Nomios - Analysis

Definitions

In order for the reader to properly understand my solution, they must first understand the following definitions. Each definition is aimed at introducing the reader as if they have no knowledge of Biology so they may not be particularly specific. If they want more thorough definitions they should consult more specialised texts.

> A **protein** is a group of molecules that can have either a functional or structural role within an organism. THey are made up by a sequence of **amino acids**. The shape of a protein is dependant on the amino acids and their sequence.
>
> **Genetics** is the study of **genes**. These are molecules that code for proteins that make up an organism. The total sequence of genes in an organism is called the **genome** of the organism.
>
> **DNA** (or **Deoxyribose Nucleic Acid** to give its its full name) is the molecular structure that makes up genes. It is made up of a sequence of **nucleotides**, these are made up of a **base** (either **A**, **T**, **G**, **C**) and a backbone that binds to other nucleotides to create a **nucleotide sequence**. One nucleotide sequence is usually just a single gene, but may be an entire genome or anything inbetween. Each nucleotide's base has an opposite base (A with T and G with C) which is bound to another nucleotide. This means that DNA is double stranded. A single sequence therefore is measured in **base pairs** (bp).
>
> **Protein synthesis** is the production of proteins from DNA. It is performed through processes called **transcription** (the production of **mRNA** is a readable version of a specific sequence of DNA) and **translation** (this is where mRNA is read to produce a protein).
>
> **RNA Polymerase** is a protein that produces mRNA from DNA. It binds to a sequence of a DNA called the **promoter** which lies before the beginning of a gene.
>
> **Expression** is the volume of protein produced from one gene. It can vary.
>
> Each 127bp of DNA is wrapped around a **nucleosome**. This structure made of eight proteins that are arranged in a cubic shape (2x2x2). Each protein in a nucleosome is called a **histone**. The amino acids of a histone can be edited to change its shape and therefore change the shape of the overall nucleosome. These edits are called **histone modifications**. This is important for **transcription** as **RNA Polymerase** may find it easier or harder to bind to a region of DNA based on the overall shape of the nucleosome. This for RNA Polymerase ability to bind to a region of DNA is called **affinity**.
>
> - **Repression** is where histone modifications which change the affinity of a nulceosome to decrease, cause for the expression of a gene to decrease.
> - **Activation** is where the histone modifications which change the affinity of a nucleososome to increase, cause the expression of a gene to increase.
>
> **Epigenetics** is the study of histone modifications and other effecs that change the expression of a gene.
>
> An organism's **epigenome** is its entire collection of epigenetic effects on its genome.

> A **histone modification sequence** is a set of histone modifications that work together to bring an overall effect on the expression of a gene.
>
> The **coding strand** is the sequence of bases that are replicated in mRNA during transcription. The opposite sequence of bases is called the template strand. As bases always pair with their opposites, the template strand is used to recreate the coding strand in mRNA form. A DNA sequence is represented as a string of letters (each letter representing a different base) with the base sequence being derived from the coding strand.
>
> This is a key for purposes in database tables:

Key Symbol	Definition
%	This will contain a unique ID for each record in the table. When a new record is added, the integer value will increment based off the value of the previous record. This is what makes each record's tid value unique.
$	This contains the unique ID of the user who created this post/started the thread. This is useful for retrieving information from the user, such as their first name that will be displayed to track messages to users in a thread.
¬	is is a string that contains the actual body of the original post in the thread.

The Problem

There are two major problems within the scientific and epigenetic communities today:

1. There is a limited amount of inter-disciplinary and international research within the scientific community. Research only takes place across the globe within major pharmaceutical organisations such as *Amgen* and *CERN* because only these large, multinational organisations have the resources to recruit scientists in the world and bring them into contact through common research. Even with *CERN* research is limited to one site in Switzerland, which means that scientists must commit to travelling frequently or emigrating entirely, which some scientists will not want to do, as they may not want to leave family and friend behind This limits the amount of collaboration between scientists, as they instead turn to local universities for research sites. This means that less of the brilliant minds of today can work together to research into some of today's most pressing questions. This is limited even more by the fact that scientists tend to stick to their particular field and therefore collaborate with scientists in their particular department at whatever university, meaning that less scientists work with others outside their field, which is necessary to delivering breakthroughs. For example, x-rays uses in medicine, when Wilhelm Rontgen, a physicist, discovered their detection of bone structures. It is that kind of daring to use a discovery in other fields that is key to finding new applications and discoveries, and this discovery is only accelerated by the collaboration of scientists in different fields as they have different expertise that could be used to create a new hypothesis and therefore, a new discovery. This will problem will be referred to in the rest of the text as **Problem A**.
2. There is currently no way to predict the effects of an naturally occurring histone modification sequence on the expression of a specific gene. This is a problem because it means that when researching an naturally occurring histone modification sequence, scientists must obtain an epigenetically altered DNA sequence (or re-create one) that is identical to the one they are trying to find the effect of, and

then analyse its effects in the lab. This is because the combined effects histone modifications in a sequence are too confusing and takes too long to determine by hand, as there are many histone modifications in even a small DNA sequence. This is quite laborious and therefore takes a lot of time and funding to research the effect of a single histone modification sequence. This slows the rate of discoveries within the epigenetic field. What's more there is no widely accessible database which attributes histone modification sequences with causing specific diseases, which means that doctors cannot attribute someone's disease to an unhealthy epigenome. This prevents patients from getting treatment and therefore prolongs suffering.
This will problem will be referred to in the rest of the text as **Problem B**.

The groups that is affected by these problems are are all scientists (for problem A) and particularly **epigeneticists** (for problem B). Therefore my target audience consists of scientists for my solution to problem A and epigeneticists for my solution to problem B.

The Solution Outline

Nomios is a solution to both of these problems.

It will consist of two parts:

1. A forum, which is an web page that users from all over the globe can access to discuss topics and answer questions.
2. A **histone modification interpreter** (or **HMI**). Which is a tool of my invention that can determine the overall effect of histone modifiction sequences of DNA sequences, particularly genes and return the resulting effect to the user.

The Forum

How It Solves Problem A

Problem A partially exists because of the fact that some scientists are unwilling to leave their homes behind in pursuit of collaboration. By creating an tool where users can talk on the internet from all across the globe, scientists can now communicate and become involved in collaborative efforts whilst remianing at their homes. This means that each scientists can use their own lab and their own team on a very specific task as part of a larger peice of research conducted by an online group of which they would be a part of.

This group would use the forum to create a thread specific to their peice of research that would include responses that include new data on a particular task or how much progress into the research being conducted was made in a particular week.

A forum would also remove the need for a large multinational organisition to be neccessary for international research. This is because scientists would be able to come into contact with each other from across the globe through communicating on the forum rather than through common research forced onto them by an organisation. This means that after becoming accquainted with each other, scientists may collaborate and begin their own research. This in itself is a benefit as it means that research is not neccisserily dictated by profit, and therefore more beneficial to the scientific community.

Problem A also includes the fact that there is little interdiciplinary research within the scientific community. By creating a forum that is open to scientists from all disciplines, they can come into contact with each other. This increases the liklihood of collaboration occuring between scientists of different disciplines and therefore, interdisciplinary research.

Technical Overview

The forum will be made up of two elements:

- A database (the backend) to contain each thread and its associated responses.
- A user interface (the frontend) with which the user can read threads, respond to them and create new ones. All of these actions will submitting data to the database, which means that their must be some interface between the backend the frontend. This user interface is discussed later in the document.

The database will be using MySQL. Therefore, it will use SQL syntax to perform CRUD (Create, Retrive, Update, Delete) actions on the database.

The database will be made up different tables, each table containing multiple attributes which describes a particular object. I will describe each table briefly below:

Thread Table

This table is used to store the original post in a thread. It uses the `tid` attribute to link responses to this original post to create an overall thread.

Name	Purpose
tid	%
uid	$
subject	This is a string that contains a summary of what the thread is about.
message	¬
dateTime	This contains the time and date when the thread was posted.

Messages Table

This table is used to store a response to the original post. It links to a particular thread using the `tid` attribute.

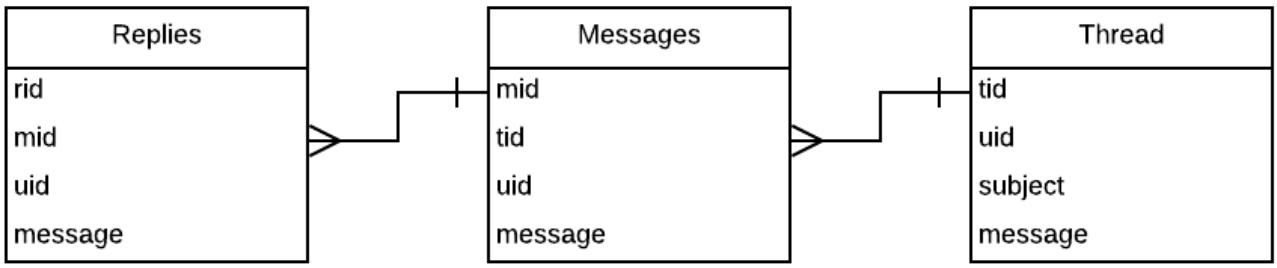
Name	Purpose
mid	%
uid	$
tid	This contains the ID for a particular thread.
message	¬
dateTime	This contains the time and date when the message was posted.

Replies Table

This table holds all of the responses to a particular message within a thread.

Name	Purpose
rid	%
uid	$
mid	This contains the ID value of the message that is being responded to.
message	¬
dateTime	This contains the time and date when the reply was posted.

These tables have relationships between each other to build up an entire thread and may consist of one original post, multiple messages and multiple replies attached to each of those messages.



Note that the `Replies` table does not need to contain a `tid` attribute, as the ID of the thread that the reply belongs to can be inferred via the value of the `tid` attribute in the message that the reply is related to.

There is one more element to this solution that should be added. Henry is a physics student who has an interest in the scienetific forum as a result. He suggested to me (Article 2.1) that the forum should include "formatting techniques".

Formatting techniques are methods are change the presentation of text. This can let the user "display information succinctly" (if the user creates a list of say, resources neccessary for an experiement) and "emphasize" certain key points in a short essay about say, methane. Overall, users can communicate much more effictively using text formatting, as they can convey what they mean more reliably.

They also may make communication more efficient as it reduces confusion between users, so less communication has to take place to convey an idea or point.

As Henry is part of my target audience, he knows what end users will want. So in order to make sure that my solution satisfies the problems that this audience suffers, I should include formatting techniques in my final solution.

How It Solves Problem B

Problem B is based of the fact that the only way to determine the final effect of a known histone modification sequence is through using lab techniques. This is because even a short DNA sequence can contain many histone modifications, making the effect of a sequence difficult to determine. However, the effect of each histone modification on its own is known.

Using these facts, I can determine the overall effect of a sequence by taking into account each histone modification's effect as discrete integer values, and then find the sum of these values. This will therefore involve a simple mathematical formula:

$$\sum_{1}^{n} m * f$$

Where:

- *n* is the number of histone modifications.
- *m* is the magnitude of the effect of the current histone modification in the form of an integer. This represents how influential a histone modification is on the expression of a gene.
- *f* is the type of effect of the current histone modification, and can either have a value of 1 or -1, for histone modifications that cause activation and repression ones respectively.

This formula finds the sum of the all histone modifications' magnitude multiplied by their effect. This therefore finds a discrete value for the change in expression of a gene using discrete values for effect and magnitude. Because of the values of f, an overall repressive effect will have a negative value and an activational one will be positive. The magnitude of the sum will indicate how large a the change in effect is.

From this formula, we can predict what the effect of a given histone modification sequence will be. This solves problem B.

However, in order to find out what the effect and magnitude values of each histone modification are, we will have to consult a table that contains these. This is partly the reason why we will have a database for this histone modification interpreter.

The other reaons for having a database include:

- To store the histone modification sequences that are created by users.
- To store the DNA sequences that are being affected by these histone modification sequences, so that the gene/s or DNA sequences that are being affected can be identified.
- To store important information in the form with each histone modification sequence that may be useful to readers.
- To attribute specific histone modification sequences to particular diseases, such as increases in the rate progression of Alzeimer's Disease due to activational histone modifications on the *APP* gene (which is a protein that is linked to the cause of Alzeimer's Disease).

The tables include within the database are as follows:

NucleosomeDNASequences Table

This table holds the DNA that may surround a single nucleosome. It should be stored as there may be multiple different histone modification sequences for one DNA sequence, such as a gene. Therefore, the same nucleosome DNA sequences will be used. This will save storage in the database.

Name	Purpose
ndsid	%
DNASequence	This is a string that contains the 127 bases of DNA that surrounds a nucleosome.

HistoneMods Table

This table will contain all of the known histone modifications.

Name	Purpose
hmid	%
name	This is a unique string that identifies the histone modification, so that the user can differentiate one histone modification from another.
effect	This represents the type of effect the histone modification has.
magnitude	This represents the amount of influence the histone modification has on the expression of a gene.

Nucleosomes Table

This table will contain all of the individual nucleosomes produced by the users.

Name	Purpose
nid	%
ndsid	This contains the value of the DNA sequence that the nucleosome contains.
histoneMods	This contains a list of IDs of histone modifications that this nucleosome contains. Each mod is differentiated from another by a comma.
nsid	This contains an ID value that represents the histone modification sequence that this nucleosome is part of.

NucleosomeSequencesTable

This table contains the overall information for a particular sequence, such as its name and important information concerning the sequence.

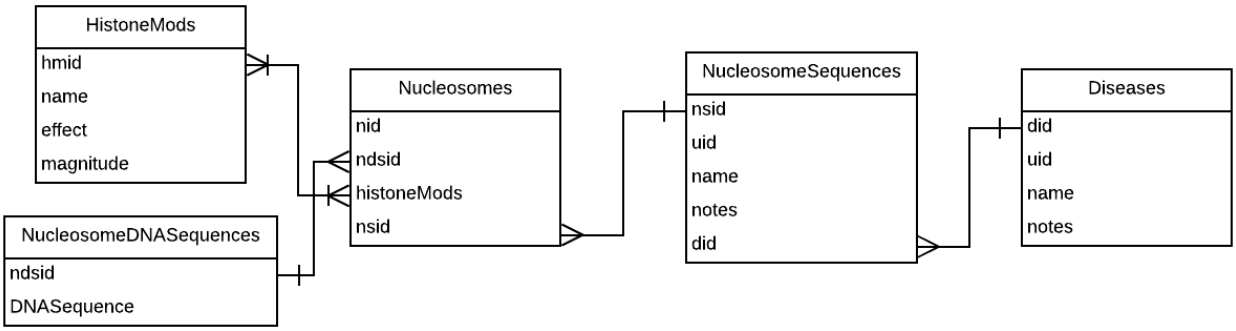
Name	Purpose
nsid	%
uid	This contains the ID value of the user who created the sequence. This will be used to make sure that only this user can edit it and be credited for it.
name	This contains a unique name relative to the user, as different scientists may submit discoveries about the same DNA sequence and histone modification sequence simultaneously.
notes	This contains important information about the sequence.
did	This may contain the ID value of the disease that the sequence that it is associated with.

Diseases Table

This table will contain overall information about a particular disease that different histone modification sequences may be attributed to. This could be because different histone modifications sequences alter the same gene, or that different histone modification sequences alter different genes which contributes to the overall disease. Note that "Diseases" is only an arbitrary name as the records of this table could be used to group nucleosome sequences based on other pieces of information, such as whether or not the sequence increases cancer risks.

Name	Purpose
did	%
uid	This contains the ID value of the user who created the disease. This will be used to make sure that only this user can edit it and be credited for it.
name	This contains a unique name, so that the disease can be differentiated from others.
notes	This contains important information about the disease, such as its symptoms and other non-epigenetic causes.

These tables have a relationship. This allows for multiple nucleosomes to be strung together to form a nucleosome sequence, multiple nucleosome sequences to be associated with a single disease and multiple histone modifications to be a part of a single nucleosome.



It is important to notes that the `Nucleosomes` table has a many-to-many relationship with the `HistoneMods` table. This is because many different nucleosomes may contain the same histone modification and one nucleosome may contain many different histone modifications.

It is also important to note that their is a structural hierarchy here, with the Diseases table at the top, containing all nucleosome sequences, which contains all nucleosomes, which contains all nucleosome DNA sequences an histone modifications. This lets us build up a large overall picture of say a disease, and break it down into further and further chunks, first by looking at the combinative effects of multiple nucleosome sequences, then the effect of each nucleosome sequence by itself, then the modifications on each nucleosome and the DNA sequence that they effect. Although note that nucleosome sequences do not have to be associated to a disease and therefore, may be at the top of their own hierarchy.

