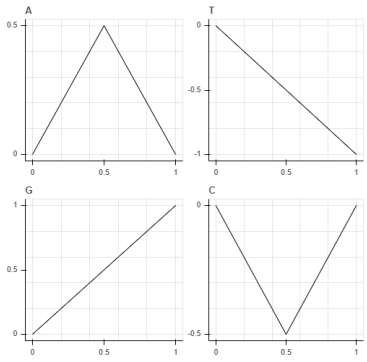


Figure 3: Squiggle algorithm directional vector representation

Benjamin Lee's Squiggle python library encodes a string DNA sequence representation into 2 two arrays of x and y coordinates. This output can then be used by a frontend solution for visualisation. In the Squiggle website, Boken, a python based web visualisation library is used is used to power the frontend. This means the Squiggle library it's self is not a full visualisation solution but rather a backend encoding tool.

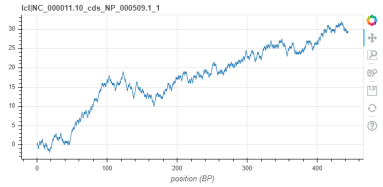


Figure 4: Squiggle algorithm visualisation

DNAvisualisation.org, Lee et al. (2019), is a web-based DNA visualisation tool that uses the Squiggle library to provide a frontend solution. The website is built using JavaScript with a Vue.js based frontend. This is a great solution because of it's simplicity. The UI makes selecting a FASTA sequence file and the visualising data fast and straightforward. However, the tool lacks nice to have features such as showing the sequence it's self and a way to save the visualisation.

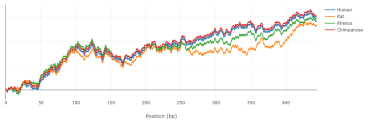


Figure 5: Squiggle algorithm visualisation with multiple sequence data

2.3. Yau's Method

Yau's method, Yau et al. (2003), uses unit vectors to map individual base sequence into a line. Unlike the Squiggle method, these are straight vectors that deviate directly from the origin. Bases C and T (pyrimidine bases) transform upwards while bases A and G (purine bases) transform downwards. The unit vectors are as below.

$$A \to \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right), T \to \left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right), G \to \left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right), C \to \left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right).$$

Figure 6: Unit vector transformations in Yau's method

The nature of the unit vectors mean that there is no 1 to 1 mapping of the X domain values with the base sequence.

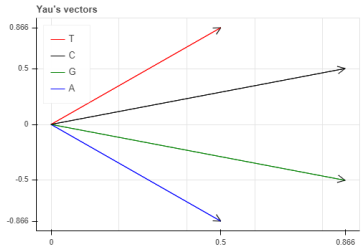


Figure 7: Gates's method visualisation

The Yau method is visualised in DNAVisualisation.org. In this example we can see a very similar graph when compared to the Squiggle method. The vectors result in greater divergences among the base sequences. This makes it clearer to recognise the structural patterns created from the base sequence.

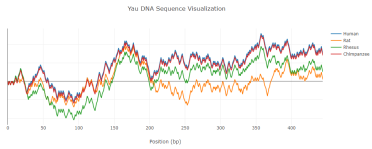


Figure 8: Gates's method visualisation multiple sequences

2.4. Gate's Method

Gate's method, Gates (1986), is the perhaps the most unique of the line-based visualisation methods. This is because the points can go in all directions of the x and y axis. In Gate's method, instances of base T result in an increase in Y while X remains same. Similarly, Instance of base A result in decrease of Y. instances of C and G result in increase and decrease of X value respectively while the Y value remains the same. Effectively, the sequence is encoded into a 2D "walk" where the 4 base sequences are mapped respectively into 4 transformations. As Gates summarises in his Journal of Theoretical Biology, the DNA sequence can be converted into specially represented visualisations to uncover canonical patters and structures. This enables unique visualisations of multiple DNA sequences.

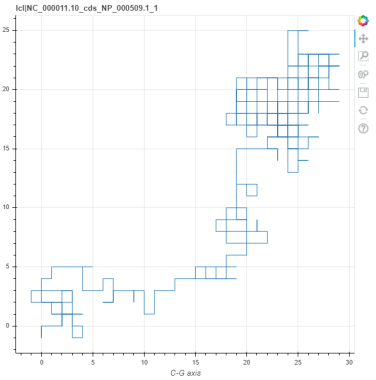


Figure 9: Gates's method visualisation

Here we can observe visualisation of a subsequence within the human genome sequences using Bokeh.

Similarly, here we can see the same visualisation on DNA visualisation.org. We can see the X and Y axis updated to facilitate the gates visualisation.

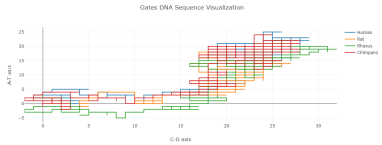


Figure 10: Gates's method visualisation multiple sequences

2.5. Randic's Method

Randic's method, Randić et al. (2003), uses a discrete set of Y values composed of each base while the values along the x axis correspond to the base position. This creates a tablature structure very similar to a musical notation for fretted string instruments such as a guitar. Unlike the previous methods, the mapping and structure of the lines representing the base sequence are more literal.

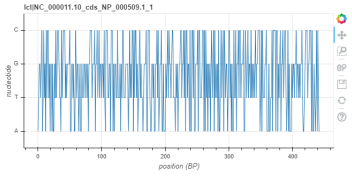


Figure 11: Radic's method visualisation

In this example, we see the Randic visualisation in action using the same sample as before. This time the y axis represent the discrete set of nucleotides as apposed to the continuous diverged values in Yau or Squiggle.

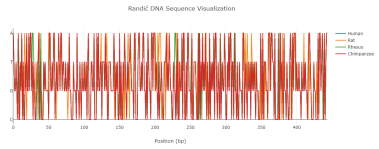


Figure 12: Radic's method visualisation multiple sequences

The visualisation generated by DNAVisualisation.org shows a busy graph with very dense information when adding additional samples. This can be difficult to interpret and any visualisation implementation of Randic would largely benefit from a zoom in and pan function. However, identifying structural differences specific regions of the base sequence are effective in this method when limiting the domain to a smaller subset of the sequence.

Figure 13: Radic's method visualisation zoomed in

2.6. Qi's Method

Qi's method, Qi and Qi (2007), is identical to Randic's in concept but with a larger set of dinucleotide Y values. These values are combination of nucleotide pairs. With 4 bases A, T, C, G we have 4X4 pairs giving a range of 16 values e.g. AA, AT, AC.. GG. The pairs represent adjacent nucleotides along the base sequence.

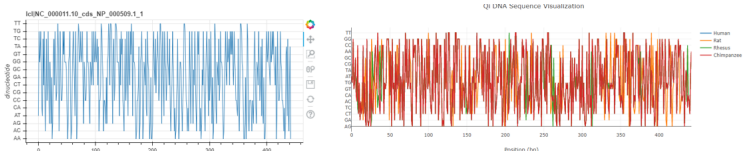


Figure 14: Qi's method visualisation single sequence (left), multiple sequences (right)

Using the visualisations generated by Squiggle and DNAvisialisation.org, we can observe very similar sequence pattern recognition in Qi's method compared to Randic's method. However, the greater base range in Y values adds more resolution to the graph. This makes identifying and comparing DNA structures easier. Despite this, recognising specific patterns is clearer in Randic's method since there are only 4 distinct values compared 16 in Qi's method. Again, Implementation of the Qi method would greatly benefit from zoom and pan features

2.7. Network

The network visualisation, often called Network Graph, allows visualisation of relationships between two entities in this case, proteins and baits. There are three core concepts of a network to understand:

- Nodes: These are the vertices that represent an entity object.
- Edges: These are the relationship between the entity objects.

- Weights: This is an optional attribute that indicates the strength of the relationship.

For our use case, we must model a many to many relationships between the two entities. This means an undirected graph must be created where relationships (edges) can be directed either way. Furthermore, the relationships themselves contain a value, this is the strength of the protein and bait interaction, therefore, the edges must be weighted.

In the Paper, Bioinformatic Analysis of Gene Variants from Gastroschisis Recurrence Identifies Multiple Novel Pathogenetic Pathways, Salinas-Torres et al. (2019), we can see the authors using a network analysis to visualise interactions between different genomes. They have used colour coordination to differentiate types of nodes and indicate interaction weight. For example, the purple edge represents a "experimentally determined" relationships while green "gene neighbourhood" relationships.

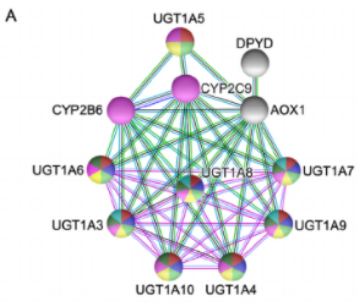


Figure 15: Network visualisation from Salinas-Torres et al. (2019)

In the paper NetworkAnalyst - integrative approaches for protein–protein interaction network analysis and visual exploration, Xia et al. (2014), we can observe a complex network graph created using NetworkAnalyst to visualise complex networks

The tool contains additional features to explore the densely populated visualisation. For example, hovering over a node will only show its direct neighbours, mouse wheel for zoom and a mouse click for drag. This is an excellent way of navigating a large data set and why they recommend using a mouse with a scroll wheel.

2.8. Edge Bundling

Edge Bundling is a subset of a network that uses a dendrogram network structure to visualise relationships. The dendrogram features the root node which then gives birth to

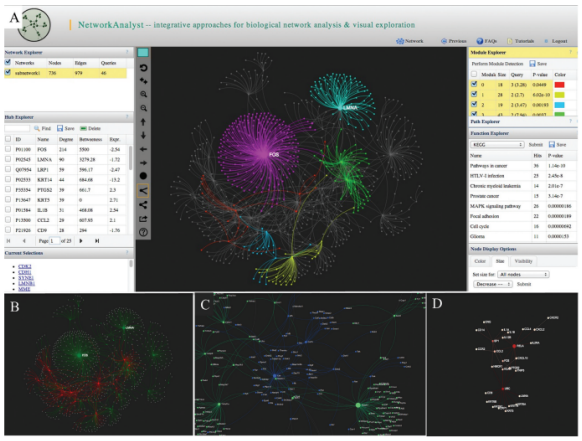


Figure 16: NetworkAnalyst's network visualisation

children and so forth. This means that each level of the dendrogram represents a layer of abstractions to from a natural hierarchy.

In Edge Bundling, the dendrograms are positioned circularly where the leave nodes from edges to each other. These edges can be formed from leave nodes where parent node that have no relation. The circular structure makes good use of space and forms a logical way of spacing the nodes out for better clarity.

In the paper State of the Art in Edge and Trail Bundling Techniques, Lhuillier et al. (2017), different techniques are discussed and analysed. eXplorer is one of the software demonstrated with it's "Radial view". It features a colour intensity map to indicate interaction weight and colour legend for the nodes.

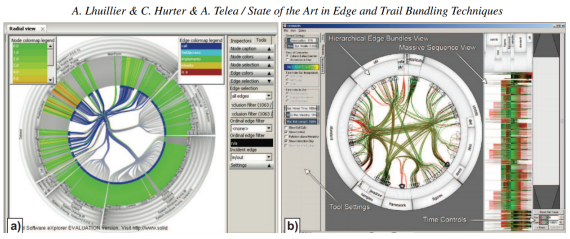


Figure 17: eXplorer's radial view

The paper highlights the advantages of using Edge Bundling over other visualisations for interactions. For example, compared to crusting and path displacement techniques, we can better see overlapping density, better scalability and adjustability using Edge Bundling. It also highlights draw backs such as clustering provides better path discrim-

ination while path displacement offers better overall overlap avoidance. The overall conclusion being, Edge Bundling is the superior method offering a bit of everything.