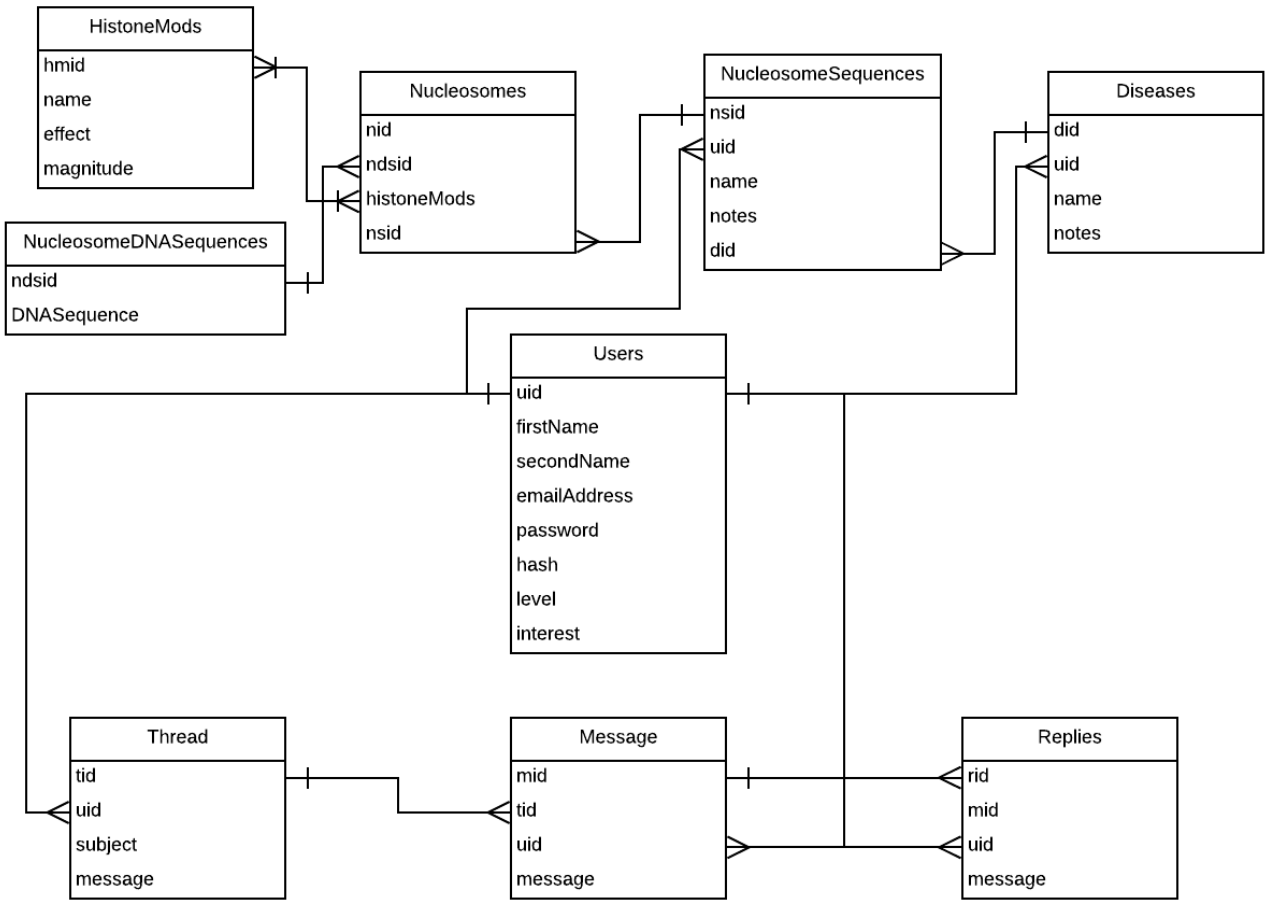
This allows doctors to read detailed information on specific diseases, the histone modifications that help cause them and the DNA that these modifications influence. This therefore helps solve Problem B as doctors can read into the symptoms of a disease and the specific causes,w which helps them determine whether or not a patient has a particular disease. This helps to solve problem B.

Elements That Affect Both Solutions

The **users** table holds a unique record (called an **account**) for each individual who uses *Nomios* and is what allows for specific nucleosome sequences, diseases, threads, messages and replies to be attributed to an individual. This allows for SQL queries which can check if a user owns say, a particular nucleosome sequence, so that we can let only the user who owns the nucleosome sequence to edit it.

Name	Purpose
uid	%
firstName	This contains the first name of the individual who owns this account.
secondName	This contains the second name of the individual who owns this account.
emailAddress	This contains a unique email address that the individual owns. This will be used to confirm the account's validity via email to prevent bots from spamming the database.
password	This contains a password that has been encrypted (cannot be read by humans) using a hashing algorithm. This prevents malicious users who can see the database from extracting passwords. Once encrypted the password cannot be decrypted, as hashing works only in one direction, making the password more secure. In order to check if an inputted password/emailAddress combination is correct, the inputted password will be hashed using the same algorithm and compared with he stored one.
hash	Contains a hashed password value using a different hashing algorithm to enhance security. This is set to NULL once the account has been activated.
level	A value of 0 represents a standard account, 1 represents an admin account.
interest	This contains a string value that holds basic information as to why the user registered an account. this could be sold to large scientific websites.

The *users* table also connects both solutions' entity relationship diagrams. This is because the *users* table has relationships with both diagrams. this is powerful because it lets a single user use one account for both solutions. This is will lower the amount of storage required for a database and creates a smoother user experience. This therefore solves both problems for a user, such as an epigeneticist.



User Interface

The user interface (UI) is important as it is through this that the user uses my solutions. Therefore, in order to cater the needs of my target audience I decided to discuss what the user interface should be, and how it should look directly with them. This would ensure that my solution would be accessible to my target audience.

Note that the aim of a good UI is a good user experience. This means that the solution is not unpleasant to look at, all actions perform as expected, easy to navigate and they can find the information they need quickly and effortlessly. It must also be responsive so that it can be used on multiple devices.

Henry is a physics student, and therefore intrested in a scientific forum. He discusses in writing what he thinks the UI should be and look like (Article 2.1). These design cues should also apply to the HMI as both solutions are inside the same web application. This means users only have to become familar with one UI instead of two.

He first writes that from a design perspective the UI should be "clean and minimalist". "Clean" means that the user interface should look bright, vibrant and not too crowded with multiple design elements. This is beneficial as it makes sure that the website is a pleasure to look at and therefore creates a better user experience. Therefore I will need to design a clean UI.

The word "minimalist" means that no unncessary information should be displayed. This makes sure that the user is not overwhelmed with information and can find the information that they are looking for quickly without having to use a search tool within the browers (such as Google Crome's Find tool). This increases the usability of the forum, as the user is not prevented from finding the information they are looking for.

This creates a better user experience. Therefore I will design a minimalist UI.

Henry also states that whilst mataining these qualities the UI should also be "very functional when required". This means that the UI should let an action be possible when the user is going to use it. For example, the user should be only able to post threads when they are in the looking at a page that lists all threads, and not when they are reading a specific thread. This also helps the user expect different actions when in different sections of the UI, so they know where to go in the UI to perform a particular action. This therefore means that the UI is easy to navigate. This creates a better user experience. When designing the UI I will ensure that each action only appears in one section (to make the UI easy to navigate).

To make sure that actions perform as expected the UI will have be designed so that either a successful result is returned after an action or an error that the **user** can fix is returned. Therefore it is important to catch any logical errors before they occur so that logical errors at runtime are not displayed, such as a lack of an input value. These errors can then be displayed to the user in the form of a user-friendly message. I will implement this in my UI.

Finally, the UI must be responsive to different devices. This is to give as many users as possible access to *Nomios*. To do this I will use a specific feature of a technology called **CSS** called **media queries**. These can change the style of a UI depending on the size of the screen of the device used.

Use Cases

In order to explain how my solutions work, I thought I would explain use cases for my solution. The forum is self-explanatory, so these use cases only apply to the HMI.

1. A doctor wishes to diagnose a patient with a highly aggressive form of Alzheimer's Disease. In order to perform this diagnosis there must be evidience of this disease. This form of Alzheimer's Disease has activational histone modifications at the gene that codes for the APP gene. The doctor will first find out what the sequence of histone modifications are in the epigenome of the patient. They then sign into *Nomios* and search the database for any sequences attributed to Alzheimer's Disease. They discover that a user has already submitted a histone modification sequence at the APP gene that creates the symptom of highly aggressive Alzheimer's Disease. They compare the patient's histone modification sequence with that on *Nomios* and discover that they match. The doctor can now diagnose the patient with this particular highly aggressive form of Alzhiemer's Disease.
2. A researcher in epigenetics wishes to discover what the overall change in expression is for a particular gene. that has a lot of repressive histone modifications and a few activational histone modifications for its histone modification sequence. They sign in to *Nomios* search the database to see if any one has already submitted this histone modification sequence. Unfortunately nobody has. They then submit their histone modification sequence with the DNA of the gene to their database. After navigating to the sequence they have created, they discover that the overall change in expression for this gene is -26. This indicates a large decrease in the expression fo the gene. This took 5 minutes to do wheras in a lab it would take weeks if not months. This sequence can then be seen by other users, including other researchers or doctors.

Technologies

There are some key technologies neccessary for this solution to work:

- **PHP** is a programming language that can handle database requests to the server and handles the responses by that server and processes those requests. It is responsible for inputting and outputting data from the database. It interfaces well with MySQL which is one reason why it was chosen. It also has a wide community in case I have any program with the language.
- **HTML** is a way of formatting text so that it can be interpreted by browesers to display a web page. The formatted text can split into parts called elements.
- **CSS** is a way of stylising the elements of HTML. It will be used to partly create the UI.
- **JavaScript** is a programming language that can create UI mechnisms (such as eroor prevention with inputs by deleting invalid characters) that CSS cannot. It can also be used to change the styles of specific elements and add content to those elements, after the page has loaded. It lets us manipulate **cookies** whihc are small text files associated with a aparticualr website stored locally on a computer. PHP lets us do this too, but JavaScript's methods are more reliable.
- **MySQL** is a database design language (DDL) which is used to both create databases and perform CRUD functions on those databases. It interfaces well with PHP which was one reason as to why it was chosen over other DDLs. CRUD is a set of basic functions that are used on a database, known:
    - Create - creates a new record in a table in the database.
    - Retrive - extracts a record froma table in the database.
    - Update - changes the data in a row in a table in a database.
    - Delete - Removes a row from a table in a database.

Research

I did carry out research into problem A and B to make sure that they were genuine problems and therefore, that my solutions are valid:

- Article 1.1 suggests that it is difficult to understand a sequence of histone modifications. "this is a code that is extraordinarily difficult to read". Here "code" refers to a sequence of histone modifications. By "difficult to read" the text is implying that a histone modification sequence is difficult to understand. Therefore, its overall effect on the expression of a gene is difficult to find out, as it is difficult to take into account each histone modification. Therefore, a solution is needed to let epigeneticists read a histone modification sequence.
- Article 1.2 suggests that a method for reading a histone modification sequence successfully has not yet been found ("not yet possble"). Here a histone modification sequence is called a "complex combination [of histone modifications]".

When it came to deciding what elements my HMI solution should contain, I spoke to a potential user: Luke. He is a Biology student at my school and may use the HMI to learn more about Epigenetics. He wrote a peice of writing to indicate what he thinks would be useful in the HMI, as outlined in Article 3.1. Here I shall discuss what he wrote and how I can incorperate this into my solution to meet my users' needs:

- He first states that because histone modficiation sequences are very "complciated" they should be simplifed by being "broken down". This he says is very key to letter users understand a sequence more easily ("makes a sequence easier to understand"). Not only this but "edits" will be easier, because specific nucleosomes can be displayed so a user can specify particular bases or histone modficiations to change, add or remove. I can incorperate this into my solution displaying a histone modification sequence one-by-one and letting the user edit a particular nucleosome when they are editing a hisotne modification sequence.

- He also writes that the entire DNA sequence should be displayed to the user. This is because DNA sequences are used in "other applications". By "applications" he includes creating "artificial DNA" which is useful in genetic engineering. Scientists might want to do this using HMI over another form of base sequence storage because it has histone modification sequences attatched to it, which they might also choose to use in other applications. I will implementn this by displaying the entire DNA sequence when a user is viewing a particular sequence.
- He suggests that HMI will be "difficult" to use. This is due to the software being "unqiue". He is suggesting that the application software is specialist and therefore, not natural to use by the "[UI is] very different" to other peices of software, such as Facebook. Therefore it should contain a "guide". I took this to mean a help menu. I will implement this by creating an entire webpage that the user can read that will contain a complete guide as to how to use the software.
- Finally he discusses the idea of "searching" for a particular histone modification sequence. This is because the database will be "very large" and therefore difficult to locate a specific histone modification sequence, which is what scientists need to do when researching a particular gene or effect on that particular gene's expression, such as the *APP* gene's increased expression in Alzheimer's Disease. I will therefore implement a search tool in my solution so that users can find a histone modification sequence by name. Taking this a step further, I can apply this to my forum and to diseases so that users can find a thread that they are contributing to and so that scientsist scan quickly locate all histone modficiations associated with the disease they are researching into.

Objectives

In order for my solutions to have solved problem A and B , each of these objectives should be met:

1. The user should be able to start threads in the forum so that they can ask questions, further their knowledge and collaborate with other scientists.
2. The user should be able to respond to threads and messages within threads so that communication can take place between users.
3. The user should not be able to remove any messages that they have posted or threads that they have started. This is to make sure that other users can still access old questions so that they can learn from them too.
4. The user should be able to process a sequence of histone modifications on a particular gene **and** recieve a result that lets them know what the change in expression will be. This is fundemental to the solution as it lets users predict how mucha gene is expressed without conducting expensive lab work.
5. The user should be able to edit a DNA sequence so that they can (for example) amend any errors produced and change the current allele to another allele. They should be able to change the sequence of histones to change the resultant expression of the gene.
6. The user should be able to describe what a particular sequence does and be able to name a sequence so that they can differentiate sequences from each other.
7. The user should be able to view groups of sequences that attribute to a particular disease so that they can collaborate with other scientists on research for a particular disease.
8. The user should be able to search for other sequences, sequences within diseases and threads including their own so that they can benefit from other users' data and navigate to their own for their own research.
9. The user should have their own account so that the data that they have submitted can be attributed to them. Similarly, they should not be able to submit new data without being signed into an account to ensure that new data is attributed to their account.

10. There should be a "help" menu that the user can navigate to so that they can learn to use the website quickly and easily.
11. Users should be able to style their forum posts using some form of mark up. This will enhance communication between users as it structures texts so that communication can be more effective and efficient.
12. HTML should not be a valid form of text when submitting posts to forums or creating notes and names for particular sequences and diseases. This is to prevent malicious attacks on the website that may distort its intended display.
13. An admin should be available in the forum that can delete inappropriate threads, messages or replies. This is to ensure that the forum remains a friendly place to communicate so that users maintain communication and collboration.
14. The result displayed once a histone sequence is interpreted (i.e. the change in expression of a gene) should be accurate to what is observed in real life.
15. The website must be viewable on a mobile and desktop device so that it can be accessed by a user no matter their device preference.

There are 15 objectivies, which makes my project fairly large in scope, particularly as the overall project is made up of two distinct solutions.

Critical Path

The critical path outlines the steps I need to take in order to complete my solutions. Each step fufills a specific objective until the project has been completed. Note that at each "test" point within a step constists of testing what was created in the current step and fixing any errors that are found.

1. The entire database should be created, containing both the tables for the HMI and the forum, as well as the *users* table. Some test records are entered. This is done so that code written after can be immediately tested.
2. Create the UI elements in CSS and HTML. This is so that we can quickly use elements we defined earlier when we are actually designing the UI.
3. Create the sign up, sign in and sign out user interfaces. This is so that we can write the scripts without having to imagine how the outputs and inputs will look like in the final product - we can just test it right away.
4. Create the sign up, sign in and sign out PHP and JavaScript scripts and test.
5. Create the user interface for the main page of the forum (where all threads will be displayed) and the UI for posting a new thread.
6. Create the PHP scripts that retrive and display all of the threads in the forum from the database and for submitting a new thread and the JavaScript scripts for error prevention and test.
7. Create the user interface for displaying a single thread's original post and all of its associated responses. Including the UI for posting a new message or reply.
8. Create the scripts neccessary to retrive and display the thread's original post and its responses from the database and test.
9. Create formatting techniques for the forum. This will be done in PHP. When PHP retrives text that includes formatting, this text will be converted to HTML amarkup so that is can be viewed by the browser. Test.
10. Implement admin privilages to the forum, such as deleting specific threads, messages and replies. Test
11. Create the UI for creating a new histone modification sequence.
12. Create the PHP and JavaScript scripts for submitting a new histone modification sequence to the database and for error prevention and test.

13. Create the UI for the search page that allow for searching through the forum and the HMI database.
14. Create the PHP script that lets the user search through the forum and HMI database to find specific threads and sequences respectively, then test.
15. Create the UI for displaying the results returned by the search. Each result should redirect the user to a page displaying only that result in detail (such as a thread page, disease page or histone modification sequence page). This last part will use JavaScript.
16. Create the PHP script that retrives results from the database and displays them in the search result UI.
17. Create the UI for displaying one histone modification sequence sequence in detail (nucleosome by nucleosome). This will also contain the UI for editing this hisitone modification sequence, but this will only display when the current user is the one who owns the histone modficiation sequence. This will be made using CSS and PHP.
18. Create the PHP script for retriving from the database displaying the histone modfication sequence. Test.
19. Create the UI for displaying a single disease and all associated histone mdofication sequences. This will also display the UI for editing the details of the disease, such as the name and important notes. This UI will only be displayed if the current user is the one who owns the histone modficiation sequence. This will be made using CSS and PHP.
20. Create the PHP script for retriving from the database displaying the disease and associated histone mdofication sequences. Test.
21. Test that the UI is responsive to different screen dimensions for different devices. Make any fixes that are neccessary.
22. Perform a final test on all elements on the *Nomios* and perform any fixes neccessary so that there are no issues that have arisen from the code being altered as elements of the solution are added to this code.

Appendicies

Here is my list of texts that I have refered to throughout this document, and their sources.

Article 1.1

Source: *The Epigenetic Revolution* by Nessa Carey.

> The pattern of modifications is referred to as a histone code. This problem that epigeneticists face is that this is a code that extraordinarily difficult to read.

Article 1.2

Source: *The Epigenetic Revolution* by Nessa Carey.

> ... it's not yet possible to make accurate predictions from complex combinations.

Article 2.1

Source: Henry - a Physics student.

> What I'd expect from a scientific forum purely in terms of design would be for the interface to be clean and minimalist but also very functional when required. Often when discussing complex topics the need for useful formatting techniques such as emboldening specific text in order to emphasize points and that I can use simple lists in order to display information succinctly.

Article 3.1

Source: Luke - a Biology student.

> A histone modification sequence can be very complicated. To simplify it, I would like it to be broken down into nucleosomes before it is displayed. This makes a sequence easier to understand and lets edits be more specific to particular nucleosomes.
>
> Sometimes researchers needs to access an entire DNA sequence for other applications, such as producing artificial DNA. The DNA sequence that is used by a particular histone modification sequence should be displayed in full for a particular histone modification sequence.
>
> This tool will probably be difficult to use. This is because it is very unqiue and therefore it will be confusing to first use as it will look very different to anything else. THe user shoudl have some form of a guide.
>
> The database will be very large and I am often looking for a specific histone modification sequences. I think that I would want to be able to search for a sequence by name so that I can retrive data from it quickly.

Nomios - Documented Design

Introduction

In this document I shall outline key components of my solution, and explain how they work. I will also outline how individual components of my solution work to form a whole in my *High Level Overview*.