3. Design

3.1. Persona and Scenario - A

Joel Greenwood

- Age: 19
- Gender: Male
- Status: Biology Student

Motivations:

- Graduate from university with First Class Honours in BSc Biological Science.
- Create visual analysis of multiple DNA samples for coursework.

Frustrations:

- Using industry-standard DNA tools is too expansive.
- Existing free tools are too basic in feature, doesn't provide enough detail for good analysis.

Scenario

- Joel wakes up early at 7am to get ready and attend his Evolutionary Biology and Conservation Genetics lecture at 9am.
- After the lecture, Joel enters the library to continue his coursework for this module.
- Joel loads up the sample genetics datasets provided for the coursework into the system database
- Using the dot plot, Joel identifies regions in the DNA samples that have similar sequences.
- Joes uses this data for the analysis for his coursework

3.2. Persona and Scenario - B

Kelly Rowe

- Age: 42
- Gender: Female
- Status: Researcher

Motivations:

- Compare interactions of bait proteins with several protein samples.
- Research diseased to understand them better and find solutions that will help food crop production.

Frustrations:

- There are no available tools for protein-to-protein interaction visualisation in the freeware space.
- Existing tools are not specific enough for the protein-based datasets.

Scenario:

- Kelly receives a report from the labs for the ten pathogenic protein samples she identified last week.
- The report contains interaction data between the baits and potato protein samples.
- Kelly uploads the interaction data into the system database for visualisation.
- Using the network visualisation, Kelly filters the dataset to find the interactions with the highest molecular weight to interaction ratio.
- Kelly uses filters next to look at proteins that belong to RD2 family.
- Using the observations from the visualisation, Kelly produces an analysis of correlations that were found between the bait and protein samples.

3.3. Use cases

3.3.1. Overview

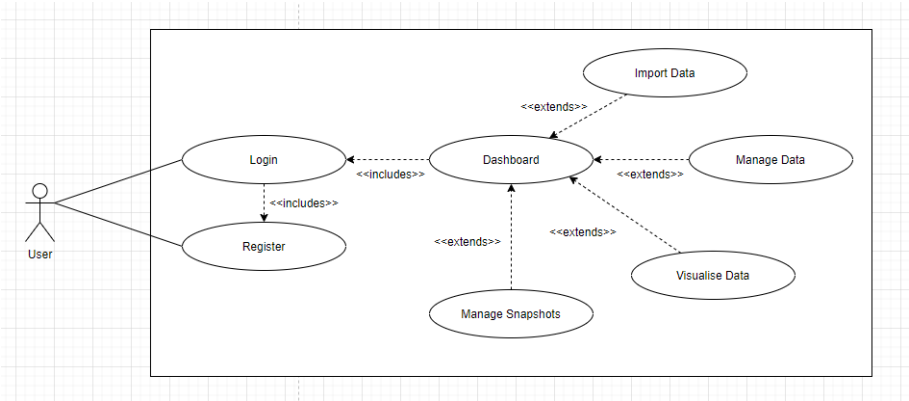


Figure 18: Use case diagram to show general system flow

The general flow for a new user should be to register an account first and then login to their dashboard. Once authenticated, the user can access several features.

- Import Data: Import protein (.fasta file)or interaction data (.csv file) into the database.
- Manage Data: Manage imported data by updating or deleting items or add new items.
- Visualisation: Visualise data by entering a visualisation screen of choice.
- *Snapshots: Manage Snapshots by selecting, updating or deleting items (added in the second iteration after user testing).

3.3.2. Visualisation

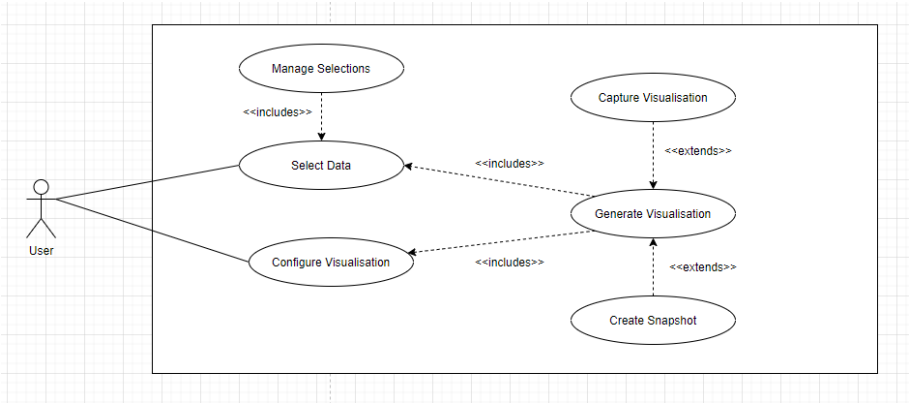


Figure 19: Use case diagram to show visualisation screen flow

The use case diagram above demonstrates a common user interaction flow for visualisation screens. When the user selects a visualisation method, they can choose sample data to visualise. Additionally, users can also modify the default visualisation configuration for their needs if necessary. For exporting the current visualisation, user can capture the visualisation to save it as an image (2nd interaction). Additionally, users can choose to save the visualisation state as a whole including it's dataset and configuration by creating a shapshot (2nd interaction).

3.3.3. Data Import

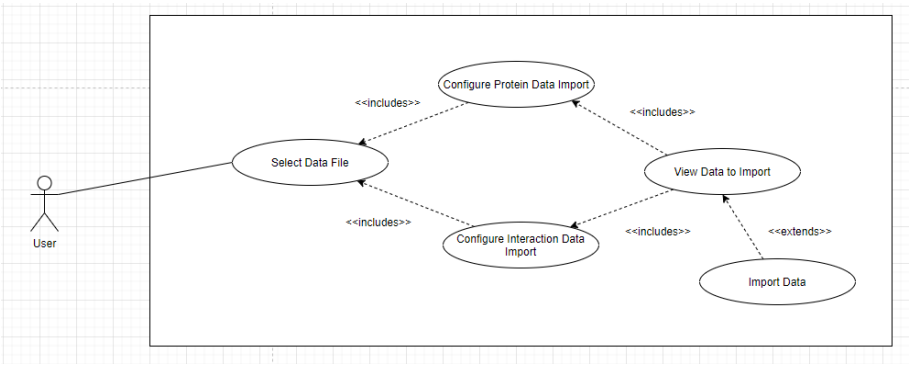


Figure 20: Use case diagram to show data import flow

First the user selects a data file to import. By checking the file format, the user is taken automatically either to the protein data configuration screen or the interaction data configuration screen. Here, users can modify the default import configuration for the respective formats if necessary. Finally, the user can view data objects created after parsing the import file for confirmation. The import can be confirmed or the user can go back to reconfigure import if changes are required.

3.3.4. Manage Data

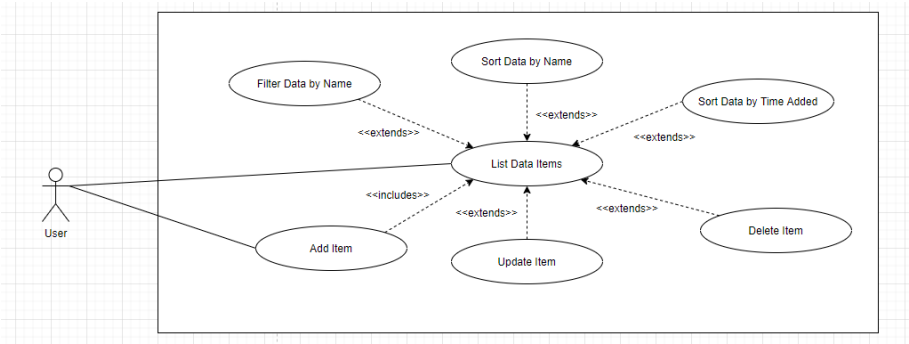


Figure 21: Use case diagram to show data managment flow

Users can manage imported data or create new items. These items are Snapshots, Proteins and Interactions. In the view, users can search by item name or sort the data by name and import date. Users can also update or delete individual items.

3.4. Requirements - MoSCoW

Must have:

- Data Importing - Facilitate allowing the user to select a data file (.fasta) for protein data and (.csv) for interactions. Enable configuration of import and store data into the database for retrieval in visualisations. [TSL requirement]
- Security - Add user authentication with mandatory login credentials to protect and encapsulate data. Allow user to register an account.
- Visualisation - Dot plot: Calculate matches between two DNA sequences taking into account Window Size and Identity Threshold.

- Visualisation - Gates: Calculate respective Y and X values for given base sequence index value using the an algorithm for Gates's method. Plot coordinates into into a line graph.
- Visualisation - Squiggle: Calculate respective Y values for given base sequence index value using the an algorithm for the Squiggle method. Plot coordinates into into a line graph.
- Visualisation - Yau: Calculate respective Y values for given base sequence index value using the an algorithm for Yau's method. Plot coordinates into into a line graph.
- Visualisation - Randic: Calculate respective Y values for given base sequence index value using the an algorithm for Randic's method. Plot coordinates into into a line graph. Allow user to zoom into data and pan with mouse interaction
- Visualisation - Qi: Calculate respective Y values for given base sequence index value using the an algorithm for Qi's method. Plot coordinates into into a line graph. Allow user to zoom into data and pan with mouse interaction.
- Visualisation - Network: Plot nodes for proteins and baits with likes to represent their relationships. Implement link weight with colour intensity to show the interaction strength. Show adjacent linked nodes on mouse hover. [TSL requirement]
- Visualisation - Edge Bundling: Plot nodes for proteins and baits with links to represent their relationships. Implement link weight with colour intensity to show interaction strength. Group nodes into set of entities and common hierarchies. Show adjacent linked nodes on mouse hover.

Should have:

- Responsive - The user interface adapts to different screen resolutions by adjusting elements to best fit the view port.
- Data Management - The user can modify, delete or add elements on top of the imported data manually.
- Visualisation Interaction - Visualisations enable interactions with it's elements and data using the cursor.

- Visualisation Configuration - Configuration of visualisations should be possible for adjustable attributes and properties.
- Interconnection between interaction visualisations (network and edge bundling) and the all others by transferring it's dataset directly.

Could have:

- Snapshots - Save current visualisation stage including it's dataset and configuration. Allow user to load the snapshot state later if needed. [User testing suggestion]
- Image Export - Export current visualisation into an image. [User testing suggestion]

Wont have:

- Mobile first design - Visualisations should be designed to reward large screen real estate, using pointers for interaction. This directly contradicts mobile first design, such use case would be impractical.

3.5. Prototypes

In order to create designs that translate fluently into human computer interaction, Norman's principles of interaction designs will be at the heart of the prototyping process. Here Donald Normal describes 6 principles used to evaluate UI which will be referenced through out this section:

- Visibility: Users need to know what all the options are and how to access them.
- Feedback: Actions need to have reactions, There must be indication that the user's action has initiated a reaction.
- Affordance: The relationship between what something looks like and how it's used must be clear.
- Mapping: Controls must resemble what they affect.
- Constraints: limitations and restrictions must guide the user.
- Consistency: Actions must cause the same reactions every time.

3.5.1. Master Layout

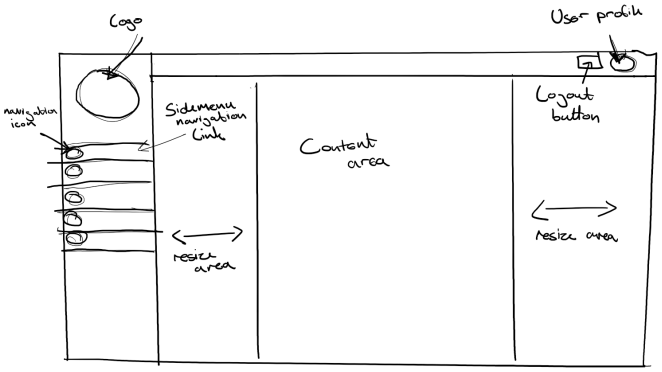
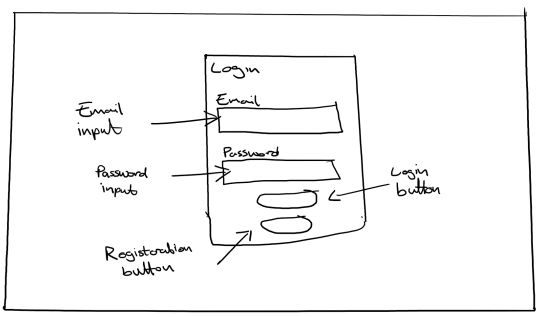


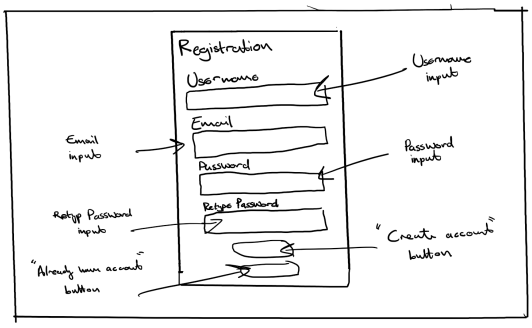
Figure 22: This is an example caption

The master layout will consist of a fixed side menu on the left and title bar on top. Inspired by the clean and minimalist layout of digitalocean.com, the side menu provides excellent visibility of navigation, utilising the else wasted horizontal space. The title bar provides user actions for logging out and screen specific content such as capture and snapshot buttons to make full use of this area. The resize area acts as a buffer for varying screen resolutions.

3.5.2. Authentication Layout



(a) Login Screen

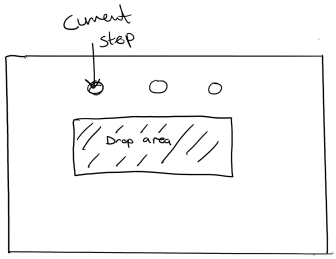


(b) Registration Screen

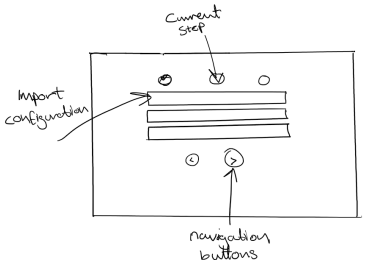
Figure 23: Authentication Screens

The login and registration forms are centered in the screen. They will contain inputs for user credentials which are validated using a defined schema for example, username up to 25 characters and password at least 6 characters long. Enforcing password complexity such as "atleast one number" will only reduce security as the single character will have restricted in combinations, 10 in this example (0,1..9) rather than 62. Allowing the user to set a preferred password is better for usability. The email acts as the unique public identifier enabling the any preferred username. Repeating the email reduces probability of errors.

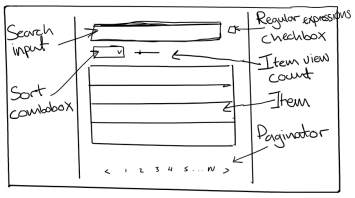
3.5.3. Data Import Data Management



(a) Data Import Screen (first) for select



(b) Data Import Screen (second) for setup

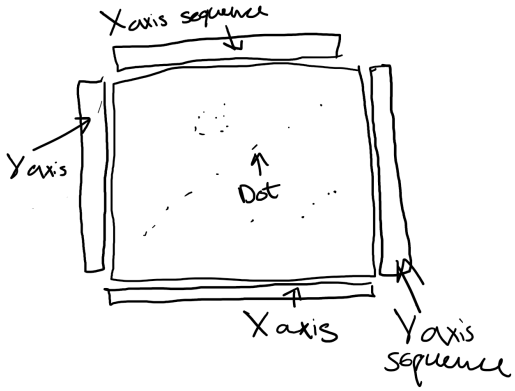


(c) Data Import Screen (last) for confirm and Data Management Screen

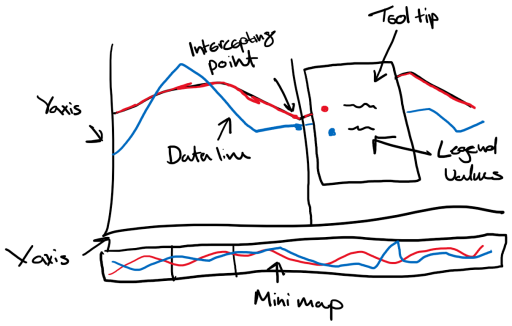
The data import is divided into 3 processes. Selecting the data file, configuring import settings and confirming the parsed data for import. Users can select a file or drag and drop which is an intuitive alternative that demonstrate good affordance. The user will be automatically directed to the appropriate configuration screen for the file type using constraints to guide the user flow. Finally, all data ready for import can be previewed first for confirmation, this prevents configuration errors. The icons show all available steps and progress which is important for visibility.

Users need to select data for visualisation, view imported data and also modify data if necessary. This means a search form is required. Users can enter the name of the data entity in the search bar to retrieve matching data items. These items display overview information for good visibility in differentiating data. Sorting options by name and import date accelerates finding information

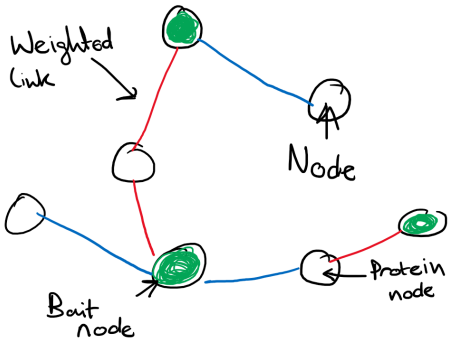
3.5.4. Visualisations



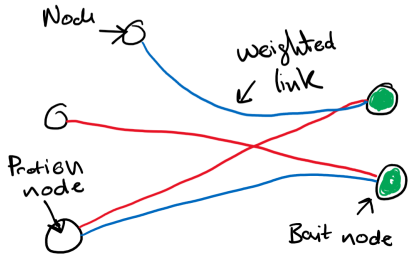
(a) Dot Plot visualisation



(b) Line based visualisations



(c) Network visualisation



(d) Edge Bundling visualisation

The Dot Plot will feature labeled X and Y axis corresponding to the base sequence which will be displayed in opposite sides to make full use of space. This way the mouse X and Y lines can be better used for measurement without overlapping the axis labels. The inclusion of the base sequence along the axis was feature not found in the current Dot Plot tools from the literature review. This presents an opportunity to enhance current tools which is particularly beneficial to short/medium size sequences. In the second iteration, I have implemented a heatmap overlaying the dots as suggested in the user feedback to quickly identify regions of significance. The colour intensity scale for the heatmap is positioned on the right.

Visualisations for Squiggle, Gates, Randic, Qi and Yau's methods can all be categorised as a line based visualisation for simplification. These visualisations contain a labelled X and Y axis containing their respective data outputs. Multiple lines can be

visualised for comparison. The pointer will draw a line on the X axis to show the domain value while a tool tip will follow along showing respective Y values in a key-value view for good mapping and visibility. Users can also zoom and pan into the data which is particularly beneficial to the Randic and Qi's methods as highlighted in the literature review. This also enables the visualisation to scale much better when handling large datasets. Finally a mini map is also displayed bellow the main view for easy navigation and visibility when zoomed in, further supporting the zoom and pan features and enhancing the visualisation from dnavisualisation.org as explained in the literature review.

The Network and it's subset, Edge Bundling, visualisations have similar characteristics and share core features. These include coloured nodes for baits and proteins with a link between them to represent an interactions. The link is weighted as indicated by it's colour intensity. Users can also hover the pointer over a node to highlight it's interactions or click on the node to isolate the interactions completely. Again, this demonstrates good mapping, using the pointer to encourage exploration of the dataset.

3.6. Dashboard

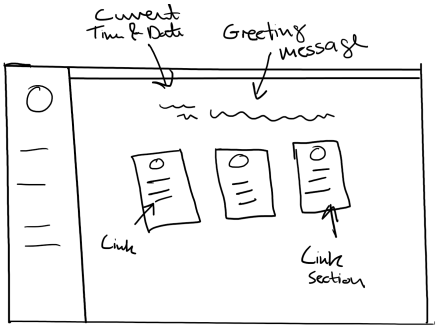


Figure 26: Dashboard Screen

The Dashboard serves as a portal screen to other core features. It contains a greeting message with the current time and date place on the side using font size and typography hierarchy for easy reading and aesthetics. The links contain icons for it's respective category demonstrating affordance while the colour scheme with the buttons matches the system theme for consistency.

3.7. Psudocode

These are the psudocode for the core visualisation logic used in the project.

Algorithm 1: Function generateDotPlot
Data: *sequenceWindow* = Integer of adjacent bases to compare,
sequenceA = Type string of given biological base sequence,
sequenceB = Type string of given biological base sequence
Result: Type array of dot points
begin
  data ← []
  if *sequenceA.length sequenceB.length* **then**
    plotSize ← sequenceA.length
  else
    plotSize ← sequenceB.length
  end
  for *x* ← 0 **to** *plotSize - sequenceWindow* **do**
    subSequenceA ← sequenceA[x **to** x+sequenceWindow]
    for *y* ← 0 **to** *plotSize - sequenceWindow* **do**
      subSequenceB ← sequenceB[y **to** y+sequenceWindow]
      isMatch ← sequenceWindowCompare ← subSequenceA, subSequenceB
      if *isMatch* **then**
        data.add ← [x, y]
      else
      end
    end
  end
end
return *data*

(a) Dot Plot points generator

Algorithm 2: Function sequenceWindowCompare
Data: *sequenceWindow* = Integer of adjacent bases to compare.
identityThreshold = Integer percentage match required got dot.
subSequenceA = Type string of given biological base sequence.
subSequenceB = Type string of given biological base sequence.
Result: Type float, percentage similarly of sub sequences
begin
  matches ← 0
  for *i* ← 0 **to** *sequenceWindow* **do**
    if *subSequenceA*[*i*] == *subSequenceB*[*i*] **then**
      matches ← *matches* + 1 **end**
    end
  end
  similarityPercentage ← matches/sequenceWindow * 100
  return *similarityPercentage* ≥ *identityThreshold*

(b) Dot Plot comparing windows

Algorithm 3: Function generateHeatmap
Data: *data* = Type array of dot points
sections = Type integer number n of n*n heatmap grid
plotSize =
Result: Type array for heatmap area points
begin
  hashMap ← []
  basePerSection ← plotSize/sections
1 **for** *i* ← 0 **to** *data.length* **do**
    location ← data[i].x % basePerSection + '0' + data[i].y % basePerSection
    if *hashMap.contains* ← *location* **then**
      hashMap[location].value ← hashMap[location].value + 1
    else
      hashMap[location] ← Object {
        x ← Math.floor(point.x / basePerSection) * basePerSection,
        y ← Math.floor(point.y / basePerSection) * basePerSection,
        value ← 0
      }
    end
  end
end
return *hashMap.values*

(c) Dot plot generating heatmap (2nd iteration)