Definitions

Below are some of the necessary terminology and their definitions for this document. Any other bits of terminology used are defined in the *Analysis* document.

> **Data Flow Diagrams** (**DFD**) shows how data is processed by a program, where it is stored as well of what is outputted and what data is inputted into the program.
>
> A **class** is a template for the **properties** (attributes) and **methods** (functions) of a particular object.
>
> A **Class Diagram** shows the structure and function of a class. It is split up into three sections. The top section displays the name of the class. The middle displays the properties of the class. The bottom section displays the methods of the class. The follow table is a key for the symbols used in a class diagram:
>
> | Key | Definition |
> | --- | --- |
> | + | A public property/method. |
> | - | A private property/method. |
> | : | The string before this symbol is the name of the property/value. The string after is data type of the property or the data type of the data returned of the method. If there is no second type than the method is **void**. This means that is returns no data. |
> | = | The string after this symbol is the default value of the property. |
>
> A **cookie** is a small text file that is stored locally on a computer that is associated with a particular website.
>
> **XSS** attacks are a type of malicious attack that allows malicious users to inject client-side scripts into a web page. For example, by inserting HTML code into a database table that is later read.
>
> A **thread** is a sequence of posts on a given topic, chosen by the first post in a thread. This first psot in known as the **original post**.
>
> A **message** is a response to a thread's original post.
>
> A **reply** is a response to a message within a thread.
>
> The **uid** is the unique ID of an account and will be stored in a cookie.
>
> A **static query** is a SQL query that does not use any variables and therefore does not change.

The **browser viewport** is where a webpage is displayed within the browser.

The **level** of a user indicates the type of account that user has. It should be stored in a cookie.

A **sequence** is a nucleosome sequence that contains a DNA sequence and histone modification sequence for each histone.

A **table** is a data structure within a database that contains data about an object.

An **attribute** is a data type for a particular property of the object a table is trying to describe.

A **record** is a single data entry in a table. It contains values for each attribute in the table. This is also known as a **row**.

A **column** is a set of values for one attribute - one value for each record. These values are of the same type as they are from the same attribute.

Overall Data describes information that names or gives a description for a particular sequence or disease. This description should aim to give an overview of the information concerning this sequence or disease.

A **HTML form** is a HTML element that contains inputs that submit data via either the POST or GET protocols. Inputs within the same form cannot have different protocols. The situations in which the POST and GET protocols should be used are outlined below:

Protocol Name	When It Should Be Used
POST	This is used for data that is very large (such as a long string) or sensitive data (such as a password). This is because POST can submit data that is much larger and is not stored in the URL so it cannot be viewed by the user.
GET	This is used for data that is short (such as an ID for a forum post) or insensitive data (such as a username). This is because GET submits data via the URL which means that data should be shorter so that the URL is still fully visible and because the URL is visible it lets malicious users look at data directly through the URL, so sensitive data should not be submitted through GET.

This is a key for purposes of attributes in database tables:

Key Symbol	Definition
%	This will contain a unique ID for each record in the table. When a new record is added, the integer value will increment based off the value of the previous record. This is what makes each record's tid value unique.
$	This contains the unique ID of the user who created this post/started the thread. This is useful for retrieving information from the user, such as their first name that will be displayed to track messages to users in a thread.
¬	is is a string that contains the actual body of the original post in the thread.

> This is a table of definitions for the types of attributes in database tables:
>
> | Type | Definition |
> | --- | --- |
> | **VARCHAR *x*** | A string with a variable length in characters up to a maximum of *x*. This means that the length of the string is always equal the length of the text in characters up to a maximum. This saves storage. |
> | **CHAR *x*** | A string with a fixed length of *x* characters. |
> | **UNSIGNED INT** | An unsigned integer value. This means that all values are positive, increasing the range of positive values that can be represented. |
> | **BIT** | A single bit. Can have a value of either 1 or 0. |
> | **AUTO_INCREMENT** | Precedes an INT type. This causes the value of the attribute for a record to be one greater than the previous record's. |
> | **PRIMARY_KEY** | This defines an attribute to be the primary key of a table. The value of this attribute for this record is unique so that a specific record may be retrieved quickly and simply. |
> | **FLOAT** | A decimal value using a floating decimal point. This increases the range of values that can be contained. |
> | **NULL** | Specifies that the value of this attribute for a record may be null (meaning that it contains no value). If not specified, assume the attribute is has the NOT NULL type (this means that the attribute for a row must have a value). |

High Level Overview

I have decided to model both of my solutions using a data flow diagram. This is because my solutions use a lot of data (for example, long DNA sequences) and the way this data is stored and retrieved and the processes for both of those is very complicated. A DFD can show these processes more simply and therefore make it easier to implement.

These DFDs use the **Yourdon and Coad** notation. This is because I find this notation to be the most clear and therefore, useful to me when implementing my solutions. In this notation, **processes** (such as algorithms) are represented by circles, external entities (input and output mechanisms) are represented with **shorter rectangles** and data stores (such as a table in database) are represented with **longer rectangles with the left edge missing**.

The complexity of a DFD can be represented in levels, with level 0 being the simplest, single process DFD and level 3 and beyond being incredibly complicated that only programmers can understand it. To make sure this document is readable by a large audience whilst still being useful during my solutions' development the complexity of my DFD diagrams will be level 2.

A **physical DFD** shows how a program should be actually implemented. A **logical DFD** is used in a business context and is therefore not relevant to this project. Because of this and so that the DFDs are relevant for later use my DFDs will be physical.

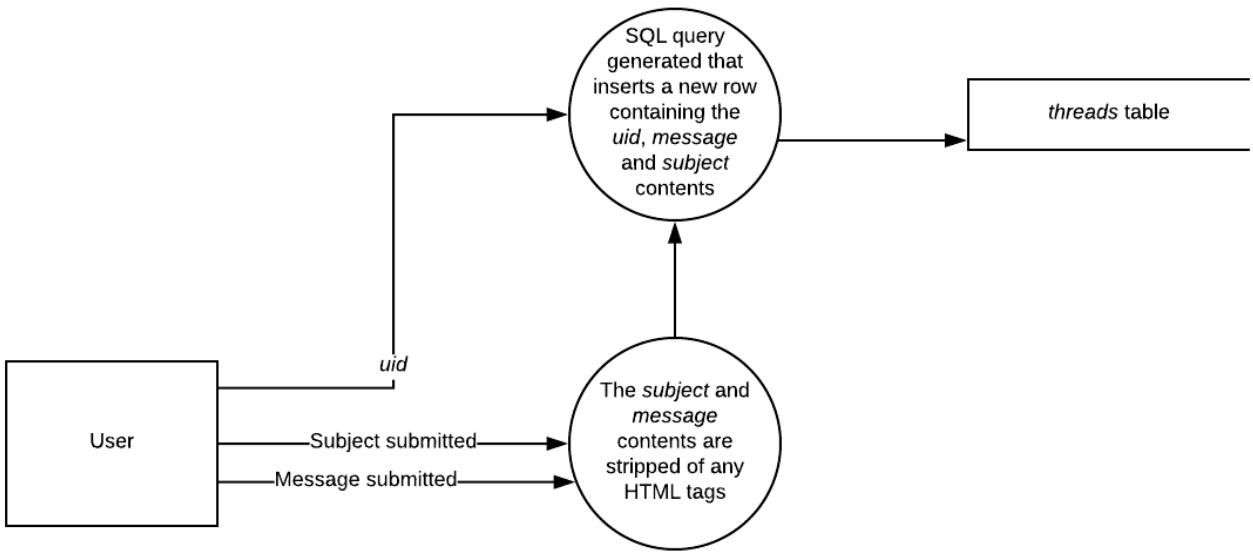
Each DFD makes references to the database. This was outlined in my *Analysis*, but will be explained in further detail in this document. Either can be consulted when reading these DFDs.

this high level overview is meant to give an abstracted idea of how the solutions work. For a more in detailed explanation of each component of the solution please consult *Technical Components* section of this document.

Forum

As an overall forum DFD would be very complicated and difficult to understand I have broken up the DFD into five different diagrams. Note that external entities should have an output and input but my diagrams do not show this. This is because when linked together all external entities show this characteristic but when broken down this is not the case.

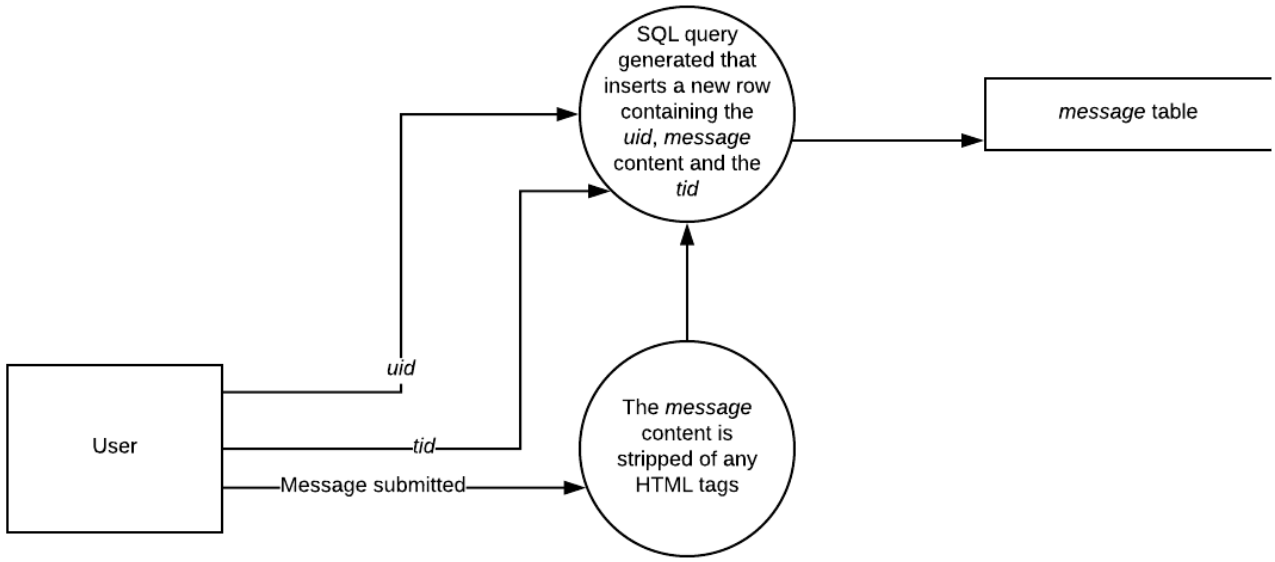
Starting A Thread



When a user wishes to start a new thread they submit a subject and body for their post (called the *subject* and *message* respectively), the contents of the *subject* and *message* are run through an algorithm that removes any HTML tags. This is to prevent XSS by a malicious user, as the client-side code (such as JavaScript which is inside HTML tags) is removed before being inserted to the database.

The stripped contents of the *subject* and *message* is then passed to another algorithm along with the *uid* which produces a SQL query that inserts a new row into the *threads* table. This is query therefore submits the *subject* and *message* contents to the table so that it can be stored when executed. The uid is stored so that the thread can be related to the user so that information such as the user's first name can be displayed with the post.

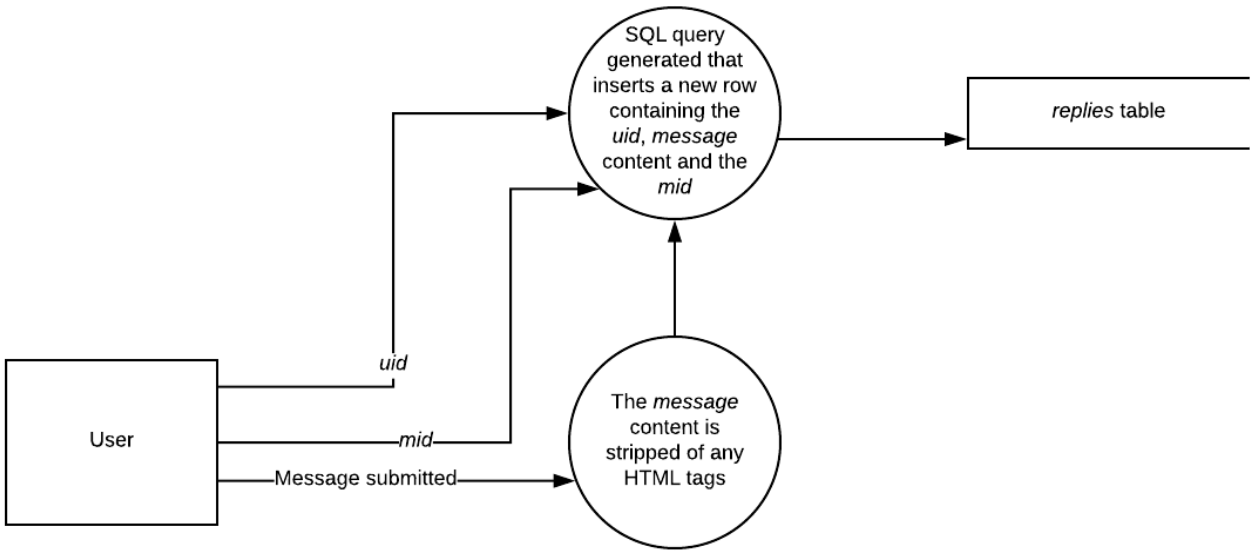
Posting A Message



When a user wishes to post a message, they input the body of the response (also called the *message*). The contents of this is the run through an algorithm that removes any HTML tags to prevent XSS attacks just like when posting a thread.

The ID of the thread (called the *tid*) is fetched from the URL of the user's current page. The *uid* is fetched too. These are used with the stripped content of the *message* to generate a SQL query which inserts a new row into the *message* table when executed. The uid is stored so that the message can be related to the user so that information such as the user's first name can be displayed with the post.

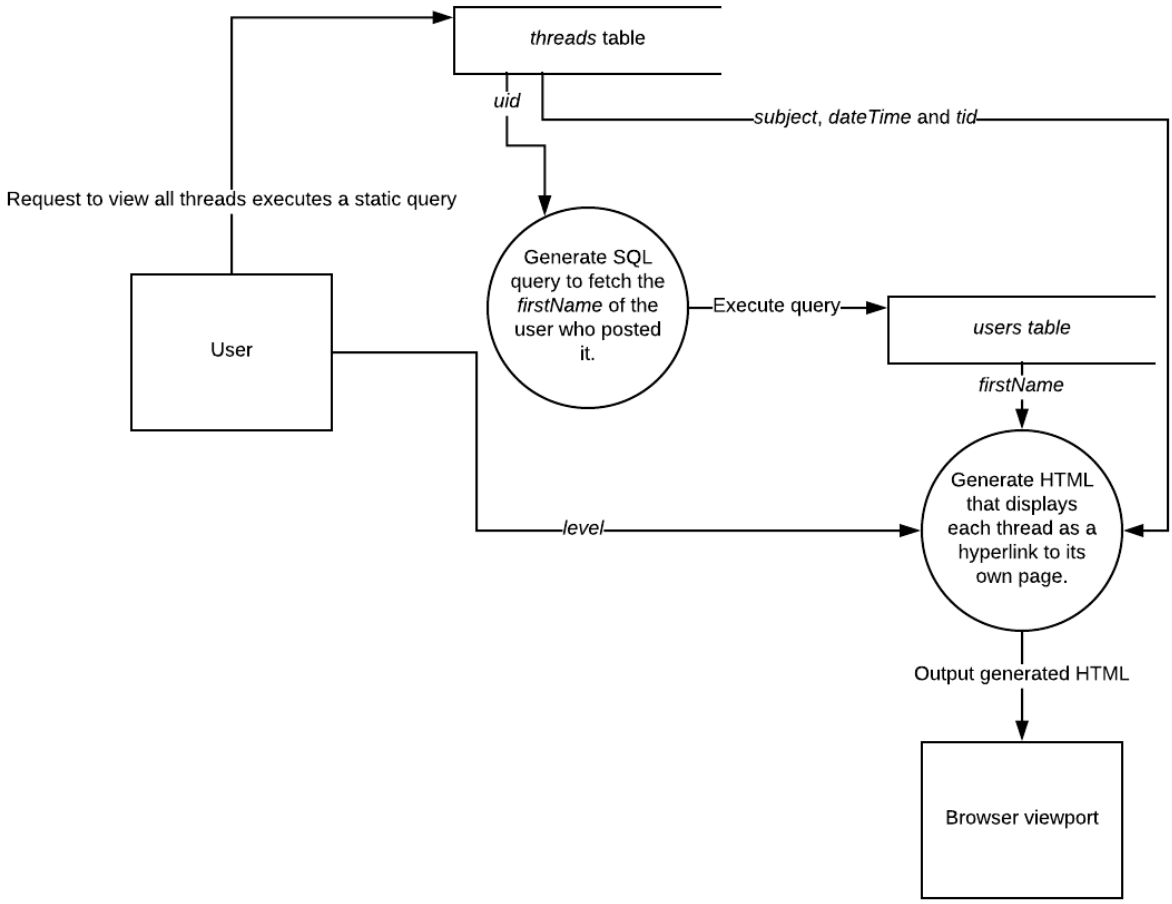
Posting A Reply



When a user wants to post a *reply* they must first input the body of text that they wish to respond with. This is called the *message*. The contents of this is the run through an algorithm that removes any HTML tags to prevent XSS attacks just like when posting a thread.

The unique ID of the message (called the *mid*) that is being responded to is fetched. This along with the *uid* and stripped contents of the *message* is processed by an algorithm to produce a new SQL query that will insert a new row into the *replies* table. This lets the data be stored so that it can be retrieved later.