Algorithm 5: Gate's Method Pseudocode
Data: *sequence* = Type string of given biological base sequence
Result: A 2d array of coordinates x and y
begin
  x ← [0]
  y ← [0]
1 **for** 0 **to** *sequence.length* **do**
    if *sequence*[*i*] == '*A*' **then**
      x.add ← *x*[*x.length* − 1]
      y.add ← *y*[*y.length* − 1] − 1
    else if *sequence*[*i*] == '*C*' **then**
      x.add ← *x*[*x.length* − 1] − 1
      y.add ← *y*[*y.length* − 1]
    else if *sequence*[*i*] == '*T*' **then**
      x.add ← *x*[*x.length* − 1]
      y.add ← *y*[*y.length* − 1] + 1
    else if *sequence*[*i*] == '*G*' **then**
      x.add ← *x*[*x.length* − 1] + 1
      y.add ← *y*[*y.length* − 1]
    end
    return [*x*, *y*]
end

(d) Gate's method

Algorithm 7: Randic's Method Pseudocode
Data: *sequence* = Type string of given biological base sequence
Result: A 2d array of coordinates x and y
begin
  x ← []
  y ← []
1 **for** *i* ← 0 **to** *sequence.length* **do**
    x.add ← *i*
    if *sequence*[*i*] == '*A*' **then**
      y.add ← 3
    else if *sequence*[*i*] == '*C*' **then**
      y.add ← 0
    else if *sequence*[*i*] == '*T*' **then**
      y.add ← 2
    else if *sequence*[*i*] == '*G*' **then**
      y.add ← 1
  end
  return [*x*, *y*]
end

(e) Randic's method

Algorithm 4: Squiggle Method Pseudocode
Data: *sequence* = Type string of given biological base sequence
Result: A 2d array of coordinates x and y
begin
  x ← []
  y ← []
  runningValue ← 0;
1 **for** *i* ← 0 **to** 2 ∗ *sequence.length* + 1 **do**
    x[*i*] ← *i* ∗ 0.5
  end
2 **for** *i* ← 0 **to** *sequence.length* **do**
    if *sequence*[*i*] == '*A*' **then**
      y.concatenate ← [*runningValue* + 0.5, *runningValue*]
    else if *sequence*[*i*] == '*C*' **then**
      y.concatenate ← [*runningValue* − 0.5, *runningValue*]
    else if *sequence*[*i*] == '*T*' **then**
      y.concatenate ← [*runningValue* − 0.5, *runningValue* − 1]
      runningValue ← *runningValue* − 1
    else if *sequence*[*i*] == '*G*' **then**
      y.concatenate ← [*runningValue* + 0.5, *runningValue* + 1]
      runningValue ← *runningValue* + 1
    else
      y.concatenate ← [*runningValue*, *runningValue*]
  end
  return [*x*, *y*]
end

(f) Squiggle method

Algorithm 6: Yau's Method Pseudocode
Data: *sequence* = Type string of given biological base sequence
Result: A 2d array of coordinates x and y
begin
  x ← [0]
  y ← [0]
1 **for** 0 **to** *sequence.length* **do**
    if *sequence*[*i*] == '*A*' **then**
      x.add ← *x*[*x.length* − 1] + 0.5
      y.add ← *y*[*y.length* − 1] − $(\sqrt{3}/2)$
    else if *sequence*[*i*] == '*C*' **then**
      x.add ← *x*[*x.length* − 1] + $(\sqrt{3}/2)$
      y.add ← *y*[*y.length* − 1] − 0.5
    else if *sequence*[*i*] == '*T*' **then**
      x.add ← *x*[*x.length* − 1] + 0.5
      y.add ← *y*[*y.length* − 1] + $(\sqrt{3}/2)$
    else if *sequence*[*i*] == '*G*' **then**
      x.add ← *x*[*x.length* − 1] + $(\sqrt{3}/2)$
      y.add ← *y*[*y.length* − 1] − 0.5
  end
  return [*x*, *y*]
end

(g) Yau's method

4. Implementation and Outcome

The MERN stack consists of MongoDB, Express, React and Node. This is one of the most popular full development stacks used in industry today. According to Stack Overflow Developer Survey 2020, 62% reported to use JavaScript. The evolution of JavaScript with ES6+ adding features such as Classes and Lambdas make it a feasible backend solution using Node, offering similar functionality of e.g. Python. Node also provides an extensive ecosystem of libraries though the Node Package Manager that is unrivaled by alternatives such as Ruby on Rails. The final project solution contains roughly 7,000 lines of code spread across 44 React components, Redux functions, style sheets and backend API. This section will break down the logic into the tech stack(its justifications included), screens and features.

4.1. Technology Stack

4.2. Frontend

I will use React.js to handle client-side logic. This will make up the majority of the project codebase. Sass (Syntactically Awesome Style Sheets) will style the web app and Redux for persistent state management.

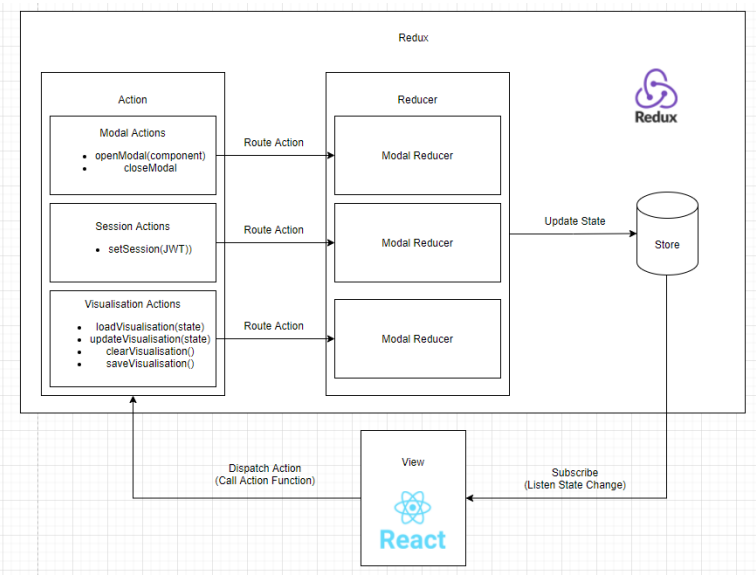


Figure 28: Diagram to show how React and Redux communicate to build the frontend

4.2.1. Styling – (Sass)

Sass is a scripting language which is complied into CSS by a module bundler, in my case, Webpack. The CSS extension language contains additional power features such as variables, nesting and modules that make development more efficient and maintainable over the long term compared to standard CSS.

Using the Modules feature in Sass, similar to the standard import in CSS we can create a natural hierarchy of styles to build on top of each other. At the highest level are "abstracts", these are fixed values relevant for the scope of the project such as colours, variables and mixins functions useful for responsive breakpoints.

Then you have the "base" level, here, styles for common elements such as forms, inputs and page layouts are stored that use the values from the abstract level. This enables code reuse as the same class can be applied to sibling elements. Code reuse is important for maintainability as changing style from the base level can easily have a global effect on the web app with minimal effort. Examples of base layouts are in the input.scss file that contains styles for input elements such as buttons, text and combo boxes across the web app.

At the component level are styles specific to individual components. These styles are unique to these components and do not have a natural higher relation to each other. An example is the titlebar.scss file that contains styles specific to the title bar and it's features such as the snapshot integration and user profile options. They also reuse features from the base and abstract levels such as "icon-button" for SVG based buttons. The hierarchy is important when compiling the files to ensure their inter-dependencies are met at compile time.

4.2.2. User Interface - (React)

React is a JavaScript library for building user interfaces. As the project is frontend heavy, a library such as React contains several key features will simplify building the user interface in a practical and efficient way.

The viriutal DOM is one of reacts core features. Essentially, when a modification to the DOM (Document Object Model) occurs, it first updates the virtual DOM in memory. Next it compares the differences against the real DOM and updates only what has changed. This is great when creating a dynamic web app as only elements that have been changed either after external user interaction or internally, will be re-rendering

without re rendering the entire screen. The outcome is a far smoother and performant user interface. Example of where it will it's used is the protein selector for the Dot Plot and Line base visualisation screens. Users can select protein items form the database, resulting in items list and visualisation being updated and re-rendered independently, far better in performance.

React divides UI elements and it's logic into components. These are separate JavaScript files that encapsulates all the component's code, breaking down the complexity of the project into logical sub-problems and encourages code reuse. An example is the Inventory component which displays a list of items that can be filtered by name and sorted. The component gets reused thought several parent components such as the Data management, Networks visualisation, Edge Binding Visualisation and protein selection components. Code reuse centralises logic saving significant amount of code maintenance work and ultimately better system stability. Figure 29 demonstrates how the logic of a screen is broken down into sub-components.

Figure 29: React component breakdown example with the Dot Plot screen.

Similar to Sass for CSS, JSX is a syntax extension to JavaScript that allows developers to effectively, describe the UI. High level JSX syntax shares similarities to HTML which is then compiled to low level ES6 JavaScript JSX by a library such as Babel. JSX embraces the fact rendering logic is coupled with UI logic and allows JSX DOM code to be written in the scope of a component JavaScript file. JSX is used in the project to describe all UI components.

Not to be confused with Redux states which are global, states in React enables pre-

sentient state management at the component level. This is important for the lifecycle of the UI component that may change over time. For example, visualisation screen components have state that must contain it's selected protein samples where the selections can change over time by the user. These changes to states will re-render all UI elements of the component that refer to it instead of the whole component, further optimising performance from what was mentioned in the Virtual DOM section.

4.2.3. State Management – (Redux)

Redux is a global state management "container" for JavaScript apps. The purpose of Redux for this project is to have persistent state for user session data, to set a React modal component to be displayed and to save the current visualisation state when creating snapshots. Effectively, React when combined with Redux allows a persistent store in state that enables the same logic and data to be shared across multiple actively listening components and routes. State management is what differentiates a website from a web app. There is a global overview of how Redux is implemented for the project.

- Modal State: The openModal(<component>) action is called, or "dispatched" in Redux terms, where the argument is a React component. For example in the Dotplot screen; openModal contains the SearchModal.js component with a parameter onSelect pointing to a function which handles logic when a protein is selected in the SearchModal.js by the dispatcher, DotPlotScreen.js The closeModal will hide the modal setting the component to null. This allows communication between the modal components and the visualisation screen components.

- Session State: The session reducer listens for the setSession(<token>) action where the parameter is a JWT (JSON Web Token). I will go further in depth of how JWT and user session work in the security section. When the action is dispatched, the user information such as their username is decoded and put into a global presentient context. This allows any API calls to refer to the user when retrieving/storing their information to and from the database. Furthermore, UI elements such as the title bar can now show the user's information. Using a state-management library such as Redux is to only way to achieve this using React.

- Visualisation State: Stores the current configuration of the visualisation screen. This includes data such as the protein/interaction samples the user has selected

and the visualisation settings. By centralising this information in a global state, it enables the titlebar.js component which contains the snapshot buttons to obtain the current visualisation data needed to create the new snapshot. This is an excellent example use case of the benefits that Redux provides.

4.2.4. Visualisations – (D3)

D3 (Data-Driven Documents) is a JavaScript library to "bring data to life" using HTML, SVG and CSS. The library takes a data driven approach to DOM manipulation where data is translated into logic that modifies SVG elements. Logic are transformations that maps to data, this makes the D3 library extremely low level compared to other higher-level Framework approaches such as Victory or Rechart. The simple and low-level nature of D3 enables finer control and optimisation, giving more freedom to execute the specific niche visualisations as required for this project.

There are 5 common stages to visualisation:

1. Preparing the data: Raw sequence data is processed using the respective functions as mentioned with the pseudocode. This data is then formatted to a JavaScript an array of object with x and y coordinates for plotting. For example the 2d array output of the Gates function is mapped to an JavaScript array $\{x : x\_value, y : y\_value\}$ to simplify plotting.

2. Create scales: Scales are D3 functions that map an input into to an output. Specifically, for this project Scales can map a data point to a view position coordinate and or a value into a colour. For example, in the Dot Plot visualisation, we have scaleLinear functions to process x and y coordinates for dots (SVG circle elements) and the heatmap (SVG rect elements). Furthermore, a scaleLinear function is used to obtain a colour output from the number of matches found in a region for the heatmap.