Viewing All Threads

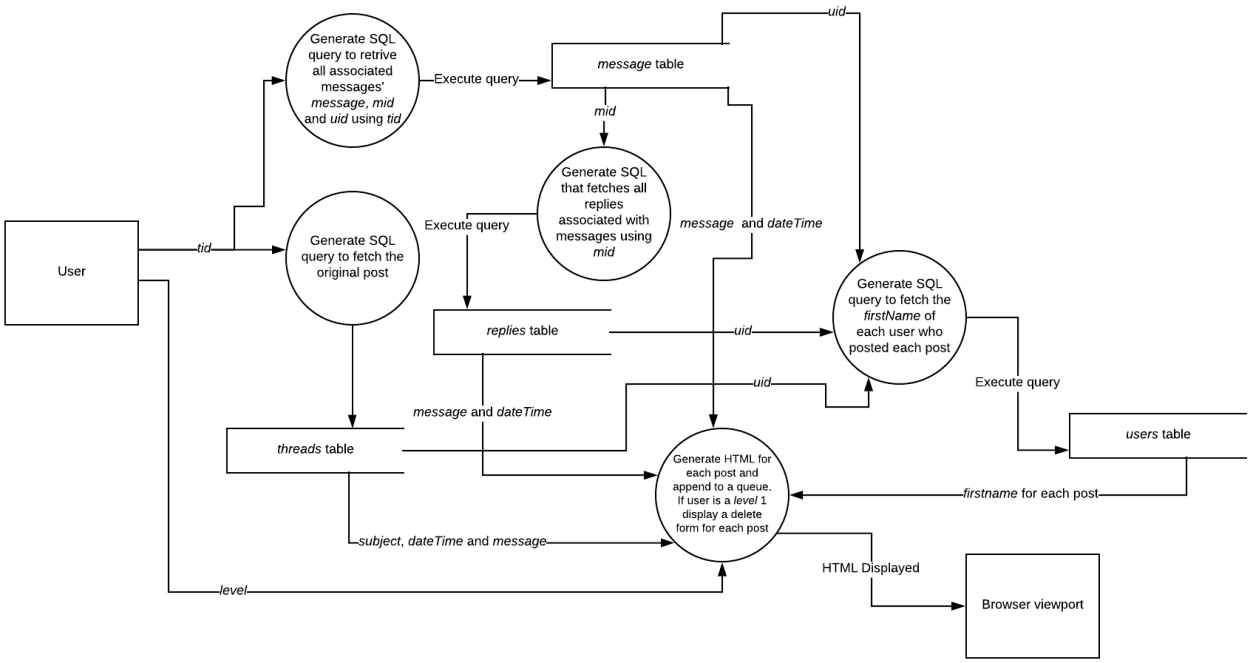


When a user request to view all threads a links to their own pages, a static query is executed on the *threads* table. This fetches the *subject*, the *dateTime* (the date an time at which the thread was started) the *uid* of the user who started the thread and the unique ID of the thread itself (the *tid*). It is ordered in chronological order using the *dateTime* attribute.

The *uid* of the user who started the thread in inputted into an algorithm which generates a SQL query. This query will retrieve the *firstName* attribute of the user who started the thread. This data, along with the *subject*, *dateTime* and *tid* is used to generate HTML that displays each thread as a hyperlink, with the *subject* being displayed to the user so that they know what the topic of the thread is, the *dateTime* so that they know how relevant the thread is to the current date and time, and the *firstName* of the user who posted it so that they relate two threads to the same user (this is useful for understanding the scientific expertise of the user for recruiting reasons).

Note that the URL for this hyperlink will contain the *tid* so that the page redirected to can use it to display that thread. This use of the *tid* is shown in the DFD below.

Viewing A Single Thread



When the user wishes to view a thread on its own page (so that they can read the entire thread), they will have clicked on a hyperlink produced by the *Viewing All Threads* DFD. This hyperlink will contain the *tid* of the thread that the user wishes to view. This *tid* is used in two algorithms:

- To generate an SQL query that retrieves the original post of the thread. When executed, this SQL query fetches the *subject*, *message* and *dateTime* attributes (the subject, body and date & time of posting respectively) of the original post. The *uid* of the user who started this thread is also retrieved.
- To generate a SQL query that retrieves the *message*, *mid* and *dateTime* attributes of each message that is associated with the original post. The *uid* of the user who posted this message is also retrieved. This *mid* is the unique ID of the message.

  After this, each message's *mid* attribute is used to generate a SQL query that retrieves data from all of the *replies* associated with each message. This data includes the *message* attribute, the *dateTime* attribute and the *uid* of the user who posted this reply.

Note that each post's *dateTime* attribute is used to order the posts based on when they were posted. This will be explained further later in the document.

The *uid*s retrieved from these three queries, are sent to an algorithm. This algorithm generates a SQL query that uses this *uid* to fetch the *firstName* of the user who submitted each post. These *firstName*s are then returned to the next algorithm.

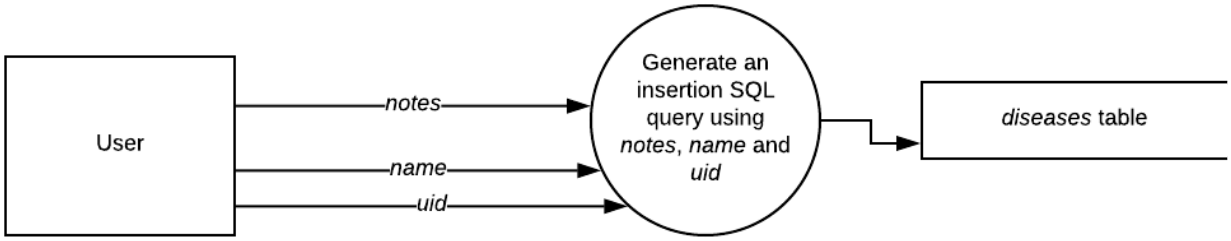
This algorithm takes these *firstName*s and the *message* and *dateTime* attributes of each post (the *subject* is also included in the case of the original post) as inputs. It also retrieves the current user's *level* attribute. Using these data, it generates HTML for each post, starting with the original post and working the *messages* chronologically, generating the replies chronologically for each message before moving onto the next one. This is thanks the the data being retrieved by the queries being returned in a chronological order. This HTML should allow for each post to be displayed as a block once it is outputted so that each can be clearly distinguished from one another in the browser's viewport. If the *level* of the user has a value of 1, then they must be an admin. This means that a delete form should be generated as part of the HTML for each post. This allows the admin to delete inappropriate or malicious posts. After this, each post's HTML is appended to a queue.

The queue is then used to display each post in order, with the original post being displayed first, the associated *messages* in chronological order and their associated *replies* in chronological order too.

Histone Modification Interpreter

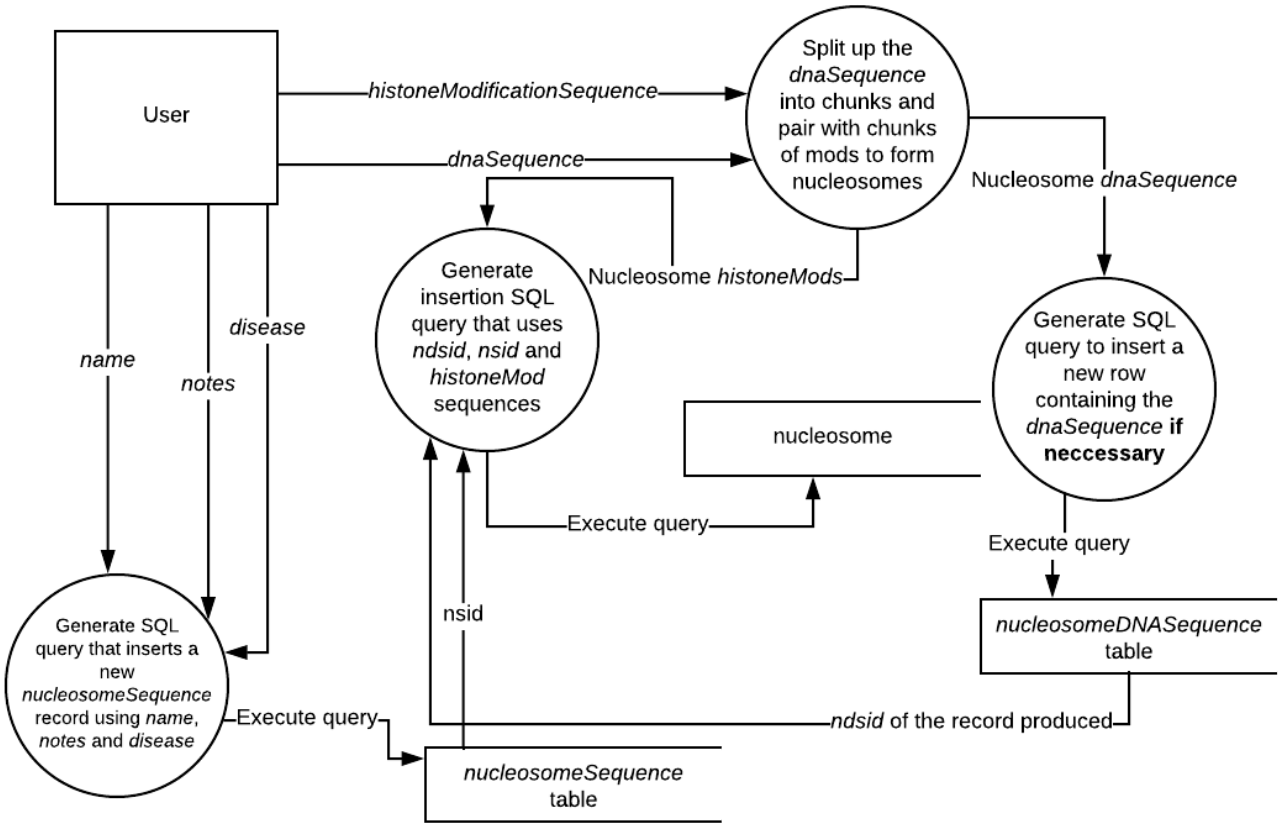
As with the forum DFD, I have broken down the DFD for the HMI into four separate diagrams. This should make the overall DFD easier to understand when each DFD is put together.

Creating A New Disease



When the user wants to create a new disease, they first submit some information via a form. This includes the *notes* and *name* of the disease. The user's *uid* is also retrieved. These data are then used to generate an SQL query which will create a new row in the diseases table when executed. This therefore stores information on the new disease and thus creates a new disease.

Creating A New Sequence



When a user creates a new sequence they submit some information in a form. This includes the *name* of the sequence, important information on the sequence (known as the *notes*), the *disease* (the ID of the disease that the sequence is associated with, the *dnaSequence* and the *histoneModificationSequence*. What happens with these data differs:

- The *name*, *notes* and *disease* data is passed to an algorithm which generates a SQL query which will insert a new record into the *nucleosomeSequence* table when it is executed. This must happen before the next step because the *nsid* (the unique ID of a *nucleosomeSequence* record) of the record inserted is necessary for a later algorithm.
- The *histoneModificationSequence* is split into different chunks as specified by the user. Each chunk represents the histone modification sequence for a particular nucleosome. The *dnaSequence* can be broken down into nucleosome-sized chunks without the user specifying each chunk from the others, as 127bp of DNA **must** be wrapped around each nucleosome. In order to split the *dnaSequence* and *histoneModificationSequence* into nucleosome-sized chunks, and to match each *dnaSequence* chunk with its respective *histoneModificationSequence* chunk, the full *dnaSequence* and *histoneModificationSequence* is passed through an algorithm.

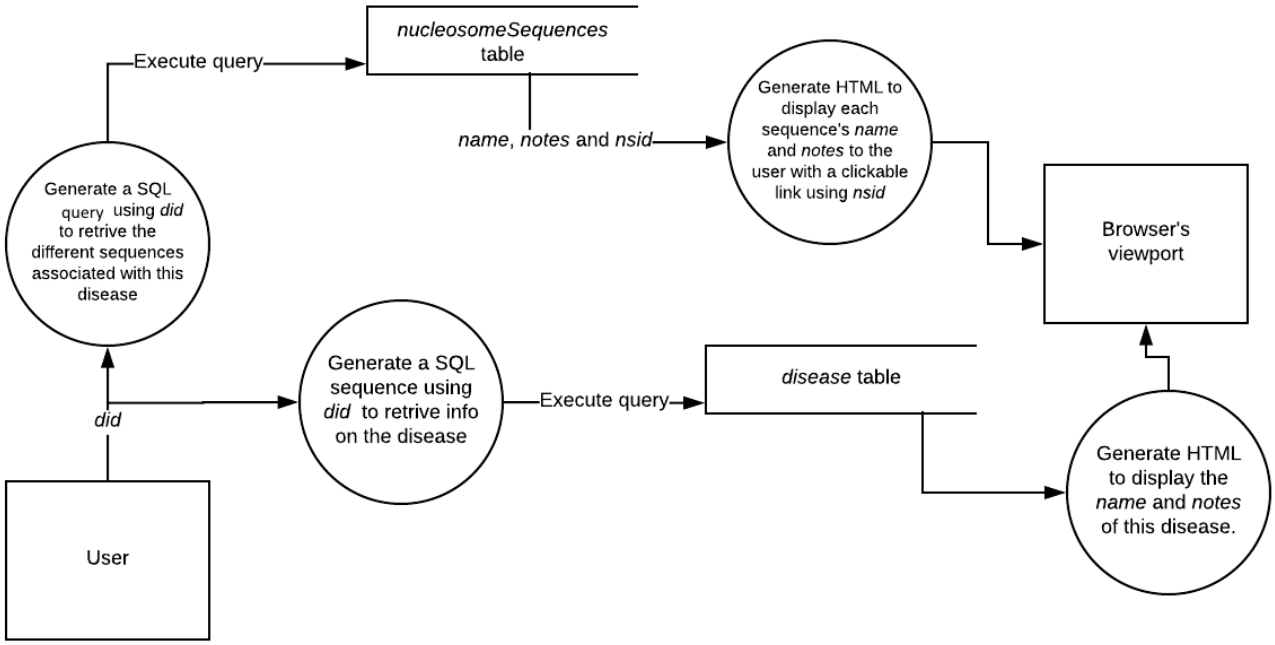
  Then each nucleosome's chunks pass through some processes.

  The *dnaSequence* chunk is passed through an algorithm that checks whether or not a this particular *dnaSequence* chunk has already been stored in the *nucelosomeDNASequence* table. This is done to prevent storage from being used unnecessarily, as different histone modification sequences may act on the same DNA sequence (for example, the *APP* gene for when studying the effects of histone modification sequences in Alzheimer's Disease). If it has not been already stored this same algorithm generates a SQL query that when executed inserts a new record into the *nucleosomeDNASequence* table. The unique ID of the record produced (*ndsid*) is then retrieved once this query is executed. If the DNA sequence has already been stored then the *ndsid* of the record that has that DNA sequenced is retrieved.

  The *histoneModificationSequence* chunk is then passed along with the *ndsid* and *nsid* that were retrieved as inputs into an algorithm. This algorithm generates a query for each nucleosome that inserts a new record into the *nucleosome* table containing these data. The *nsid* was retrieved from the previous bullet point. It stays constant so that all of the nucleosome records produced will be associated with the same *nucleosomeSequence* record and therefore, the be a part of the same nucleosome sequence.

  Note that the *ndsid* associates a DNA sequence chunk with a nucleosome. Therefore there is a many to one relationship between the *nucleosome* table and the *nucleosomeDNASequence* table. This was outlined in the *Analysis* document.

Reading A Disease



When the user wishes to read the contents of a disease they will have the unique ID of the disease (*did*) stored in their URL. This *did* is extracted and then used as inputs in two separate algorithms:

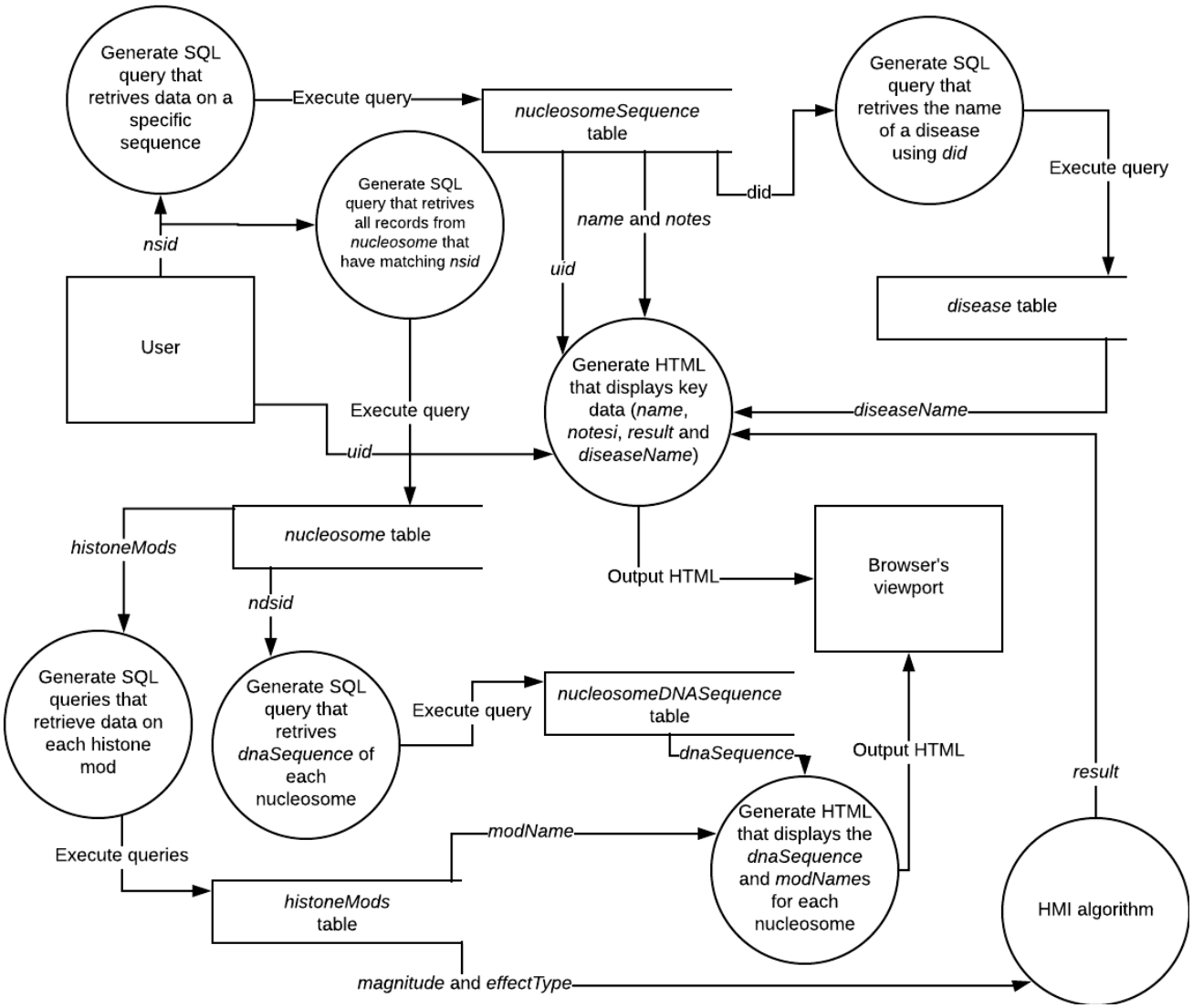
1. The first generates an SQL query that will retrieve data on each sequence that is associated with that disease. Any record that contains the same *did* value as the one inputted will be returned. This SQL query is then executed and the *name*, *notes* and *nsid* (the unique ID of the nucleosome sequence) attributes are returned for each record.

   The data contained within these attributes is then used in another algorithm that generates HTML for each sequence, displaying the *notes* and *name* data and using the *nsid* attribute to create a clickable hyperlink that will redirect the user to a page to read that sequence.

2. The second algorithm generates an SQL sequence using *did* to retrieve data such as the *name* and *notes* on that particular disease from the *disease* table. When executed, the retrieved data is used to generate HTML that displays these two pieces of data to the user.

The second process is used first so that the important information about the disease can be displayed at the top of the web-page, so that the user knows what the sequences all have in common and any important information about the disease that these sequences may contribute to, such as a symptom or likelihood of a disease occurring in an individual.

Reading A Sequence



When a user wishes to read a sequence they will have a unique ID for the nucleosome sequence they are trying to read (known as the *nsid*) in their URL. This will be used in two separate processes:

- The first generates a SQL query that retrieves the *did* (the unique ID of the disease that the sequence of is associated with), the *uid* of the user who created this sequence, the *name* of the sequence and the *notes* of the sequence. The last three pieces of data are sent to the **Display Algorithm 1**. However, in order to retrieve the name of the disease (*diseaseName*) that this sequence is associated with, a SQL query is generated using *did* that does this. Once it is executed, if a record contains the same *did* as the one specified, the diseaseName of that record is returned. This is then passed to the **Display Algorithm 1**.
- The second is used to generate a SQL query that retrieves data on all of the nucleosomes associated with this sequence. The values that are returned by each nucleosome is the *ndsid* (the unique ID of the *nucleosomeDNASequence* record that contains the DNA sequence that is wrapped around this nucleosome) and the *histoneMods* (contains a list of unique IDs, each associated with one record in the *histoneMods* table) attributes. Each piece of data goes through a different process:
    - The *ndsid* data is used to generate a SQL query that retrieve the *dnaSequence* from the *nucleosomeDNASequence* table of the nucleosome at hand. This *dnaSequence* is used in **Display Algorithm 2**.
    - The *histoneMods* attribute as an input in a different algorithm. This algorithm takes each histone mod ID found in *histoneMods* and uses each to generate a query that finds the name of the

histone modification (*modName*), the magnitude of the effect of the histone modification (*magnitude*) and the type of effect (*effectType*) when executed. The *modName* is used in **Display Algorithm 2**. The *magnitude* and *effectType* data of each nucleosome is used in the "HMI algorithm". This algorithm find the overall change in expression of the DNA sequence (the *result*). This result is sent to **Display Algorithm 1**.

The following table outlines the different display algorithms and how they are different from each other in this diagram:

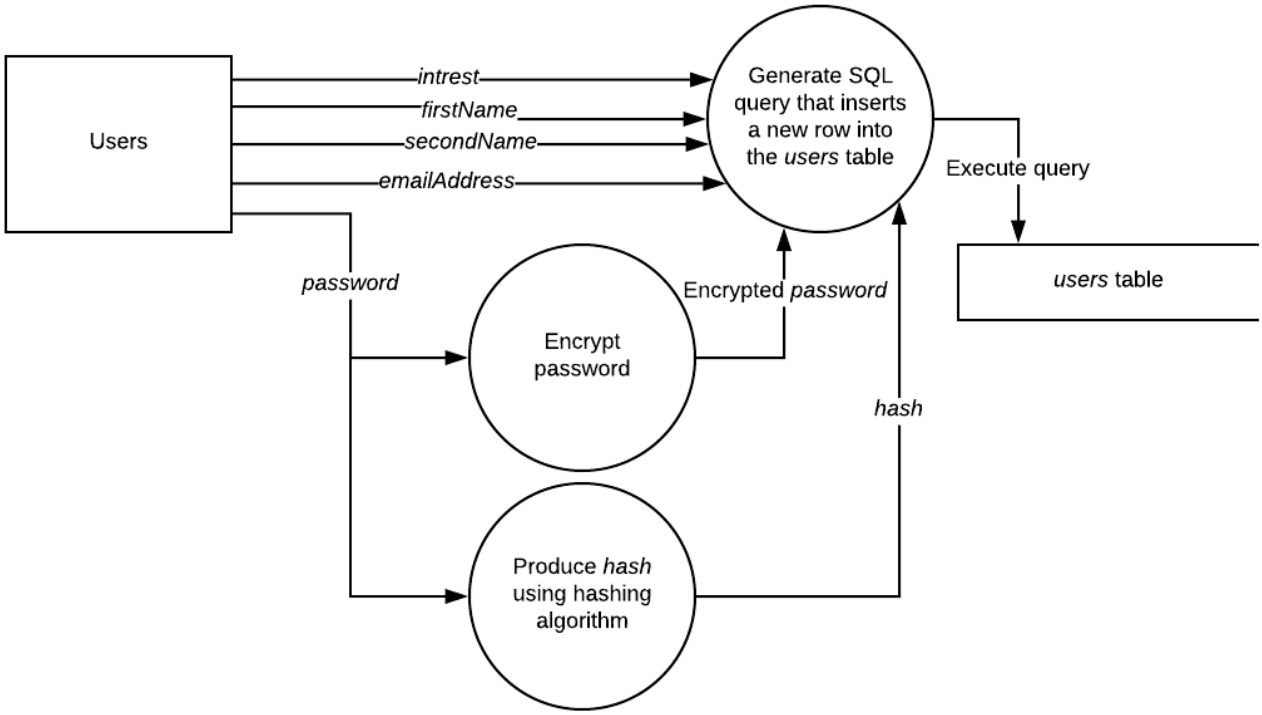
Name	Purpose
Display Algorithm 1	This generates HTML that displays key data about the sequence such as the *name*, *diseaseName*, the *result* and *notes*. It is started first so that this information can be displayed at the top. It also checks to see if the *uid* of the user who created this sequence matches that of the user who is trying to read it. If it does then an edit form is displayed so that they can edit the sequence.
Display Algorithm 2	This generates HTML for each nucleosome, displaying its DNA sequence and histone modification sequence. This displays a breakdown of the sequence, nucleosome by nucleosome. This is started second.

Both

As with the other DFD sections, I have broken down the DFD that is used for both sections into two separate parts. Again this is for clarity for the reader. These DFDs are used for the user and therefore are necessary for the other diagrams to work correctly as this is where information such as the *uid* is found and stored and how the user gets an account to use in the first place.

Please note that for these diagrams, when a user signs in they must use an email-address and password combination. These are the details a user sign in with.

Signing Up

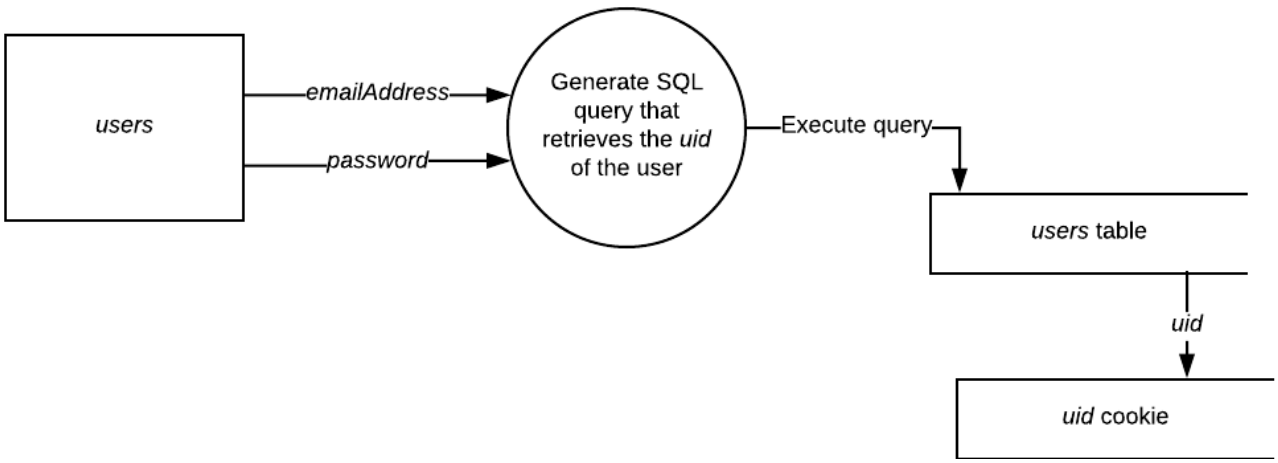


When a user wishes to create a new account they must first submit data via a HTML form. After submission the *password* they submit is used as an input for two separate algorithms:

1. The password is encrypted before being stored in the database to prevent malicious users who have accessed the database from obtaining passwords to maliciously use other user's accounts.
2. The password is entered into my own hash algorithm to produce a hash of a fixed length. This *hash* is used to verify whether the account has been activated or not. This is used to prevent spam bots from signing in and thus accessing the forum or HMI. The way this works is explained in the *Hashing Algorithm* section of this document.

The outputs of these two algorithms are then used with the other data that the user inputted in another algorithm. This algorithm first ensure that the email address is unique. This is done by comparing the inputted *emailAddress* with those of the already existing records. If no matches are found then the email address must be unique. If there are an error will be thrown. It then generates a SQL query that inserts a new record containing this data into the *users* table. Therefore creating a new account.

Signing In



If the user wishes to sign into an account they first submit a HTML form that takes two inputs:

- An *emailAddress* that is unique for each account.
- The *password* for the account.

These two inputs are used in an algorithm that generates a SQL query that retrieves the *uid* of the account if the combination of these inputs is matched by the respective attributes of any record in the *users* table. Once the query is executed, the returned *uid* is stored in a cookie so that it can be accessed later, say when creating a new sequence.

By this cookie being set the user has signed in and will be able to access the forum and HMI of the web application.

Technical Components

This is where the specific components of my solutions will be described in detail. This will help me implement each component later during my *technical solution*.

Please note that for the *SQL Queries* sections I have used example cases where appropriate and where an SQL query is static I have specified so.

Also note that for the database tables, some attributes with NOT NULL is used. NOT NULL is used because those attributes must contain a value. Before reading the database tables, please refer to the **Definitions** section of this document.

Also note that while in the *Analysis* document I discuss the tables of the database, here I discuss them from a technical viewpoint so that it is useful to me when implementing them. The purpose of their description in my *Analysis* document is to discuss their purpose and how they interact with each other (using entity-relationship diagrams) to form an overview of the backbone of my solutions. Please consult that document for any entity-relationship diagrams.

Note that for all of my PRIMARY KEY attributes I have decided to set their type as UNSIGNED INT AUTO_INCREMENT. This is because UNSIGNED INT gives me the greatest range of integer values, so that the amount of records that can be stored on the table is at its maximum. It is also because AUTO_INCREMENT ensures that each record will have a unique primary key as each value will be greater by 1 than their previous record's.

Forum

Database Tables

The forum may be made up of a large amount of threads. Each thread may contain a large amount of messages. Each message may contain a large amount of replies. Therefore the forum has the capacity to be very large and my database tables need to be able to hold many records in order to cater for this. There are three tables, each one containing records of a specific type of post.

Threads Table

Attribute Name	Type	Purpose
tid	UNSIGNED INT AUTO_INCREMENT PRIMARY_KEY	%
uid	UNSIGNED INT	$
subject	VARCHAR 2000	This is a string that contains a summary of what the thread is about.
message	VARCHAR 3000	¬
dateTime	DATETIME 8	This contains the time and date when the thread was started.

This table is used to contain the original post in a thread as well as the topic of that thread.

uid uses a UNSIGNED INT type so that it can contain the same values as the *uid* attribute from the *users* table. Therefore allowing a relationship.

The *subject* attribute should contain a summary of the topic or question of the thread and therefore should not be longer than the original post's *message*.

However I am aware that scientific jargon can be quite lengthy and therefore the *subject* requires a long length maximum length in characters. This also goes for the *message* attribute.

As the forum has the capacity to be quite large it needs to cut unnecessary storage use. VARCHAR is appropriate as it only stores as many characters as necessary up to a maximum as it does not have a fixed length. Thus, I have decided to use the VARCHAR 2000 type for the *subject* attribute and VARCHAR 3000 for the *message* type.

dateTime has a length of 8. This is the maximum length of the DATETIME datatype. I chose the maximum length because it offered the best accuracy of time so that this could be shown to the user so that they know exactly when the thread was started. This goes for all posts.

Messages Table

Name	Type	Purpose
mid	UNSIGNED INT AUTO_INCREMENT	%
uid	UNSIGNED INT	$
tid	UNSIGNED INT	This contains the ID for a particular thread.
message	VARCHAR 3000	¬
dateTime	DATETIME 8	This contains the time and date when the message was posted.

This table is used to store responses to the original post (called messages).

For the reasons outlined in the *threads* table, I have decided to use a type of UNSIGNED INT for the *uid* attribute and the VARCHAR 3000 type for the *messages* attribute.

I decided to use the type UNSIGNED INT for the *tid* attribute so that a *tid* attribute value from the *thread* table can actually be stored in it to form a many-to-one relationship with that table.

Replies Table

Name	Type	Purpose
rid	UNSIGNED INT AUTO_INCREMENT	%
uid	UNSIGNED INT	$
mid	UNSIGNED INT	This contains the ID value of the message that is being responded to.
message	VARCHAR 3000	¬
dateTime	DATETIME 8	This contains the time and date when the reply was posted.

This table is used to store responses to the original post (called messages).

For the reasons outlined in the *threads* table, I have decided to use a type of UNSIGNED INT for the *uid* attribute and the VARCHAR 3000 type for the *messages* attribute.

I decided to use the type UNSIGNED INT for the *mid* attribute so that a *mid* attribute value from the *messages* table can actually be stored in it to form a many-to-one relationship with that table.

SQL Queries

Creating A New Thread

```
INSERT INTO threads VALUES(NULL, 17, "Hello world!", "Hi everyone! Just an introduction thread!");
```

This query uses `NULL` for the *tid* attribute. This is because this attribute has the AUTO_INCREMENT property. This means that the previous record's value and increased it by one, and if no record exists the value becomes 1. However, if a record is removed the value will still increment based on the value that record had. AUTO_INCREMENT acts more like a counter.

The *uid* of the user who started this thread is 17 and has the *subject* of "Hello world!" and *message* of "Hi everyone! Just an introduction thread!".

A new row is then added to the *threads* table.

Creating A New Message

```
INSERT INTO messages VALUES(NULL, 16, 3, "Hello, I am Yamanaka!");
```

Here I used `NULL` for the *mid* value because of the reasons I outlined in the **Creating A New Thread** SQL query.

The *uid* of the user who created this message is 16, the *tid* of the thread that it is a part of is 3 and the value of the *message* attribute is "Hello, I am Yamanaka!".

This query will then insert a new row using this data into the *messages* table.

Creating A New Reply

```
INSERT INTO replies VALUES(NULL, 18, 2, "Hi Yamanaka!");
```

Here I used `NULL` for the *rid* value because of the reasons I outlined in the **Creating A New Thread** SQL query.

The *uid* of the user who created this message is 18, the *mid* of the message that is being replied to is 2 and the value of the *message* attribute is "Hello, I am Yamanaka!".

This query will then insert a new row using this data into the *replies* table.

Deleting A Thread

```
DELETE FROM replies WHERE mid = (SELECT mid FROM messages WHERE tid = 1);
DELETE FROM messages INNER JOIN threads ON messages.tid=threads.tid WHERE tid = 1;
```

When a thread is being deleted all of the associated messages and replies should be deleted too as they will no longer be able to be accessed, so it reduces unnecessary storage use. Therefore the queries should delete a thread and all of its associated replies and messages.

In all threads we specify the rows that should deleted using a primary key. This keeps the queries easy to read so that they are easier to maintain as only one attribute is specified.

In the first query, the *replies* records are deleted. The primary key used to specify them is *mid*. The value of this primary key is found by a **subquery**. A **subquery** is a SQL query within another SQL query and must be surrounded by parenthesises. This subquery retrieves the *mid* value of all messages that are associated with the thread specified. This therefore deletes all replies that are associated with messages that are associated with the thread we are deleting.

The second query deletes all of the *messages* records and the *threads* record simultaneously. This is done by the messages and threads table undergoing a **join**. This is where an attribute which is found in both tables is used to return the values of attributes of rows from different tables together as one row.

An **inner join** returns rows that have matching values in their shared attribute.

The join combines the rows of both tables using a shared attribute (*tid*) so that they are deleted together. The rows that are deleted are specified using *tid* so that only the *messages* records and *threads* records that have the specified value are deleted.

As a result, the *threads* record and all related *messages* and *replies* records are deleted.

Deleting A Message

```
DELETE FROM replies INNER JOIN messages ON replies.mid=messages.mid WHERE mid = 5;
```

This query deletes all of the replies associated with a message, as well as the message itself.

This is does simultaneously by using an inner join. The attribute use to join the rows is the *mid* attribute. The way this works is explained in the **Deleting A Thread** section.

The *mid* key is specified using a `WHERE` clause. This means that the only records that are deleted are those that contain the specified value (5) for the *mid* attribute.

Deleting A Reply

```
DELETE FROM replies WHERE rid = 23;
```

This query removes a record from the *replies* table. The record to be deleted is specified using a `WHERE` clause. The attribute used in this clause is *rid* with a value of 23. Therefore the record with a vale of 23 for its *rid* attribute is deleted.