3. Plot elements: using the output from the scales, we can plot SVG elements such as circles and rectangles to a position in the screen to build the visualisation.

4. Define mouse events: We can assign a function to trigger on specific mouse events such as "mouseenter" or "mousemove". For example, in the Network visualisations, a mouseenter event is triggered when hovering overing over a node which

highlights it's interactions including respective adjacent nodes. Mouse events add tremendous value to the visualisations and allow intuitive interaction to obtain specific data easily.

5. Clipping: Unwanted elements are clipped out of the svg such as lines and data points out of bounds from the axis domain and range. This can occur when zooming and panning the visualisation for example in the the Network and Randic's method. Clipping these elements also improve the overall performance when interacting especially with larger datasets that render many data elements concurrently.

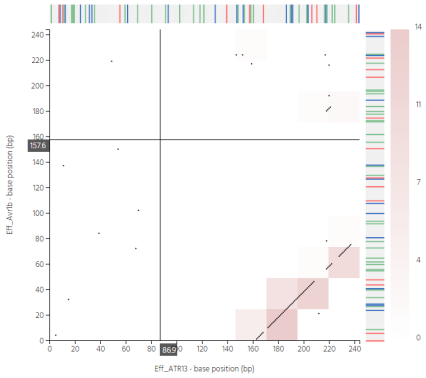
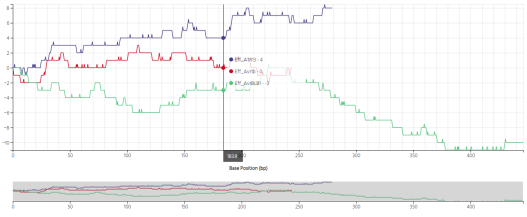
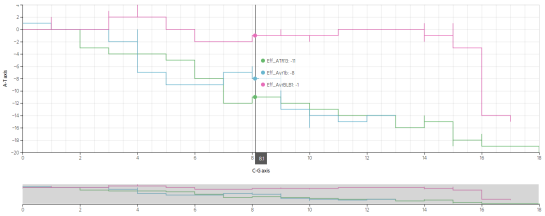


Figure 30: Dot Plot Visualisation

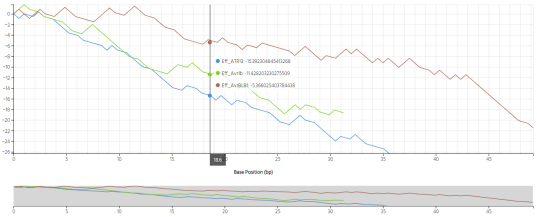
The Dot plot uses the D3 scaleLinear function to map, dots, axis labels, base sequence and heatmap area data positions to the screen position. The mouse events are called to render the axis guidelines with the axis values obtained by inverting the output of the scaleLinear function with the pointer coordinates.



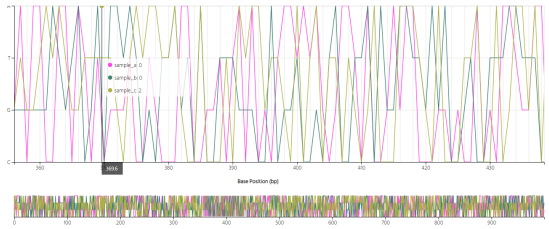
(a) Squiggle Visualisation



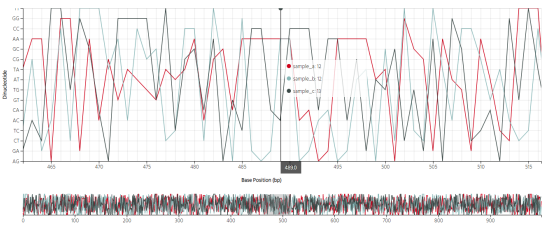
(b) Gate's Method Visualisation



(c) Yau's Method Visualisation



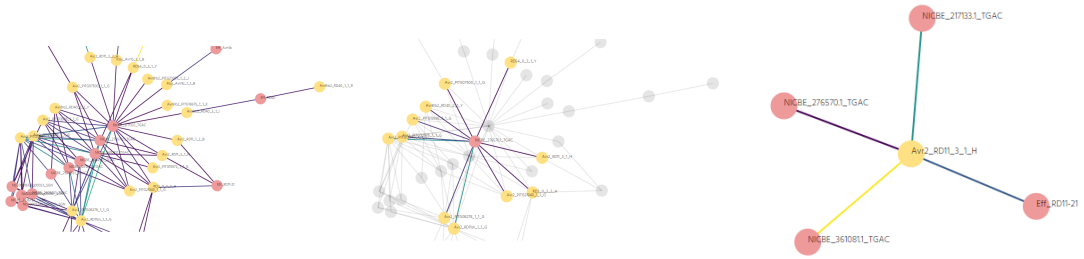
(d) Randic's Method Visualisation



(e) Qi's Method Visualisation

Figure 31: Line Based Visualisations

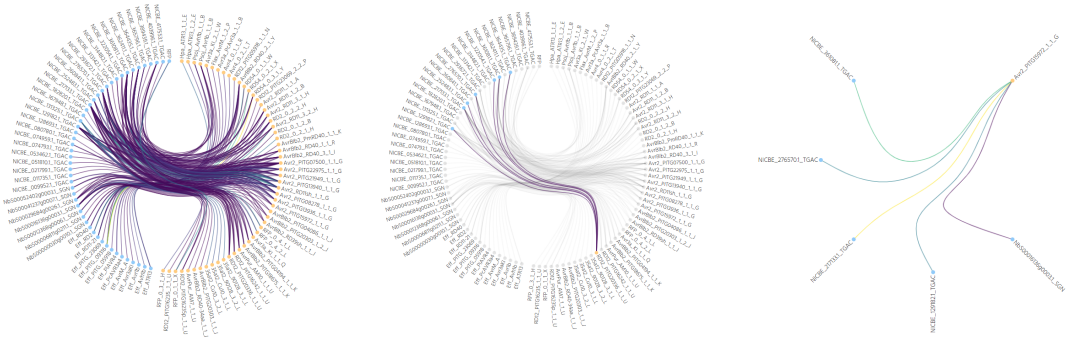
The line based visualisations use the scaleLinear function to same effect but the main and mini maps have different scales as the main can be zoomed and panned. The zoom ability works by obtaining the scroll amount (z dimension value) using D3's zoom functions then calulating the new upper and lower bound. This range is then feedback into the the scale followed by re-rendering of the lines to update the visualisation. Panning works in a similar way using pointer change as the offset value to calculate the new scale domain. Values for the tool tip Y values and nearest line point when hovering the mouse are calculated using the bisect D3 function. This takes advantage of the fact the domain values are already sorted in order by calculating where the scale inverted mouse x coordinate can be inserted (in order) revealing the index of the nearest data point.



(a) Default state (b) Hovering over node (c) Clicked on node

Figure 32: Network visualisation

The Network visualisation makes full use of the `force` D3 library to simulate physics while moving and positioning nodes. The user can move data around facilitating exploration of interactions. In order to do this effectively, connected nodes must follow too as explained by Norman's principle for good mapping. Therefore, a physics system is essential. `force` functions are called during mouse drag events for each node. The formatted link data is inserted into a JavaScript set which allows constant time access to its elements. Using a set is an important optimisation since when hovering or clicking on a node, we need can obtain its related nodes fast from an array of thousands of interactions. The zoom and pan feature works the same as explained in the section above.



(a) Default state (b) Hovering over node (c) Clicked on node

Figure 33: Edge bundling visualisation

The Edge Bundling visualisations use a tree data structure with D3's `cluster` function to store nodes as dendrograms through D3's `hierarchy` function grouping baits

and proteins separately. The `cluster` can the be directly iterated by the `selection` function to execute the render logic for nodes and links. Sets are used retrieve adjacent linked nodes in constant time as in the Network visualisation. Finally, to position the nodes circularly, the following equation is used to transform its current x and y coordinates: $x = y * cos((x-90)/(180*\pi))$ and $y = y * sin((x-90)/(180*\pi))$.

4.3. Backend

4.3.1. Database – (MongoDB)

I have implemented a Non-SQL database solution in MongoDB over a traditional SQL solution such as PostgreSQL. The main advantages of using MongoDB for this project are

- Flexible data models: A flexible schema allows far easier modification of the database which simplifies development. For example, my initial plan was to different protein models from bait, however, after experimentation due to MongoDB's flexible schema, I was able to determine it would be best to abstract them as one object. Similarly, adding dimensions feature to the schema required minimal effort while SQL solution requires table modification.
- Horizonal Scaling: SQL database requires vertical scaling (adding more hardware to servers) while horizontal calling allows spreading load across multiple, modular, similar servers. This saves a lot of money over the long term when greater data/performance is required to facilitate a growing userbase. While it's outside the scope of this project now, if in the future it where to go into production, storing just single genomes samples can take up multiple gigabytes and increased processing time making horizontal scaling critical to the feasibility of the web app.
- Faster Queries: SQL databases are normalised in a way that complex queries require e.g. JOINS would need concatenation and formatting of multiple tables. Using MongoDB, I can optimise queries with the choice of schema. For example, storing interaction object's protein bait relationships as a array of objects will save extra seeks querying a composite table when compared to a SQL solution.

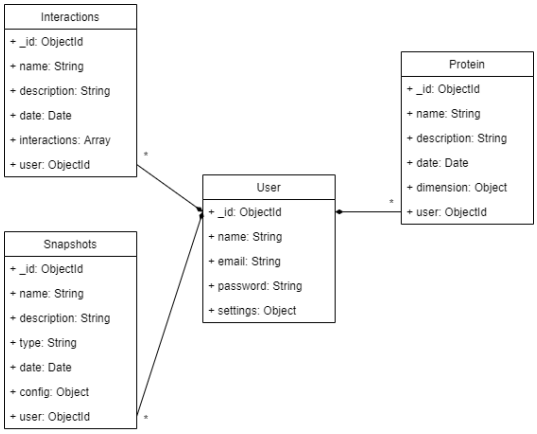


Figure 34: Database UML diagram

here is an overview of the database structure using a UML diagram. There are 4 documents (tables) in the collection (database). The Cluster is situated on a local mongoDb server, a production environment would be a cloud solution such as mongoDb Atlas. The life cycle for Interactions, Proteins and Snapshots are dependent on a User and naturally from a 1 to many relationship. Furthermore, this means that the all of the user's information will be encapsulated into their account for privacy. Encapsulation is an important requirement because datasets can be sensitive private information such as the research datasets, modelled in the scenario for Kelly Rowe.

The all ObjectId fields are indexed for fast data retrieval. As mentioned before, one of the key advantages of using MongoDB is it's use of arrays and nested document object as property attribute types for documents which completely negates need for composite tables found at normalised SQL databases. Snapshots use a nested object to store the config. The JSON format used by MongoDB document models mean that React state objects can be directly serialised into the database as an attribute and deserialised back into the frontend React state later. This is extremely powerful and developer friendly as little to no validation or loops need to be written.

Proteins also contains a nested document which stores custom dimensions. Dimensions are user set attributes for a protein that are not necessarily common across all proteins or optional properties. An example of a dimension is the protein weight property that is found in the TSL dataset for non-bait proteins. Dimensions can be used in the Network and Edge Binding visualisations for filtering the database.

The backend will handle server-side logic using Node.js. Majority of backend logic

will be database interactions for this, I will use Express to build a REST API. The API will consist of GET requests for data retrieval, POST to import data and DELETE to remove individual records. Requests will be lean using optional parameters let single routes handle multiple request e.g. GET /api/protein?id=ObjectIdaccessionNumber=molecularWieght= Database request to MongoDB will handled by mongoose.js.

4.3.2. API – (Express.js)

Express is a web framework for Node that wraps HTTP requests and responses making it straight forward to map URLs to backend server-side functions. Effectively, an express server runs in the background of the server side listening to HTTP requests and responses with defined functions for the respective URL forming an API.

Models (MongoDb document objects) and Routes (HTTP endpoints) have been split into sperate .js files to separate logic and easy maintenance.

Figure 35: MongoDB schema for users (left), Bulk import API route for proteins (right)

For example, this is the user model based on the mongoDB user document. Using mongoose, a schema can be formed containing validations to protect the document's data integrity.

The mongoose schema can be imported by the Routes script to interact with the database. In this example we look at the /add endpoint for protein models. The post requests accepts an array of proteins. This allows for one or multiple items to be accepted in the same route. Items are mapped to update functions. This allows existing

proteins to be updated and well as appended, again this adds greater versatility to the endpoint. Finally operations are executed using a bulkWrite function, this executes the update functions in parallel threads which is substantially faster than a serial loop alternative. The schema defined previously in the model will reject any documents that fail it's validation requirements to protect the collection's integrity.

Errors are handled with .catch callback function which responds with a HTTP 400 status code and error message so that I can be handled appropriately by the frontend. Similarly, on success, the .then callback returns a HTTP 200 status code to inform the frontend. Middleware allows a function call to be made to pre-process an Express endpoint request. In this example, verifySession middleware is called to validate the user session provided in the HTTP header. How this works is explained in the security session. If a valid user session is not found in the HTTP header, the requests is rejected with a HTTP 401 code. This is very important as it protects endpoints that modifies/retrieves user data by preserving its confidentiality.

4.4. Security and User Authentication

The project implements a token-based security model using JWT (JSON Web Tokens). JWT open source standard (RFC 7519) security token that encodes data such as identity and other confidential information to be shared across security domains. For this project JWT is used as a means of validating the user session.

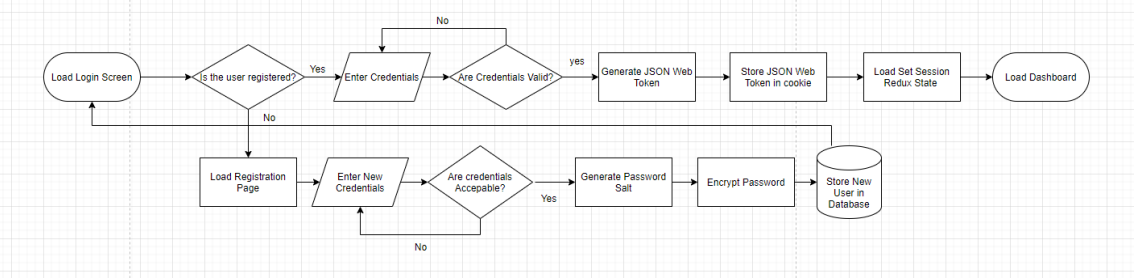


Figure 36: User authentication overview

4.4.1. JSON Web Token (JWT) and Cookies

When login credentials are valid, a new JWT is created containing the user id and username to provide the session context. This is singed using a token secrete only known to

the backend meaning any unauthorised modification will render the token invalid. The JWT it's self is a string which gets passed to the response header where it is analysed by the frontend and stored in the cookie. The cookie has a set expiry of 7 days to prevent session Hijacking attacks. Furthermore, a new JWT is created for each successful login to prevent session fixation attack. Now, data from the cookie is decoded and stored in the session Redux state. The user can now access the web app and their data. When logging out, the cookie is cleared and the user must login again to obtain a new token.

4.4.2. Public and Private Routes

In the frontend, there are two React wrapper components for the React-router routes. These are PrivateRoute.js and PublicRoute.js. PrivateRoute.js only grants access into the route if a valid session token exists in the cookie. It then decodes the JWT and stores it's data in into the session state if the state is empty. The PrivateRoute protects all user-based screens. PublicRoute grants access to public domain routes such as the login and register screens where session cookies are not required. These create a system of protecting the routes against unauthorised access.

4.4.3. API Protection

When requesting user information from the backend database, the appropriate API endpoint is called containing the user's token from Redux session state. A middleware function, sessionVerfier.js, in the backend is called to verify authenticity of the JWT from the request. If the token can be decoded using the secrete token, the user id stored in the JWT is forwarded to the API endpoint for context allowing access to user's data. If the token cannot be decoded, an HTTP error 400 blocks entry to the endpoint. Similarly, if no token is provided, a HTTP 401 is returned.

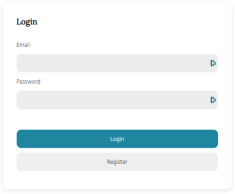
4.4.4. Encryption

Upon user registration, passwords are stored as hashed strings instead of plaintext. By encrypting the password, it protects the user credential's confidentiality e.g. form being stolen encase of a database breach. The hashing algorithm used is Bcrypt. Compared to other popular hashing methods such as SHA, Bcrypt is fast and harder to decrypt using multi-processing hardware such as GPUs. Passwords are also hashed with a Salt. Salts

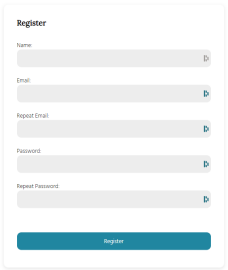
are randomly generated strings that add to complexity and uniqueness to the hashed output. This means that same passwords will be stored as different strings in the database preventing dictionary attacks.

4.5. Screens

4.5.1. Login and Registration



(a) Login screen



(b) User registration

Figure 37: Unit Testing

The login and registration screens are the only public routes for the web app meaning they are accessible by anyone without need be authenticated. Their internal process are highlighted in the security section. These forms a validated using a schema as explained in the validation section. Forms communicate with the backend API using `axios` library for post and get requests.

4.5.2. Dashboard

Using `axios.js`, get requests retrieve count for all of the user's data for summary and `moment.js` formats the time/date for the welcome message. `<Link/>` from `react-router` redirects the user to their desired page by appending to the history stack. Similarly `<NavLink/>` does the same for the `sidemenu` while also providing alternative css if the respective URL matches the current location. The user information for the `sidemenu` and `titlebar` is retrieved from the `session store` as the JWT is decoded from the cookie.

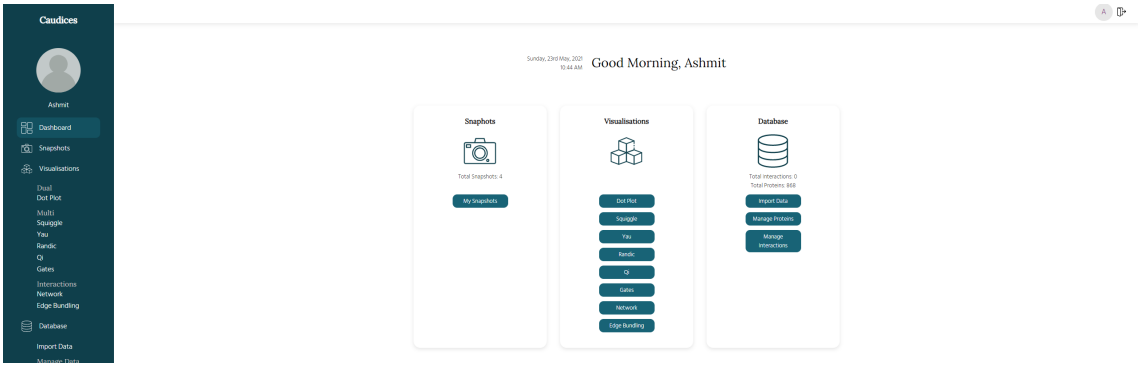
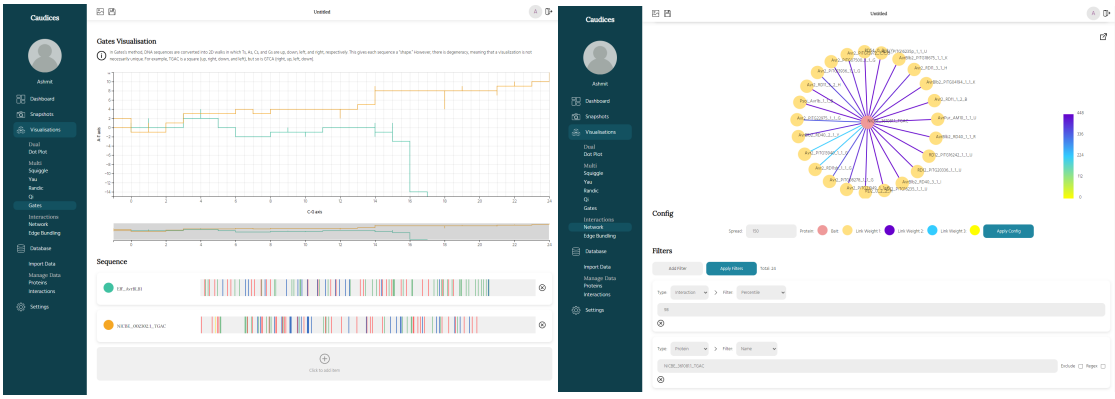


Figure 38: The Dashboard screen

4.5.3. Visualisations



(a) Gate's method visualisation screen

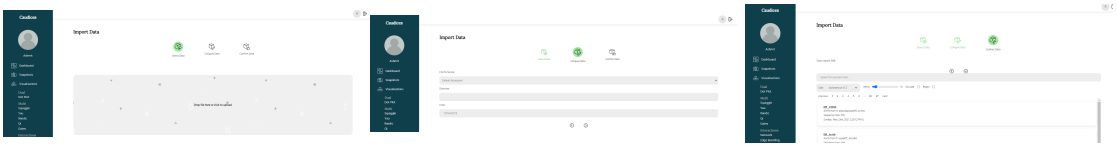
(b) Network visualisation screen

Figure 39: Example visualisation screens

Visualisation use the `ProvisionalItem` component to trigger a modal containing the `Iventory` component to select data for visualisation. This updates React dataset state to render the respective visualisation. Changes to the configurations will update the config React state. re-rendering the visualisation. The Network and Edge Bundling visualisation also feature a filter system to brakedown the data. This overcomes the challenge of visualising 100,000 interactions by only showing the most relevant data. By default, 98th percentile interactions of molecular weight to interaction strength is shown, isolating the strongest interactions. The filter system uses a tree data structure where parent nodes (protein, baits or interactions) have child nodes for different filter

metrics each containing it's filter function as leaves. These functions are placed in a pipeline React state (array of filter functions) which are executed in order, filtering interaction data down.

4.5.4. Data Import



(a) Data Import Screen 1 (b) Data Import Screen 2 (c) Data Import Screen 3

Figure 40: Data Import Screens

The screen component for data import is split into 3 components as described in the design section which breaks down the complexity. The user selects a data file either using a open file dialogue or drag and drop. The file extension is automatically determined and user is taken to the appropriate configuration screen containing the TSL dataset defaults set for .fasta files and .csv interactions files for simplicity. The user can see all data before importing using the Inventory component with facilitates search and sorting as explained in the data management section.