Displaying All Threads

```
SELECT subject, tid, firstName, dateTime FROM threads INNER JOIN users ON threads.uid=users.uid;
```

The `SELECT` statement retrieves data from a table. Here the attributes are specified so that we only retrieve data that is stored as these attributes for each record in the table. We have specified the *subject*, *tid* and *firstName* attributes. The `FROM` clause specifies from which table we are retrieving data from. In this case this is the *threads* table.

However, the *firstName* attribute is not located in the *threads* table. In order to retrieve the *firstName* of the user who started the thread we have to use the `INNER JOIN` clause. This will fetch data from a row from each table and display them as a single row. The way this works is explained in the **Deleting A Thread** section.

The attribute used to join the two tables is *uid*. This lets use retrieve any and all data from the *users* table about the user who posted the thread.

The *subject* and *firstName* data will be used to display key information about a thread, next to the hyperlink that redirects the user to the page that displays an entire thread which is generated using the *tid* attribute.

The *dateTime* data is also inserted into this HTML so that the user knows which threads are the most relevant to that particular day.

The output that this query will return could look like:

subject	***tid***	***firstName***	***dateTime***
"Hello world!"	1	"Shinya"	21 Feb 2018 15:52:01
"Help on the glomerulus?"	2	"John"	21 Feb 2018 16:31:56
"Recruiting researchers for a start up co!"	3	"Alexander"	22 Feb 2018 08:12:10

Displaying A Single Thread

```
SELECT subject, message, firstName, dateTime FROM threads INNER JOIN users ON
threads.uid=users.uid WHERE tid = 3;
SELECT message, mid, firstName, dateTime FROM messages INNER JOIN users ON
messsages.uid=users.uid WHERE tid = 3 ORDER BY dateTime DESC;
SELECT rid FROM replies WHERE mid = (SELECT mid FROM messages WHERE tid = 3);
```

In order to display the original post and all of its associated messages and each their associated replies, there must be three queries - one for each type of post. In each of these queries the value of the *tid* attribute is specified because we are looking at one specific thread, so only the posts relating to that thread should be retrieved.

- For the **original post** (first query) we execute a query very similar to the one explained when **Displaying All Threads**. One of the only difference is that the value for the *message* attribute is retrieved instead of the *tid attribute*. This is because the *message* value needs to be shown to the user so that they know the details about the topic or question that the thread is about. We do not need to retrive the *tid* value because we are not using that data later in the page. All of returned values are displayed to the user.
- For the **messages** (second query) a similar query to the first query is made. The differences include the lack of a *subject* attribute being retrieved and a different table having data being retrieved from (the *messages* table). The reason why the *subject* attribute no longer has its value retrieved is because it is not located in the *messages* table as the subject of a thread is only located in the original post.
- For the **replies** (third query) a `SELECT` statement is used to retrieve the data. The attributes we are retrieving data from is *rid*. This query is only used to count the number of returned records by the query. This will be explained in more detail in the **Thread Displaying Algorithm**. The replies returned are specified by the *mid* attribute's value. This value is determined by a subquery. This subquery finds the *mid* value of each message that is associated with the thread. Therefore all of the replies returned are indirectly associated with the thread.

For the **messages** post types the *dateTime* attribute is retrieved. This is used in the generation of the HTML for each post. This is so that the user can view when each post was submitted. This therefore lets them know the relevance of the posts of the thread, as well as to the context of each post. This is not needed for the **original post** as there is only one original post.

For the **messages** there is an `ORDER BY` clause. This sets the returned records in a particular order. In this case, the order is using the *dateTime* attribute and uses `DESC`. This means that the records are returned descending from latest to earliest.

The *replies* query that retrieves the actual *replies* data can be found in the **Thread Displaying Algorithm** where it is explained.

Queue

The **queue** will be used to keep the HTML of all posts in a thread in order. This is possible because of the **first in, first out** principle. This is where the first piece of data added to a queue is the first one that is removed from the queue. Therefore, the order that the data is added into the queue will be preserve. This will be useful when outputting to the browser's viewport, as each post will be printed from latest to earliest, with the replies to one message being displayed before the replies to another.

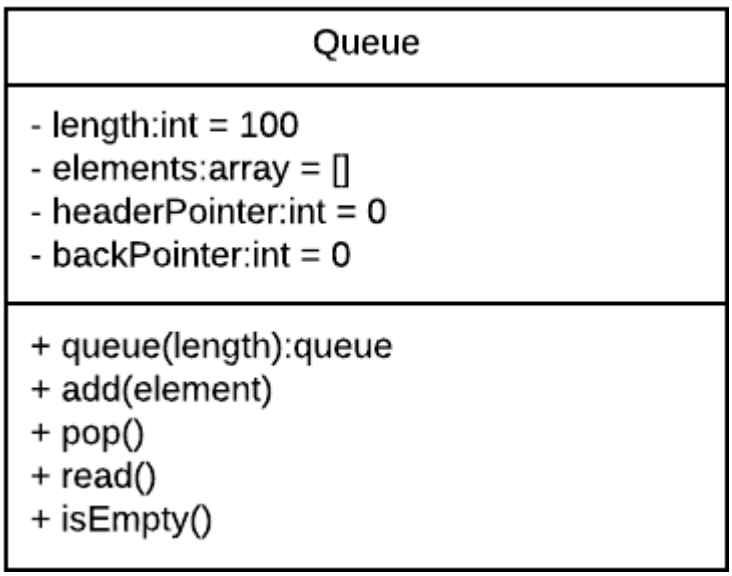
An example of a queue is a queue of people at a festival, each standing behind another. The person at the front of the queue was the first one to arrive and is the first one let in (removed from the queue). Anybody else who wants to get in has to join the back of the queue so they will be the last to enter the festival.

All queue have a fixed length. there are two types. The type of queue I wish to use is called a **circular queue**. The other is called a **linear queue** What this means in terms of my festival analogy is that there is space for up to 100 people being in the queue at any one time. In a linear queue nobody moves forward, they are just at the front of the queue by the person in front of them being removed from that queue. Therefore the queue may run out of space even if there is only 1 person left in the queue. In a circular queue people join the queue by moving in front of those at the front. The person who was previously at the front of the queue remains at the front technically speaking, as they will still be served first. This means that the queue can always make use of spare capacity and is therefore more efficient.

An **element** of a queue is one of the pieces of data that make up the queue.

Instantiation is the creation of an object from a class template.

The structure and functions of the queue can be displayed as a class diagram:



I have created a table to explain the purpose of each property and method:

Name	Type	Purpose
length	Property	This is defines the maximum length of the queue in number of elements. It is private so that is cannot be changed once the queue has been instantiated. It is set to a default value of 100 in case the user does not specify the lengths of the queue when it is instantiated.
elements	Property	This is the array that will contain the elements in a sequence. It is private so that the programmer cannot directly manipulate it so that they have to use the queue. It is given a default value that sets the property as an array (this is necessary in PHP and many other programming languages). I chose to use an array as it means I can find the front and back of the queue using indexes as the queue will be set is a sequence.
headerPointer	Property	This is an integer that holds the index of the element that is at the front of the queue. It is set to a default value of 0. This is because the array is empty by default. It is private so that it cannot be manually changed by the programmer and therefore prevents breaking the queue.
backPointer	Property	This is an integer that holds the index of the element that is at the back of the queue. It is set to a default value of 0. This is because the array is empty by default. It is private so that it cannot be manually changed by the programmer and therefore prevents breaking the queue.
queue(length)	Method	This is the **constructor** of the class. This means that this method is called to instantiate the class to create and object. This is where defaults can be set. There is an parameter in this method called "length". If the user enters a value here that sets the lengths of the queue. If not value is entered it will be set to the default of 100. It returns a queue object that can be stored in a variable. It is public so that the programmer can call it.
add(element)	Method	This method adds an element to the back of the queue. The element that is being added is specified by the argument "element". It will also increment the backPointer property so that it points to the new last element in the queue. It is public so that the programmer can call the method.
pop()	Method	This method removes the element at the front of the queue. It increments the front pointer so that it points the new element at the front of the queue. It is public so that the programmer can call the method.
read()	Method	This method returns the value of the element at the front of the queue. It is public so that the programmer can call the method.

Name	Type	Purpose
isEmpty()	Method	This method checks if the queue is empty or not. If it is it returns **true**. If it isn't it returns **false**. It is public so that the programmer can call the method.

Post Submission Algorithms

These algorithms add their respective type of post to the database in their respective tables. They are used when a user is submitting a new post. They are all somewhat similar to each other but have differences that are significant enough to separate them.

I have used pseudocode to outline how each of these algorithms will work. This is easy to read and understand as it is based on English whilst still being useful to me as it is structured enough to translate it into whatever programming language I need (PHP) as it uses the same constructs as many programming languages. However each algorithm uses functions that feature in PHP. This is necessary because PHP interfaces with MySQL (which is the management system of my database) which is needed so that the posts can be inserted into their respective tables in the database. The functions and their purpose are outlined in the table below:

Function Name	Purpose
isset(*x*)	This checks if variable *x* has a value or not.
strip_tags(*x*)	This removes any HTML tags that are in string *x*.
array_push(*x*, *y*)	This appends an element *y* to the end of an array *x*.
mysqli_query(*x*, *y*)	The executes a query *y* (a string) using the database connection *x*. It returns a result that is contextual to the query *y*.

In addition to this there is two other features of PHP that I use. This is `POST` which is known as a **superglobal** dictionary. A superglobal is a type of variable that can be accessed from any scope. It is used to store any data that is submitted via the POST protocol. A specific piece data can be retrieved by specifying its string name such as `$_POST["subject"]`. The data is submitted by HTML forms that use the POST protocol. The inputs on these forms can have names specified by a string value. These are the names used to retrieve their associated data using the `$_POST` superglobal.

There is another superglobal which is used called `GET`. This is also a dictionary and its values are retrived using the GET protocol. The value are therefore in the URL of the current page.

I also use the `$_COOKIE` superglobal to access cookies stored on the user's computer. This includes accessing the *uid* of a signed in user that is stored in a cookie as explained earlier in this document.

The `db` variable contains the connection the the database and is a global variable with the same characteristics as a superglobal.

Threads

```
1   IF isset($_POST["submitted"])
2       subject = strip_tags($_POST["subject"])
3       message = strip_tags($_POST["message"])
4       errors = []
5
6       IF subject == "" OR subject == "Subject"
7           array_push(errors, "Subject has an invalid value")
8       ENDIF
9
10      IF message == "" OR message == "Message"
11          array_push(errors, "Message has an invalid value")
12      ENDIF
13
14      IF empty(errors)
15          query = "INSERT INTO threads VALUES(NULL, "+$_COOKIE['user']+", '"+subject+"',
'".message."')"
16          result = mysqli_query(db, query)
17          OUTPUT("Success")
18      ELSE
19          FOREACH errors AS error
20              OUTPUT(error)
21          END FOREACH
22      END IF
23  ENDIF
```

On line 1 we check to see if the form has been submitted. We do this by seeing if one of the inputs has been set. This input could be hidden and have a default value, and will only be accessible to PHP once the form has been submitted.

On lines 2 and 3 we remove any HTML tags that are in the string values of the *subject* and *message* inputs. and assign them to their respective variables of the same name. This prevents malicious client-side code from entering our database and being displayed on a user's screen.

On line 4 we initiate the `errors` variable to be an array. This puts it in an appropriate scope so that later if statements can access it.

Lines 6 to 12 check if the value of either the `subject` or `message` variables is set to be the default value or if it is empty. If either of these is the case then we append an appropriate error message to the `errors` array using the `array_push()` function.

On line 14 we check if the `errors` variable is empty or not. What happens next depends on what the result of this expression is:

- If it is **empty** the run lines 15 to 17. On line 15 we create a SQL query in the form of a string. This SQL query was explained earlier in this document. The *uid* of the user who is creating this thread is the that of the user who is currently signed in. Therefore we can retrieve this *uid* which is stored in the cookie which may be named "user". To access this value we use the `$_COOKIE` superglobal. The reason it is named "user" is for security reasons is so that malicious users do not find the cookie name and realise that it is an ID as "ID" is not in the cookie name.

In order to include the value of our variables in the query, we have used **concatenation**. This is where two strings are tied together and is indicated in my pseudocode by a `+`. I have included different quotation marks within the string that surrounds the variables `subject` and `message` because those variables need to be interpreted as strings by MySQL and as the query itself is a string, different quotation marks are needed to differentiate those strings from the rest of query.

On line 16 we execute this query and store the returned result in `result`.

On line 17 we notify the user with an appropriate response.

- If it is **not empty** then errors must have occurred. Therefore we need to display those errors to the user so that they are not confused and can respond to them. The `FOREACH` statement takes an array (`errors`) and the variable *y* (`error`). It loops through each element in the array, storing the element at that time in variable *y*. We then display this error to the user. Thus, all errors are displayed to the user.

Messages

```
1   IF isset($_POST["submitted"])
3       message = strip_tags($_POST["message"])
4       tid = $_GET['thread']
5       errors = []
6
7       IF message == "" OR message == "Message"
8           array_push(errors, "Message has an invalid value")
9       ENDIF
10
11      IF empty(errors)
12          query = "INSERT INTO messages VALUES(NULL, "+$_COOKIE['user']+", "+tid+",
'".message."')"
13          result = mysqli_query(db, query)
14          OUTPUT("Success")
15      ELSE
16          FOREACH errors AS error
18              OUTPUT(error)
20          END FOREACH
21      END IF
22  ENDIF
```

This algorithm is very similar to the one in the **threads** section. The main differences here are:

- There is no *subject* data as a thread is not being submitted.
- There is no error checking on the *subject* data as it does not exist.
- We retrieve the thread ID (*tid*) of the thread that is being posted to. This is done using the `$_GET` superglobal. This is because the user can only post a message when viewing the thread, so the *tid* value should be in the URL. It should be in the URL because the ID of the thread is short and insensitive.
- The value of the `query` variable is different. It is an Insertion query into the *messages* table. Full details on the query was explained earlier in this document.

Replies

```
1   IF isset($_POST["submitted"])

3       message = strip_tags($_POST["message"])
```

```
4       mid = $_POST['mid']
5       errors = []
6
7       IF message == "" OR message == "Message"
8           array_push(errors, "Message has an invalid value")
9       ENDIF
10
11      IF empty(errors)
12          query = "INSERT INTO replies VALUES(NULL, "+$_COOKIE['user']+", "+mid+",
'".message."')"
13          result = mysqli_query(db, query)
14          OUTPUT("Success")
15      ELSE
16          FOREACH errors AS error
17              OUTPUT(error)
18          END FOREACH
19      END IF
20  ENDIF
```

This algorithm is very similar to the algorithm outlined in the **messages** section. The differences are:

- Instead of retrieving the *tid* value, the message ID (*mid*) value of the message being replied to is retrieved instead. This is done via the POST protocol. This is because the form that the user will be submitting will contain a *message* input that is can contain a large amount of data so the form should be submitted via POST.
- The `query` variable contains a SQL query that creates a new row in the *replies* table. This query was explained earlier in the document.

Thread Displaying Algorithm

This algorithm is used to retrieve and display an entire thread. The order in which the posts that make up a thread should be:

1. The original post is displayed first.
2. Each message is displayed from latest to earliest so that the most relevant posts are shown first.
    1. When it is displayed, if the message has any replies, they are all displayed from earliest to latest. This is so that the user can more easily understand replies.

It makes use of the **queue** class which was explained earlier in the document. There are some PHP-specific functions that are used in the pseudocode that outlines this algorithm. Their purpose is outlined in the table below. Other functions have been outlined earlier in this document:

Function Name	Purpose
mysqli_fetch_array(*x*)	This function returns an array of strings for a row stored in the variable *x*. This makes the data stored in result accessible. This variable contains the results returned by the `mysqli_query()` function.
mysqli_num_rows(*x*)	This function returns the number of rows in the result variable *x*. This variable contains the results returned by the `mysqli_query()` function.

```
1   IF isset($_GET["thread"])
2       queryT = "SELECT subject, message, firstName FROM threads INNER JOIN users ON
        threads.uid=users.uid WHERE tid = "+$_GET["thread"]
3       queryM = "SELECT message, mid, firstName FROM messages INNER JOIN users ON
messsages.uid=users.uid WHERE tid = "+$_GET["thread"]
4       queryR = "SELECT rid FROM replies INNER JOIN users ON
             messages.uid=user.uid WHERE mid = (SELECT mid FROM messages WHERE tid =
"+$_GET["thread"]
5       length = mysqli_num_rows(mysqli_query(queryM))+mysqli_num_rows(mysqli_query(queryR))+1
6
7       result = mysqli_query(db, queryT)
8       row = mysqli_fetch_array(result)
9       html = generateHTML(row)
10
11      htmlQueue = new queue(length)
12      htmlQueue->add(html)
13
14      result = mysqli_query(db, queryM)
15      WHILE row = mysqli_fetch_array(result)
16          html = generateHTML(row, level)
17          htmlQueue->add(html)
18          queryR = "SELECT message, firstName FROM replies INNER JOIN users ON
replies.uid=users.uid WHERE mid = "+row[1]
20          resultR = mysqli_query(db, queryR)
21          WHILE mysqli_fetch_array(resultR)
22              html = generateHTML(rowR, level)
23              htmlQueue->add(html)
24          ENDWHILE
25      ENDWHILE
26
27      DO
28          OUTPUT(htmlQueue.read())
29          htmlQueue.pop()
30      WHILE htmlQueue->isEmpty() == FALSE
31  ENDIF
```

there is one arbitrary function used in this algorithm: `generateHTML()`. This is used to describe using the `row` variable as part of the HTML that displays a given post. It will in actual fact be string containing HTML concatenated to the elements of he `row` array. I chose to describe it as a function so that the pseudocode is easier to understand as the HTML code would provide unnecessary complexity to the algorithm.

The first line of code checks to see if there if the variable `thread` exists and is set to a value in the URL via the GET protocol. If this is the case then the rest of the code can be executed.

Lines 2 to 4 each set a string that contains the query for retrieving data on the thread, messages and replies in the thread to a variable, with respective names for each post type. They use concatenation with the `$_EGT` superglobal to specify to thread ID (the *tid*) for each query. These queries were explained earlier in the document. The `$_GET["thread"]` expression retrieves the *tid* value. It is called "thread" so that malicious users can less easily tell that it is used for the ID of the thread.