4.5.5. Data Management

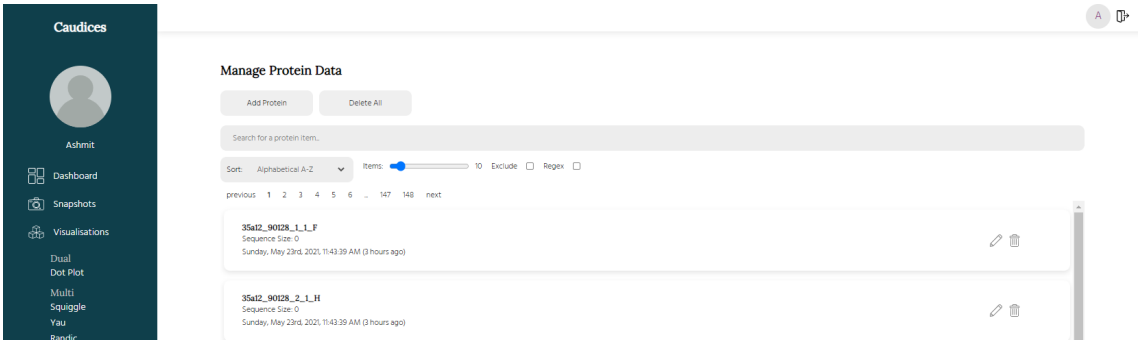


Figure 41: The Dashboard screen

Data selection, modification and deletion is handled by the Inventory component. This component is reused in several pages including the data management, snapshots

and search screens/modals. The Inventory takes props for array of items, the item type (e.g. protein/interaction/snapshots) and if actions such as select/delete are allowed along with its callback function to trigger. Item list of the respective item type components are then rendered. Users can search for specific items names using the search bar or a regular expression input. Furthermore, items can be filtered using the .sort function which takes in the appropriate comparator function. Paginating list is important to prevent too many item components being rendered at once, this improves performance and the TSL which would slow down the UI.

4.5.6. Settings

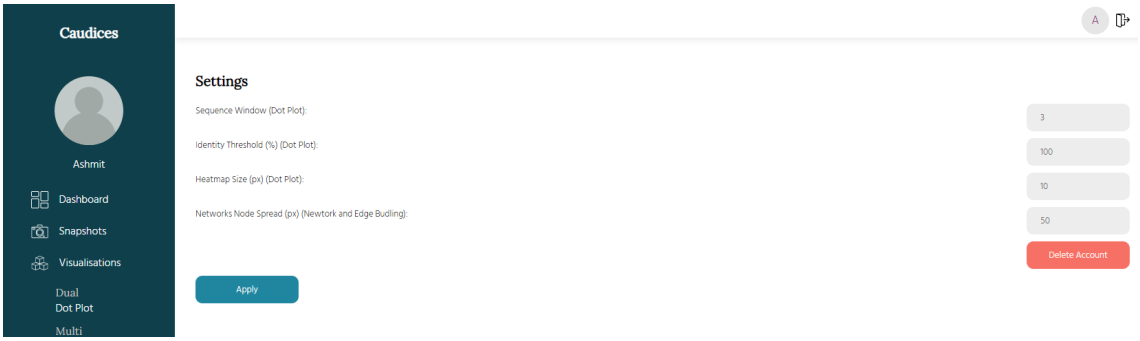
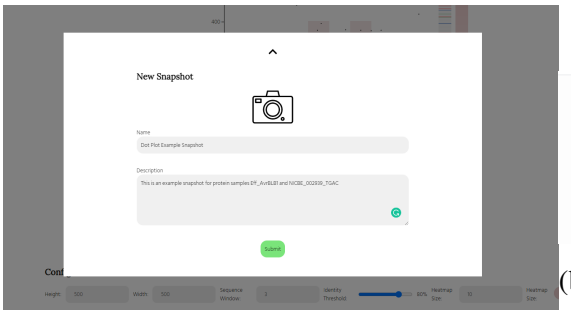


Figure 42: Settings screen

We can find and update default configurations of visualisations in the settings screen. These communicate with the Redux `session` state to update the configuration values. These values are then accessed by the respective visualisation screens when rendering the SVG. The delete account button removes the user from the database and their associated data then logs them out by deleting their JWT session cookie.

4.6. Features

4.6.1. Snapshots



(a) New snapshot modal

(b) Title bar after loading snapshot

Figure 43: Authentication Screens

A user can save the current state of the visualisation by creating a snapshot using the save button at the title bar. This will create a snapshot document in the database which stores the visualisation's configuration and dataset as serialised JSON object, taking advantage of MongoDb's object model. Users can select a saved snapshot from the Snapshots screen which then renders the visualisation using the deserialised configuration and dataset from the snapshot document. Users can also edit the name and description of the snapshot at anytime.

4.6.2. Modals

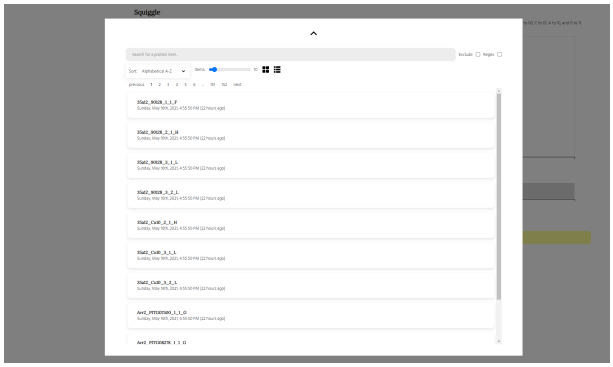
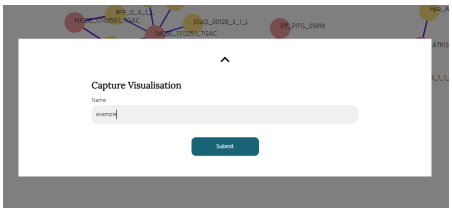


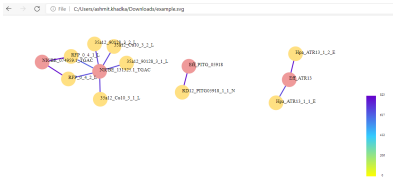
Figure 44: Search modal used to select a protein item for a visualisation

Modals are hover elements that displays a component on top of the current screen component for additional functionally. Modals are used in the project to search for protein/interaction item to visualise, to create/update snapshots and protein items or to display the capture component. This is more convenient and better for usability as the user can stay on the current page to perform these actions rather than being redirect to another screen. Modals work by dispatching an openModal action function which displays the React component (with props) passed in as an argument as a child component to the modal. The closeModal action then hides the modal setting it's child component to null.

4.6.3. Captures



(a) Capture modal



(b) Output SVG image

Figure 45: Unit Testing

Captures is a feature that enables the user the save the current visualisation as an SVG image. It uses a modal to display the capture form on top of the current visualisation when clicking the capture icon in the title bar. Here, the user can enter the image name. As D3.js renders an SVG elements for visualisation, the capture feature works by duplicating the SVG properties to a .svg file.

4.6.4. Validation

Data validation of forms or structured inputs both in the frontend and backend API such as user registration and updating protein items is handled using the Joi library. This is a great library because it allows definition of fixed schemes (Joi objects) with requirements for each property to effectively validate user input. Furthermore, advanced features such as comparisons used in the registration form to compare retyped email can directly be set in the schema to save additional maintenance work.

For the forms themselves, the React Hook Form library is used. This is another very popular and great library because it preserves React's core client side philosophies by keeping things asynchronous. Additionally, it can be used with Joi to keep the data validation method consistent.

4.6.5. Error handling and Notifications

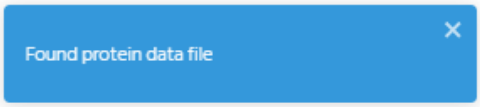


Figure 46: Notification component when fasta file is selected in the data import

The web application features alot of asynchronous calls particularly with promises used for API calls through the axios library. The .then and .catch functions handle the errors. Backend API calls respond with HTTP status codes (200 for success or 400 for errors) containing a message body for error description. The response is then handled in the frontend, by notifying the user using the React Tostify library. Simmilar to Redux, this library creates a store to which you can add notifications by calling the tostify function. This forms a stack data structure where notifications are pushed on top and shown then poped out after the timer. Using notifications provides good feedback to the user which is important when interacting with asynchronous functions.

4.6.6. Linking Visualisations

Interaction data from the network and edge bundling visualisations can be visualised using the other visualisation methods directly without needing to manually add the data. This works by passing the selected data items into an array as a parameter from the URL of the screen. The array is sanitised by default and converted into JSON to retrieve the protein samples. This is a very powerful feature that enables easier analysis of the interactions to identify DNA sequences regions of similarities.

4.7. Code and Documentation

The JavaScript codebase follows Google's JavaScript Style Guide for File structure, formatting and naming. This creates consistency through out the code base using a logical, predicable code structure that makes future maintenance straightforward.

Even though JavaScript makes up the majority of the project's codebase, many of these rules are transferable across multiple language domains specially the formatting rules for Sass. One exception for the rule is the use of tabs for indentation rather than the recommended double space. This is simply because tab indentation is a single button press and provides more visual separation across all popular IDEs.

Comments are placed at the top of the .js to explain what the file does. For React components the purpose of it's props and the output render element are explained. Comments are also placed at the function level where relevant, again, to explain what they do. It's important to add these comments because the project uses an array of libraries that have bespoke functionality beyond vanilla JavaScript.

A formal documentation is written in the README.md file that engages in a deep dive into what the visualisations are and how they are implemented along with the system architecture. The .md file is formatted with headers, it's intended for use with a repository hosting service such as GitHub. This is important for future maintenance of this project.

5. Testing

5.1. System Testing

5.1.1. Method

Jest is the industry leading testing framework for JavaScript used to in the project to perform unit testing of core functions for system verification. Jest is a great framework because it combines both the test runner and assertion logic similar to the unittest library for python. Scope of unit tests are outputs of core visualisation logic functions. By calling the function using a known input and comparing against its respective output, we can determine if the processing is correct. i.e. entering a DNA sequence string and comparing with its known Cartesian output. The unit tests for visualisation logic where written first before implementing into the frontend. This has helped a great deal when

debugging in development. All tests can be found at `system.test.js`

5.1.2. Results

```
test("Dot Plot method long sequence.", () => {
    const sequneceA = "CGCTCGGGTGTAGCAGAGAAGACGCCTACTGAATTGTGCGATCCCTCCACCTCAGCTAAGGTAGCTACCAATATTTAC
    const sequenceB = "ACAGTCATATTGGGGTGCTCCTAAGCTTTTCCACTTGGCTGGGTCAGCTAGGCCTCCGTGCCCGGAGTTTCGGCGCAC
    const sequenceWindow = 8
    const identityThreshold = 100
    const result = tools.getDotPlotPoints(sequneceA, sequenceB, sequenceWindow, identityThreshold)
    const expected = [
        { x: 77, y: 855 },  { x: 89, y: 672 },
        { x: 90, y: 673 },  { x: 113, y: 470 },
        { x: 269, y: 614 }, { x: 332, y: 774 },
        { x: 537, y: 798 }, { x: 538, y: 799 },
        { x: 539, y: 800 }, { x: 619, y: 468 },
        { x: 727, y: 632 }, { x: 786, y: 900 },
        { x: 887, y: 105 }, { x: 888, y: 929 },
        { x: 895, y: 806 }
    ]
    expect(result).toStrictEqual(expected);
});
```

(a) Example unit test for the Dot Plot logic

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

$ jest
 PASS  src/scripts/
  √ Gate's method normal sequence. (4 ms)
  √ Gate's method lower case sequence. (1 ms)
  √ Gate's method mixed case sequence. (1 ms)
  √ Gate's method no sequence. (1 ms)
  √ Yau's method normal sequence. (1 ms)
  √ Yau's method lower case sequence.
  √ Yau's method mixed case sequence.
  √ Yau's method no sequence. (1 ms)
  √ Squiggle method normal sequence. (1 ms)
  √ Squiggle method lower case sequence. (1 ms)
  √ Squiggle method mixed case sequence.
  √ Squiggle method no sequence. (1 ms)
  √ Randic's method normal sequence. (1 ms)
  √ Randic's method lower case sequence. (1 ms)
  √ Randic's method mixed case sequence. (1 ms)
  √ Randic's method no sequence.
  √ Qi's method normal sequence.
  √ Qi's method lower case sequence.
  √ Qi's method mixed case sequence.
  √ Qi's method no sequence. (1 ms)
  √ Dot Plot method long sequence. (265 ms)
  √ Dot Plot method no sequence.
  √ Dot Plot method complex sequence. (8 ms)
  √ Dot Plot method mixed sequence lengths. (20 ms)