After this the length of the thread in posts is calculated. This is done by using the `mysqli_num_rows()` function on the queries created earlier for the messages and the replies. As there is only ever a single original post in a thread we can just add one to this total value to find the length of the thread.

On line 7 we execute the query for retrieving the original post data and store its result in `result`. On line 8 we retrieve the data stored in `result` as an array using `mysqli_fetch_array()`. Line 9 generates the HTML using `row` and stores the resulting string in `html`.

Line 11 creates a new queue object and stores it in `htmlQueue`. This is done by using the `new` keyword (this instantiates and initialised the object) and the queue constructor (this builds the object by setting default values and calling necessary functions). We pass the `length` variable as an argument into the constructor. this is necessary because if the thread is over 100 posts long, then some posts will not be displayed as part of the thread as the queue is of a fixed size.

Line 12 adds the contents of the `html` variables to the back of the queue. This stores the HTML mark up necessary to display one post.

Line 14 overwrites the `result` variable by storing the returned value of `mysqli_query()` which takes the `queryM` (messages SQL query) as an argument. This isn't a problem as we do not use the previous `result` value again. This is repeated throughout the algorithm.

Next the messages and replies need to be added to the queue.

Lines 15 to 23 forms a while loop. This type of loop is where the nested code is repeated whilst a certain condition remains true. In this case as long as the `row` variable can be assigned a value, the while loop will continue. The condition is checked for at the beginning of the loop. This is different to a do-while loop which checks for the condition at the end of the loop. I chose to use the while loop so that the `row` variable can be used throughout the rest of the loop. It is important to note that for each iteration of the loop, the `row` variable is assigned an array of values from the next row stored in `result`. This `row` variable therefore stores the current message data based on the current iteration of the loop.

Within the while loop at line 16 and 17, we generate the HTML that displays the message post and then append this to the back the queue respectively. Line 18 generates a SQL query based on the current message. This query was not previously explained so I will discuss it now:

```
SELECT message, firstName FROM replies INNER JOIN users ON replies.uid=users.uid WHERE mid
= 5
```

> This is a `SELECT` statement. It retrieves the `message` and `firstName` attributes from each returned row from the *replies* and *users* tables respectively. There are returned together as one row by using the `INNER JOIN` clause. This uses the *uid* attribute from both tables to only return rows together as one if they have matching values. The `WHERE` clause specifies which rows should should be returned. The ones specified are those that have a *mid* (message ID) value of 5, in this case.

Line 18 concatenates the *mid* value of the current message to the query string. This is stored in `row` and as the order of the data stored in `row` is the order of attributes being selected for in the query, the *mid* value will always be stored at the index of 1 in the array. Thus, we concatenate `row[1]`.

We then execute this query using `mysqli_query()` and store the result into `resultR` on line 19.

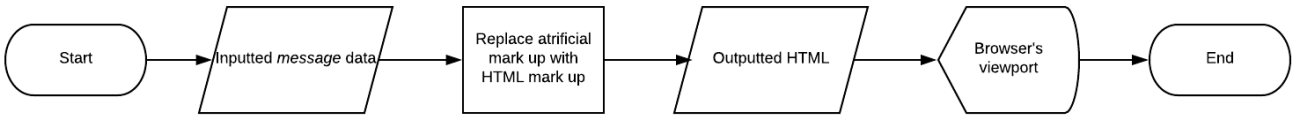
We then use this result to fetch the data in the form of an array for each row and store it into the `rowR` variable using `mysqli_fetch_array()`. This is done in a while loop in the same way as the while loop that we are already in. The code within this wile loop generates HTML mark up using `rowR` as a string and then appends this to the back of the queue. Both while loops then end.

We finally set up a do-while loop. The code within this loop repeats until the queue is empty. This code outputs the element (the HTML code) at the front of the `htmlQueue` and then removes it using `htmlQueue.pop()`. Thus making the next element in the queue the front element for the next iteration. Once all elements are removed the queue is empty and the loop finishes. Therefore each post is displayed in the order that it is added to the queue and therefore in the correct order.

Note that the `Level` argument in the `generateHTML()` function calls is used to specify the level of the current user. if it has a value of 1, then the user is an admin and therefore a delete form should generated as part of the HTML. It is not defined in the code because it has a larger scope than this algorithm as it may be used in other components of the solutions, such as a displaying the type of account the user has.

Mark Up Displaying Algorithm

This algorithm converts the artificial mark up that was submitted by a user in a post's *message* attribute and converts it to HTML mark up so it can be displayed to the user. Artificial mark up should be passed instead of the HTML mark up because any HTML mark up is removed when a post is submitted to prevent malicious client-side code from being displayed. This process is outlined in the below flowchart:



I will explain what each symbol means:

Shape	Name	Purpose
Lozenge	Beginning/End	To represent the start or end of a flowchart.
Rectangle	Process	To represent an algorithm or function.
Parallelogram	Data	To represent data.
Irregular Pentagon	Display	To represent a display.

This flow chart shows that the data from the *message* attribute of a post is taken as an input to an algorithm that replaces any of the artificial mark up with their respective HTML mark up code that can actually be displayed. The data with these replacements is then outputted and displayed in the form of a post to a user.

I am interested in the only process in this algorithm. This is where the artificial mark up is translated to HTML mark up that can actually be displayed. This algorithm will be further explained by the following pseudocode:

```
1   FUNCTION convert(message)
2       chars = ["[b]", "[/b]", "[i]", "[/i]", "[li]", "[/li]", "[/l]"]
3       HTMLChars = ["<b>", "</b>", "<em>", "</em>", "<li>", "</li>", "</l>"]
4
5       i = 0
6       DO
7           message = str_replace(chars[i], HTMLChars[i], message)
8           i++
9       WHILE i < length(chars)
10
11  RETURN message
12  ENDFUNCTION
```

There are two functions that are used in the algorithm that should be defined:

Function Name	Purpose
str_replace(*x*, *y*, *z*)	This PHP-specific function replaces and substring *x* with the substring *y* in the string *z*.
length(*x*)	This returns the length of string *x* in characters.

This algorithm is defined as a function. This is so that the code can be repeatedly used on all types of post without having to be rewritten therefore saving time and resources during both development and run time. It is not defined as a subroutine as this code returns a value. A subroutine cannot return a value.

Line 1 defines the following code as part of the function called `convert` which takes a parameter that is not optional called `message` (optional parameters have assignment operators [=] attached to them). This parameter must be a string and will contain data from the *message* attribute of a post.

Lines 2 specifies each artificial mark-up tag as an element as part of an array. Line 3 does this too but with HTML mark up tags.

Line 5 initialises the variable `i` and sets it to a value of 0.

Lines 6 to 9 form a do-while loop. In this loop `i` represents the current iteration count and is used to retrieve the artificial and HTML tags from their respective arrays by using it as an index to these arrays. I chose to use a do-while loop because the condition is checked at the end. This condition lets the loop continue for as long as `i` is less than the number of different mark up tags. However `i` increments at each loop,. Therefore the condition should be checked at the end of the loop so that the loop can end immediately after the condition is no longer met.

Line 7 replaces any instances of the current artificial mark up tag with its respective HTML mark up tag in the `message` variable. This is possible because they share the same index. Line 8 then increments the `i` variable ready for the next loop (and therefore, the next set of tags).

Histone Modification Interpreter

Database Tables

NucleosomeDNASequences Table

Name	Type	Purpose
ndsid	UNSIGNED INT AUTO_INCREMENT PRIMARY_KEY	%
DNASequence	CHAR 127	This is a string that contains the 127 bases of DNA that surrounds a nucleosome.

This table is used to store the DNA sequence of a nucleosome. The DNA sequence was not stored with the nucleosome because users might want to study the effects of different histone modification sequences on the same gene.

The *DNASequence* attribute use the CHAR datatype because it is a string of different characters, each character representing a base in DNA (A ,T, C, G), therefore representing a DNA sequence. Note that a character "N" represents an unknown base (this will be useful to researching who want to use the tool will the histone modifications on a gene that is not yet fully understood). I chose CHAR over VARCHAR because there is a fixed amount of DNA for each nucleosome and therefore that should be no change in the character length of the *DNASequence*. It has a character length of 127 because this is the exact length of DNA in base pairs that wraps around a nucleosome.

HistoneMods Table

Name	Type	Purpose
hmid	UNSIGNED INT AUTO_INCREMENT PRIMARY_KEY	%
name	VARCHAR 50	This is a unique string that identifies the histone modification, so that the user can differentiate one histone modification from another.
effect	BIT 1	This represents the type of effect the histone modification has. A value of 1 represents a repressive effect, 0 represents an activational one.
magnitude	FLOAT UNSIGNED	This represents the amount of influence the histone modification has on the expression of a gene.

This table contains all of the known histone modifications. Each one can be linked to many different nucleosomes via the primary key.

The *name* attribute contains a unique string that should be human readable and distinguishable from other histone modification names by a human. As it is as string I have used the VARCHAR datatype. This will also save storage if the name used is shorter in characters than the maximum limit. In fact, this maximum limit is 50 characters long. I chose this length because each name should be short enough to be displayed

alongside each other on the UI. However, I understand that within Biology it is important to give name that has meaning within it. For example hyperglycaemic can be broken down into two parts:

- "hyper" which means high.
- "glycaemic" which can mean "sugar levels".

Therefore this word means, high sugar levels and this can be deduced by any scientist who hasn't even heard of the word before - only these parts! Therefore I should use a similar naming system which may create large names. Hence the attribute length.

The *effect* attribute contains a value which shows the type of effect of a histone modifications. As there can only be one of two values stored, I chose to use a BIT datatype. This is also lowers the amount of storage used as only a single bit is used per record instead of the bytes offered by VARCHAR.

The *magnitude* attribute contains an FLOAT value. This type of value is a decimal one. I chose to use this datatype because it means that the size of the effect of each histone modification can be stored more precisely. It is UNSIGNED because magnitudes are only represented by positive values. UNSIGNED makes negative values invalid and therefore ensures that all records are valid.

Nucleosomes Table

Name	Type	Purpose
nid	UNSIGNED INT AUTO_INCREMENT PRIMARY_KEY	%
ndsid	UNSIGNED INT	This contains the value of the DNA sequence that the nucleosome contains.
histoneMods	VARCHAR 243	This contains a list of IDs of histone modifications that this nucleosome contains. Each mod is differentiated from another by a comma.
nsid	UNSIGNED INT	This contains an ID value that represents the histone modification sequence that this nucleosome is part of.

This table contains nucleosomes. Each nucleosome is part of a sequence. This sequence is identified by the *nsid* attribute.

The *ndsid* attribute links each nucleosome with a particular DNA sequence using the *ndsid* primary key from the *hsitoneMods* table. The *nsid* attribute links each nucleosome with a particular nucleosome sequence. Therefore, both attributes should have the same datatype as the *histoneMods* primary key.

The *histoneMods* attribute is a string that contains a sequence of *histoneMods hmid* (unique IDs). As it is a string and because the quantity of histone modifications on each nucleosome varies, I chose to use the VARCHAR datatype. This will reduce that amount of storage used if the string is shorter than the maximum value which it often will be. I chose to use a maximum value of 243 characters. This is because this is the maximum possible length of the string. Each histone modification can only be used once on a nucleosome. If every histone modification was used, based on the length of the *hmid*s in characters and including the commas that separate each ID value the maximum length would be 243 characters.

Nucleosome Sequences Table

Name	Type	Purpose
nsid	UNSIGNED INT AUTO_INCREMENT PRIMARY_KEY	%
uid	UNSIGNED INT	This contains the ID value of the user who created the sequence. This will be used to make sure that only this user can edit it and be credited for it.
name	VARCHAR 300	This contains a unique name relative to the user, as different scientists may submit discoveries about the same DNA sequence and histone modification sequence simultaneously.
notes	VARCHAR 2000 NULL	This contains important information about the sequence.
did	UNSIGNED INT NULL	This may contain the ID value of the disease that the sequence that it is associated with.

This data contains overall information on a nucleosome sequence. The primary key (*nsid*) is used to link together multiple nucleosomes from the *nucleosomes* table.

The *uid* and *did* attributes contain values that link to particular *users* and *diseases* records respectively using the respective *uid* and *did* primary keys located in these tables. Therefore these attributes should contain the same datatypes as those primary keys.

Each nucleosome sequence does not have to be associated with a disease. Therefore the *did* attribute has the NULL type.

The *name* attribute is a string that may vary in length. Therefore in order to use the minimum amount of storage possible, I have decided to use the VARCHAR datatype as it has a non-fixed length up to a maximum. This maximum is set to 300 characters. I chose this because I learnt from the titles of scientific articles that they can be quite lengthy and therefore that this attribute should have a reasonable maximum length.

The *notes* attribute is a string that may also vary in length, so it should also use the VARCHAR datatype. As this attribute can contain different types of important information, it may need a high maximum character length. Therefore the max character length I chose is 2000. I decided that it may be null to give the user more freedom as they may be case where the name serves and adequate description.

Diseases Table

Name	Type	Purpose
did	UNSIGNED INT AUTO_INCREMENT PRIMARY_KEY	%
uid	UNSIGNED INT	This contains the ID value of the user who created the disease. This will be used to make sure that only this user can edit it and be credited for it.
name	VARCHAR 300	This contains a unique name, so that the disease can be differentiated from others.
notes	VARCHAR 2000 NULL	This contains important information about the disease, such as its symptoms and other non-epigenetic causes.

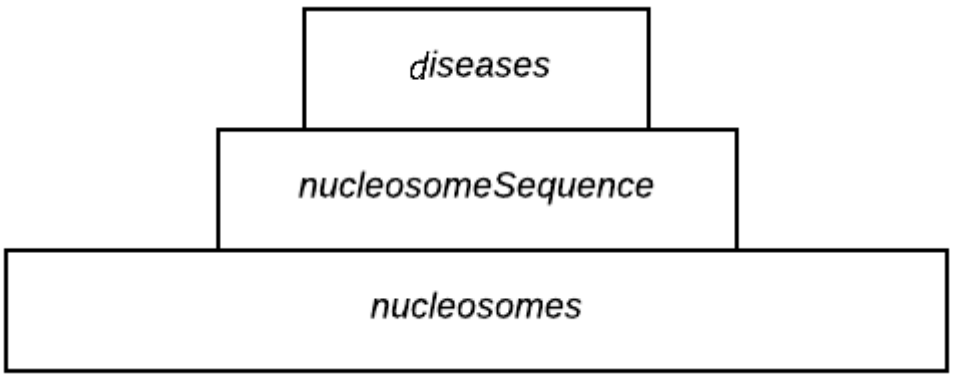
This table contains records which group sequences together based on a certain characteristic, such as a disease that they all cause. For example, one record might describe Alzheimer's Disease.

The sequences are linked to a disease via the *did* primary key. This is what allows for the many-to-one relationship between the *nucleosomesSequences* table and the *diseases* table.

The *name* and *notes* attributes have their datatypes explained in the **Nucleosome Sequences Table** section.

A More Detailed Outline

To more easily explain how these tables work in relation to each other, I have created a diagram.



Here the each table is represented by a rectangle. The higher to position of the rectangle, the higher in the hierarchy it is. The width of the rectangle illustrated the quantity of records that are within that table.

As we can see the *diseases* table's records group together many different nucleosome sequences, but have fewer records. This is because there is a many to one relationship between the *nucleosomeSequences* and *diseases* tables, which means that many different *nucleosomeSequences* records are group together by one *diseases* record.

Likewise the one *nucleosomeSequence* record groups together many different *nucleosome* records. They have fewer records than the latter for exactly the same reason as the *diseases* table.

SQL Queries

Creating A Disease

```
INSERT INTO diseases VALUES(NULL, 2, "Alzheimer's Disease", "This is a group of nucleosome
sequences that contribute to Alzheimer's Disease");
```

This query adds a new rows to the *diseases* table.

As a new record is being created, a `INSERT` query should be used. This is because this is the type of query that performs the creates a new row in a table in MySQL. We specify the `disease` table as this is the table that a new row should be inserted into.

The following paragraph explains that contents of the `VALUES` parenthesis. They specify the value for each attribute for the new record and are separated by a comma.

We pass the `NULL` keyword for the *did* attribute as its value depends on the AUTO_INCREMENT part of this attributes datatype. We then pass the *uid* of the user who created this disease (2), *name* of the disease and the *notes* that contains a description of the disease in this case (it may also include important information, important contributors, key developments, etc).

Creating A Nucleosome Sequence

```
INSERT INTO nucleosomeSequence VALUES(NULL, 3, "APP Increase Caused By Acetylation Histone
Modfiications", "This sequence increases the expression of beta-amyloid by increasing the
affinity of the promoter using histone modifications that encourage acetylation.", 61);
```

This query adds a new rows to the *nucleosomeSequence* table.

For this reason I have used the `INSERT` query. The `nucleosomeSequence` has been specified as the table for the new record to be inserted to.

We specify `NULL` for the *nsid* attribute. This is because the value of the attribute is determined by AUTO_INCREMENT. The values of the other attributes are all specified.

Although the *did* attribute may be NULL, a value is given here. This means that this sequences is associated with a disease.

Creating A Nucleosome DNA Sequence

```
INSERT INTO nucleosomeDNASequence VALUES(NULL,
"AAGACCACGGGCGCTCGGCGCCTTGGCTAATCCCCGTACATGTTGTTATAAATAATCAGTAGAAAGTCTGTGTTAGAGGGTGGAGTGACCATAAA
TCAAGGACGATATTAATCGGAAGGAGTATTCA");
```

This query creates a new record in the *nucleosomeDNASequence* table. This is why require an `INSERT` query.

The *ndsid* is set to `NULL`. This is because its value is automatically entered as its value by the AUTO_INCREMENT feature.

The *DNASequence* is set to have a string that is 127 characters in length containing four letters, each representing a base of DNA.

Creating A Nucleosome

```
INSERT INTO nucleosomes VALUES(NULL, 12, "1,13,45,16", 42);
```

This query adds a new row to the *nucleosomes* table.

Because a new record is being added to the table, an `INSERT` query is used. The `nucleosomes` table is specified as the table that the new record should be added to.