Test Suites: 1 passed, 1 total
Tests:       24 passed, 24 total
Snapshots:   0 total
Time:        2.847 s
Ran all test suites.
Done in 4.03s.
PS C:\Users\ashmit.khadka\Documents\Projects\CMP-6013Y\frontend>
```

(b) Unit test results

Figure 47: Unit Testing

The final system passes all 24 unit tests. Furthermore, performance in terms of completion time for tests are fast demonstrating good implementation the initial pseudocode design. The results provide confidence in the system's stability and readiness to be launched into production.

5.2. User Testing

5.2.1. Method

For user testing, participants where asked to interact with the web application and their feedback was obtained for system validation. Participant A, Joel Harrison, is Biology graduate. Participant B, Lukasz Milik is a Computer Science graduate. Due to the complexity of setting up the system remotely, participants where invited locally for testing. This enabled setting a controlled environment for user testing providing better visibility and easier feedback. Participants setup a new account and where asked to engage with the web application's core features.

5.2.2. Results

Users found registration and login straight forward, they liked the default configuration settings in the data import. Selecting the data for visualisation was simple and clear what they had to do. Participants liked the interaction features especially in the network visualisation. Both expressed a way to save the visualisation as a possible suggestion. Joel suggested an image while Lukasz proposed a way of saving the visualisation "state". Joel also made observations to include colour intensity scales and heatmap overlay for the Dot Plot. Overall, the feedback was very positive, with users picking up features intuitively. Using their suggestions, the MoSCoW requirements have been updated including the features in the could have for the next development iteration.

6. Limitations and Extensions

6.1. Machine Learning

A machine learning algorithm can be implemented to predict the probability of a given protein interacting with a bait. This model can be based on training of previous protein-bait interactions using their base sequences as data. If successful, the web application would save much lab efforts to obtain the interaction data. This it would streamline and automate much of the process to create a fast and inexpensive way of finding interactions. In order to perform Machine learning, it would require training from an enormous dataset which may not be practical. Finding interactions of new protein samples may be out of scope the current model. Therefore, a realistic implementation could complement the current process.

6.2. Data Managment

The number of Interactions can be extremely large (over 100,000 interactions in the sample dataset). This is why currently, the interactions are imported from a data file. However, adding or modifying interactions on top of a existing imported dataset can be useful if required. This would be faster than re-importing a new dataset.

Datasets could be shared among users. Currently, user data is encapsulated within their user account. A share feature can be implemented enables sharing of protein or interaction item with another user. This can be useful in a research environment where

multiple researchers are collaborating together. A feature to search another user by email (public unique identifier) could add value by saving time needed to export and transfer data. By simply changing the user attribute in the MongoDb from a single ObjectId to an array of ObjectIds, we can create a one to many relationship between the data and user. Further extension can be to share a Snapshot in the same way.

6.3. Visualisations

The Dot Plot visualisation can further extended by appending multiple sequences together along the axis. This would allow direct comparison of several DNA sequences in one view which makes pattern recognition far easier. For example, the visualisation from FlexiDot visualisation bellow, we can see 36 Dot Plots concatenated together. Because the D3 based Dot Plot is a React component, implementing this feature can be as straight forward as positioning the components in a CSS grid and creating a global X and Y axis components.

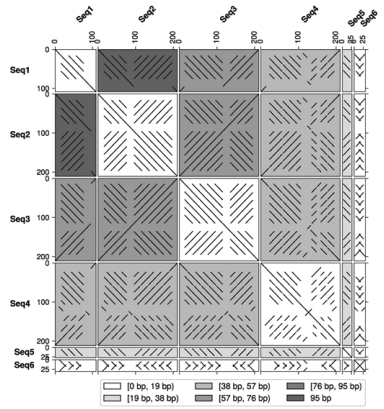


Figure 48: FlexiDot's multi dot plot visualisation option

The Edge Bundling visualisation can be developed further by utilizing it's hierarchical groupings. Levels in the dendrogram could represent a layer of abstraction for example, the genome family type. This would provide further analysis to the visualisation allowing the user to recognise relationship patterns between different abstraction levels. The example bellow demonstrates the hierarchical grouping of "leaf" nodes

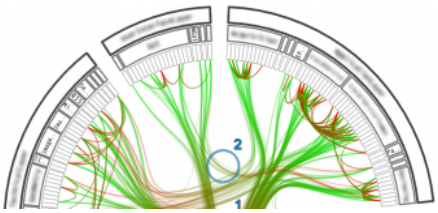


Figure 49: Grouped Edge Bundling

7. Evaluation

I will base my success criteria on comparing the project outcomes against the requirements defined in the design section using the results of my testing. The project delivers all the visualisation requirements set in the must have section. Using D3, SVG elements are rendered into the screen dynamically based solely on the data chosen by the user. Best practices for the D3 has been used by taken full advantage of several key features of the library. These include appropriate use of different scales for discrete and continues data, zoom function for mouse event transformations, tree data structures for optimal data storage/retrieval and force functions to simulate psychics. As a result, when compared to some of the current solutions such as dnavisualization.org's visualisation for Squiggle and Qui methods, the project solution provides a much smoother interaction and performance using the same dataset. This justifies using the current technology stack especially D3 as the rendering engine. Visualisations also contain extra features not found in existing mainstream projects such as the heat map overlay and base sequence adjacent to axis found in the Dot Plot.

Visualisation are also interactive to add value when exploring data. The Network and Edge Bundling visualisations, for example, calculates and highlights adjacent nodes with relationships on mouse hover and clicking the node it's self automatically adds a filter to show it's relationships in isolation. Through the user testing, these interactions have shown to be intuitive with users demonstrating good mapping between actions and affect.

Customisation options can also be applied to modify the visualisations for specific use cases for example the window size of the dot plot. Changing colour facilities cases of colour blindness and the customisable filter system for network base visualisations divides a large data set to manageable junks.

Users can import protein data using the industry standard FASTA format for nucleotide/amino acid sequence text representation. Interactions are parsed in .csv format with data stored in a 2d grid system for reduced file space. Furthermore, users can configure import setting for example, selecting the FASTA file's many NCBI identifier variations, meeting the requirements set in the design. The data can be viewed for confirmation before importing using the Inventory component. This component is reused throughout the web application for managing data including sorting, filtering and searching options to simplify finding items and actions to modify and delete items where appropriate meeting the data management requirement. In testing, users had no resistance finding items and making changes.

Snapshots and Image exports where suggested made from the user feedback placed in the could have requirements. Using the the modal system and Redux, the snapshot feature has been fully implemented along with the image export option into a salable SVG format. The next step would be to add rasterised image export options such as PNG formats for this feature.

Overall, from my system and user testing I am confident that the web application outcomes fully meets the all must have and should have requirements to a level of good quality. Furthermore, the could have user suggestions have also been implemented in the second iteration. For future work I would like to develop further, the Edge Bundling visualisation to take advantage of it's hierarchical grouping capacity as highlighted in the limitation and extensions section. I have also taken an iterative approach to improve upon the feedback received from the user testing by implementing additional features and enhancing current ones.

Using the MERN stack was a risk due to it's high complexity. However, after investing almost 100 hours of courses from Udemy for JavaScript ES6, React and Redux, Sass and MongoDB, I have managed to implement best practices to build a fully functioning final solution. The decision to spend the early stages of the project, learning the technology has ultimately paid off.

8. Conclusion

Visualisation tools are necessary to bridge the gap between large, complex data and analysis. The project outcomes manage successfully to breakdown the protein and bait dataset provided by TSL into digestible and exploreable visualisations which can be

used to fuel research into plant disease. Furthermore, the system architecture is flexible enough to handle any protein data using the industry standard .fasta file format. React has proved to be an excellent framework to build advanced UI. MongoDb has shown that object orientated databse models work great when interfacing with web applications. The smooth performance of D3 compared to existing solutions demonstrates why it's the most used visualisation library even today.

The project tech stack as helped gain first class honours for other modules with use of React and in HCI coursework 2 and D3 in the Advanced Programming Techniques and Concepts coursework. Additionally, I have integrated skills from other modules into the project such as security features from Developing Secure Software. Using a cloud database solution such as MongoDb Atlas and Saas web hosting platform such as Heroku, the web app can be deployed online straight away. It can serve as an open source alternative to current expensive tools used in industry and academia or a more complete solution compared to the existing open source options as explored in the literature review.

References

Cabanettes, F. and Klopp, C. (2018). D-genies: dot plot large genomes in an interactive, efficient and simple way. *PeerJ*, 6:e4958.

CIA, C. I. A. (2020). The world factbook 2020. *The World Factbook*.

Gates, M. A. (1986). A simple way to look at dna. *Journal of Theoretical Biology*, 119(3):319–328.

Kinealy, C. (2006). *This great calamity: the great Irish Famine: the Irish Famine 1845-52*. Gill & Macmillan Ltd.

Lee, B. D. (2019). Squiggle: A user-friendly two-dimensional dna sequence visualization tool. *Bioinformatics*, 35(8):1425–1426.

Lee, B. D., Timony, M. A., and Ruiz, P. (2019). Dnavisualization. org: a serverless web tool for dna sequence visualization. *Nucleic acids research*, 47(W1):W20–W25.

Lhuillier, A., Hurter, C., and Telea, A. (2017). State of the art in edge and trail bundling techniques. In *Computer Graphics Forum*, volume 36, pages 619–645. Wiley Online Library.

Oerke, E. (2006). Crop losses to pests. *The Journal of Agricultural Science*, 144:31.

Qi, Z. and Qi, X. (2007). Novel 2d graphical representation of dna sequence based on dual nucleotides. *Chemical Physics Letters*, 440(1-3):139–144.

Qi, Z.-H., Li, L., and Qi, X.-Q. (2011). Using huffman coding method to visualize and analyze dna sequences. *Journal of Computational Chemistry*, 32(15):3233–3240.

Randić, M., Vračko, M., Lerš, N., and Plavšić, D. (2003). Novel 2-d graphical representation of dna sequences and their numerical characterization. *Chemical Physics Letters*, 368(1-2):1–6.

Salinas-Torres, V. M., Salinas-Torres, R., Gallardo-Blanco, H., Cerda-Flores, R., Lugo-Trampe, J., Villarreal-Martínez, D., and Martínez, L. (2019). Bioinformatic analysis of gene variants from gastroschisis recurrence identifies multiple novel pathogenetic pathways: Implication for the closure of the ventral body wall. *International Journal of Molecular Sciences*, 20:2295.

Seibt, K. M., Schmidt, T., and Heitkam, T. (2018). Flexidot: highly customizable, ambiguity-aware dotplots for visual sequence analyses. *Bioinformatics*, 34(20):3575–3577.

Strange, R. N. and Scott, P. R. (2005). Plant disease: a threat to global food security. *Annual review of phytopathology*, 43.

Xia, J., Benner, M. J., and Hancock, R. E. (2014). Networkanalyst-integrative approaches for protein–protein interaction network analysis and visual exploration. *Nucleic acids research*, 42(W1):W167–W174.

Yau, S. S.-T., Wang, J., Niknejad, A., Lu, C., Jin, N., and Ho, Y.-K. (2003). Dna sequence representation without degeneracy. *Nucleic acids research*, 31(12):3078–3080.