We specify `NULL` for the *nsid* attribute. This is because the value of the attribute is determined by AUTO_INCREMENT. The *ndsid* of the record is 12 (this specifies the record in the *nucleosomeDNASequence* that the nucleosome is associated with).

The *histoneMods* attribute has a values of "1,13,45,16". This means that this nucleosomes contains the histone modifications specified by the rows in the *histoneMods* table that have the *hmid* (the primary key of the *histoneMods* table) values of 1, 13, 14 or 16. Therefore this nucleosome has four histone modifications.

The nucleosome sequence that this nucleosome is associated which is specified by the *nsid* (primary key of *nucleosomeSequences* table) value, 42.

Editing A Disease

```
UPDATE diseases SET name="Age Related sequences", notes="Any sequences related to aging.
BREAKING NEWS: Histone modifications that cause aging to accelerate may have been identified."
WHERE did=11;
```

This query edits the values contains in a *diseases* record. Therefore an `UPDATE` query is used.

Firstly the table that is being edited is specified. In this case it is `diseases`. This is necessary so that the query can be executed on a particular table.

The `SET` clause specifies which attributes are being updated with new values, as well as the new values to be used.

After this the *name* and *notes* attributes are updated with new values. As we can see, this query is meant to update the *notes* attribute as the some breaking news is added. However, it is important to note that when the user wants to update only one attribute, because the attributes that are updated are fixed, the attributes that are not updated will just receive the old value so that it stays the same and so that the attribute that is being updated is the only one that is changed. Therefore we can always use this query to update one or both attributes. For example:

- If a user only wants to change the *notes* from "Any sequences related to aging." to "Any sequences related to aging. BREAKING NEWS: Histone modifications that cause aging to accelerate may have been identified.".
- Then the *name* attribute will not be changed.
- So the *name* is set to its original value and the *notes* are set to new value. Therefore only the *notes* is changed.

A `WHERE` clause is used to specify the record that is being updated. The record is specified using the primary key. In this case the primary key has a value of 11.

Editing A Nucleosome Sequence

```
UPDATE nucleosomeSequence SET name="APP Expression Increase", note="Exactly as named. NOTE: New
histone modification have been added that were recently discovered that contribute to the
expression increase.", did=NULL WHERE nsid=7;
```

This query changes the data contained within a *nucleosomeSequence* record. This is done by creating an `UPDATE` query.

Firstly the *nucleosomeSequence* table is specified to be the table that has its records updated.

After the within the `SET` clause we find that three attributes can be updated: *name*, *notes* and *did*. The first two are self explanatory, but the third requires a little more information. A nucleosome sequence can be associated with a particular disease. However, as scientific developments continue, it may be discovered that a particular histone modification sequence may affect another disease much more than the disease it is currently associated with. In that situation the disease in which it is associated with should change. This therefore involves changing the *did* value.

However, in this case we have set it to `NULL`. This could be because of new evidence that disproves thatthis sequence is associated with any disease or any other group, therefore it should contain no value to represent no disease association.

We also specify which nucleosome sequence is being edited using the `WHERE` clause and the primary key (*nsid*) of this table. This lets the user update one specific nucleosome sequence.

Editing A Nucleosome

```
UPDATE nucleosomes SET ndsid=478, histoneMods="7,8,32,56,78,13,51" WHERE nid=5871;
```

This query edits the data for a specific nucleosome. This lets the user change the histone modifications and DNA sequence that are apart of the nucleosome. Because we are editing the data for a nucleosome we use an `UPDATE` query.

The `SET` clause specifies the new data for the *ndsid* and *histoneMods* attributes. The *ndsid* contains a value that refers to a primary key in the *nucleosomeDNASequences* table, therefore indirectly referencing to a DNA sequence. Therefore if the *ndsid* attribute is changed, the DNA sequence of the nuclosomes is also changed.

The *histoneMods* attribute contains a list of *hmid*s. These are the primary keys for records in the *histoneMods* table. Therefore, each primary key specifies a histone modification. Therefore if this attribute changes, the histone modifications for this particular nucleosomes also changes.

In order to specify which record should be updated, a `WHERE` clause is used. This specifies the nucleosomes using the primary key (*nid*). In this case the record with a primary key value of 5871 is updated. This lets the user edit a particular nucleosome.

Displaying A Disease

```
SELECT name, notes, uid FROM diseases WHERE did = 13;
SELECT name, notes, nsid FROM nucleosomeSequences WHERE did = 13;
```

These queries are used to retrieve overall data on a disease and all of the associated sequences. Therefore, a `SELECT` query is used. The first query retrieves overall data on the disease. The second query retrieves data on each associated nucleosome sequence.

Firstly the attributes are specified. This is done so that the query can retrieve specific types of data to from a table. The attributes retrieved are different for each query:

- The **first** query retrieves data from the *name* and *notes* attributes. Because all of the associated nucleosome sequences are grouped by the characteristics shown in the *name*, and because the *notes*

may contain important information on all of them, these will be displayed at the top of the page, so the query should be executed first. The *uid* attribute is retrieved so that *uid* of the user who created this disease can be compared to the *uid* of the current user.
- The **second** query retrieves data from the *name* and *notes* attributes of the *nucleosomeSequences* table. This is so that the user know some overall information about each each sequence. The *nsid* is the primary key of this table and is also retrieved. This is so that it can be used to create a hyperlink that redirects the user to a page on the details of this particular nucleosome sequence.

Both queries need to specify which records the data is being retrieved from. This is done by using the `WHERE` clause. Because the nucleosome sequences are associated with a disease, they will share the same *did* value. Therefore both queries specify the same value for *did* in the `WHERE` clause.

Displaying A Sequence

```
SELECT nucleosomeSequences.name, notes, diseases.name, did FROM nucleosomeSequences INNER JOIN
diseases ON nucleosomeSequences.did = diseases.did WHERE nsid = 16;
SELECT nid, histoneMods, dnaSequence FROM nucleosomes INNER JOIN nucleosomeDNASequences ON
nucleosomes.ndsid = nucleosomeDNASequences.ndsid WHERE nsid = 16 ORDER BY nid ASC;
```

These queries retrieve overall data of a particular nucleosome sequence, as well as data on each nucleosome that is associated with each nucleosome sequence. As data is being retrieved `SELECT` queries are used. There are two separate queries:

- The **first** query retrieves overall data concerning the nucleosome sequence. It retrieves the *notes* and *name* attribute from the nucleosomeSequences table and the *name* attribute from the *diseases* table. However, because both of these tables have attributes with the same name (*name*) the tables that each attribute comes from must be specified as a prefix followed by `.`. This shows that the attribute is a member of that table object. Hence `nucleosomeSequences.name`. We also retrieve the *did* attribute that is present in both tables. This is so that it can be used to generate a hyperlink from the current page to a page that offers further details on the particular disease that the sequence is associated with.

  In order to retrieve these data from different tables on the same row, the `INNER JOIN` clause is used. The attribute used that links the two tables together is the *did* attribute.
- The **second** query retrieves data on each nucleosome. The *histoneMods* attribute are retrieved from the *nucleosomes* table. It is used in the **Sequence Displaying Algorithm**. The *dnaSequence* attribute is retrieved from the *nucleosomeDNASequence* table. An `INNER JOIN` clause is used to return the data from both tables together as a single row. It uses the *ndsid* as the attribute which joins the two tables. It uses the `ORDER BY` clause to retrieve the data from the first nucleosome to last (this keeps the sequence in the submitted-order). This uses the *nid* attribute as the first nucleosomes will have a lower value than the later ones due to the AUTO_INCREMENT feature.

Both tables specify which records are being retrieved using the `WHERE` clause. It uses the *nsid* key because it is used to establish a relationship between the *nucleosomes* and *nucleosomeSequences* tables (as outlined in my *Analysis*).

NDSID Retrieving Algorithm

This algorithm finds the *ndsid* value of a record that contains a specified DNA sequence in the *dnaSequences* table.

```
1   FUNCTION findNdsid(dna)
```

```
2       query = "SELECT ndsid FROM nucleosomeDNASequences WHERE dnaSequence = '"+dna+"'"
3       result = mysqli_query(db, query)
4       ndsid = -1
5       num = mysqli_num_rows(result)
6       IF num == 1
7           ndsid = mysqli_fetch_array(result)[0]
8       ELSE
9           query = "INSERT INTO nucleosomeDNASequences VALUES(NULL, '"+dna+"')"
10          result = mysqli_query(db, query)
11          result = mysqli_query(db, "SELECT ndsid FROM nucleosomeDNASequences ORDER BY ndsid
DESC LIMIT 1")
12          ndsid = mysqli_fetch_array(result)[0]
13      ENDIF
14      RETURN ndsid
15  END FUNCTION
```

This algorithm is contained within a function so as to reduce the complexity of the **Sequence Submission Algorithm**. This will make it easier to read after development and therefore make maintenance quicker. In line 1 a parameter (`dna`) is defined. This is not optional.

Line 2 generates a query that retrieves the *ndsid* from an already existing record with a given DNA sequence. This query is explained further below:

> ```
> SELECT ndsid FROM nucleosomeDNASequences WHERE dnaSequence =
> 'AAGACCACGGGCGCTCGGCGCCTTGGCTAATCCCCGTACATGTTGTTATAAATAATCAGTAGAAAGTCTGTGTTAGAGGGTGGAGTGACC
> ATAAATCAAGGACGATATTAATCGGAAGGAGTATTCA';
> ```
>
> As this query is retrieving data from a record a `SELECT` query is used. The attribute that is being retrieved is the *ndsid* attribute. In order to specify which record is being retrieved a `WHERE` clause is used. It specifies the value of the *dnaSequence* attribute that the record should contain.
>
> A maximum of one record can be returned by this query. This is because each record has a unique *dnaSequence*.

This query is then executed. Line 4 initialises the `ndsid` variable. This is done now so that it has a larger scope so that its value can still be changed within the following IF statement and then returned outside of that IF statement.

After this the number of records returned is stored into the `num` variable. This variable is then checked on line 6.

- If one result has been returned then the *ndsid* value can be immediately retrieved from `result` using `mysqli_fetch_array()`. However, instead of returning an array, I chose to use this function as an array itself and return the only element in the array to the `ndsid` variable.
- If no results have been returned then a new record should added to the *nucleosomeDNASequences* table. This is so that we can retrieve its *ndsid* value after it has been added. The query is generated using the `dna` variable on line 9 and is executed on line 10. Line 11 executes a query that is defined below:

  > ```
  > SELECT ndsid FROM nucleosomeDNASequences ORDER BY ndsid DESC LIMIT 1
  > ```

> This query retrieves the *ndsid* value from the most recently added record. Therefore a `SELECT` query is required.
>
> The record with the greatest *ndsid* value will also be the most recent due to the AUTO_INCREMENT feature. Therefore the `ORDER BY` clause is used to order the returned records. The `DESC` keyword is used to return records in from those with the greatest `ndsid` value to the lowest. There ca only be one record that is the most recent. Therefore the `LIMIT 1` keyword is used, where `LIMIT` forces the query to only return a capped amount of records and `1` what this cap is in records.

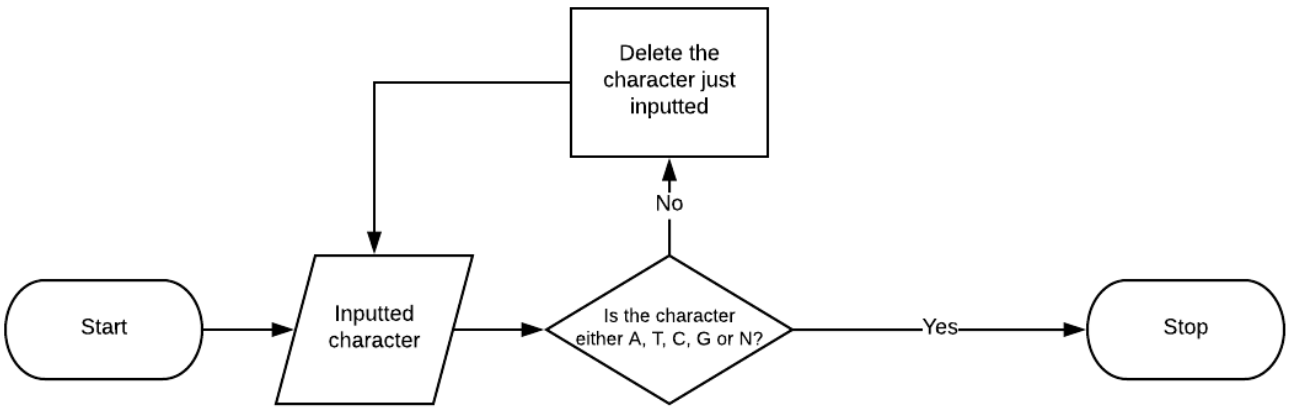
The *ndsid* value is then retrieved from this query using `mysqli_fetch_array()`.

The `ndsid` is then returned on line 14. The fact that a value has been returned is exactly why this algorithm is a function and not a subroutine.

DNA Validating Algorithm

This algorithm checks to see if the DNA data inputted is valid or not. however, this is all checked on the client-side before the data is submitted. This reduces the complexity of the **Sequence Submission Algorithm** and therefore increases the speed of maintenance as it will be easier to read. Because this algorithm will be used on the client-side of my solution, it will most likely be written in JavaScript. This is because it is the only programming language that operates on the client's computer that is part of my **Technology** list outlined in my *Analysis* document.

I have chosen to describe this algorithm using a flowchart because the algorithm is simple enough that it does not need to be incredibly structured for me to translate it into actual code later.



This flowchart uses the symbols described earlier in this document.

When the user inputs a new character, they create a new piece of data. This algorithm takes place on each character. This character is checked against a single decision. If the character is either A, T, C, G or N then the algorithm stops and the next character can be entered- this character is valid. However if it is not, then the invalid character is deleted and the user is then asked to input a valid character. Because this algorithm is applied to each character, the entire sequence will be valid.

Sequence Submission Algorithm

This algorithm allows for a new nucleosome sequence to be added to the database using data that a user has entered. I have decided to describe this algorithm using pseudocode because it is structured enough to let me translate it into source code later, yet it uses English so that the reader can understand it. Some of the PHP-specific functions used are defined in the **Forum** section of this document. Any others are defined in the following table:

Function Name	Purpose
explode(*x*, *y*)	Returns an array of strings from a single string (*y*) that has been separated by the substring *x*.
str_split(*x*, y)	Returns an array of strings from a single string(*x*) that has been split up into substring of a character length *x*.
mysqli_num_rows(*x*)	Returns an integer that represents the number of rows that are stored in the result *x*. This result is produced by mysqli_query().

```
1   IF isset($_POST["submitted"])
2       dnaSequence = $_POST["dnaSequence"]
3       histoneMods = $_POST["histoneMods"]
4       did = $_POST["disease"]
5       name = mysqli_real_escape_string(strip_tags($_POST["name"]))
6       notes = mysqli_real_escape_string(strip_tags($_POST["name"]))
7       errors = []
8       num = myqi_num_rows(mysqli_query(db, "SELECT nsid FROM nucleosomeSequences WHERE
name='"+name+"' AND uid="+uid+"))
9       IF name == "Name" OR name == "" num > 0
10          array_push(errors, "Name has an invalid value")
11      ENDIF
12
13      IF name == "Notes" OR notes == ""
14          array_push(errors, "Notes has an invalid value")
15      ENDIF
16
17      IF dnaSequence == "ATCG" OR dnaSequence == ""
18          array_push(errors, "DNA sequence is invalid")
19      ENDIF
20
21      IF histoneMods == ""
22          array_push(errors, "Histone Mods are invalid")
23      ENDIF
24
25      IF empty(errors)
26          query = "INSERT INTO nucleosomeSequences VALUES(NULL, "+uid+", '"+name+"',
'"+notes+"', "+did+")"
27          result = mysqli_query(db, query)
28          result = mysqli_query(db, "SELECT nsid FROM nucleosomeSequences ORDER BY nsid DESC
LIMIT 1")
29          nsid = mysqli_fetch_array(result)[0]
30
31          modsArray = explode("|", histoneMods)
32          dnaArray = str_split(dnaSequence, 127)
33
34          IF length(modsArray) < length(dnaArray)
35              array_push(modsArray, "")

36          ELSEIF length(modsArray) > length(dnaArray)
```

```
37              array_push(dnaArray,
"NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN")
38          ENDIF
39
40          i = 0
41          DO
42              mods = modsArray[i]
43              dna = dnaArray[i]
44              ndsid = findNdsid(dna)
45              query = "INSERT INTO nucleosomes VALUES(NULL, "+ndsid+", '"+histoneMods+"',
"+nsid+")"
46              result = mysqli_query(db, query)
47              i++
48          WHILE i < length(modsArray)
49          OUTPUT("Success")
50      ELSE
51          FOREACH errors AS error
52                  OUTPUT(error)
53           END FOREACH
54      ENDIF
55  ENDIF
```

This algorithm is the largest in this solution. I will therefore explain it in detail so that the reader can understand it fully.

This algorithm is similar in some ways to the submission of a post in the **Forum** section of this document. Line 1 shows this by checking for the submission of a form via POST. This was explained in the **Forum** section.

Lines 2 to 6 retrieve data from this form via the POST protocol. The `name` and `notes` variables are set to data that has been exposed to the `strip_tags` and `mysqli_real_escape_string` functions. This is to prevent any malicious client-side code from being submitted to the database, and to ensure that quotes within the string are properly escaped so that the generated queries are not disrupted, respectively. The `dnaSequence` data does not need to be exposed to these functions as it will have already undergone validation as explained in the **DNA Sequence Validating Algorithm**.

Line 7 initialises the `errors` variable.

Lines 8 to 24 perform error checks on the data after being assigned to a variable to ensure that the data is valid. Line 8 finds the number of records that have the same name as the one that was submitted and was submitted by the current user. This is further explained by the following SQL query:

> ```sql
> SELECT nsid FROM nucleosomeSequences WHERE name="APP Expression Increase" AND uid=7
> ```
>
> This query is used to find the number of sequences that have the same name as the one that was submitted and nothing more. Therefore a `SELECT` query should be used. However, the attribute with he smallest values should be the only attribute to be retrieved so that the query is as quick as possible because we are not actually going to use any of the retrieved data! Therefore the *nsid* attribute is used.

> The `WHERE` clause is used to specify the *name* of the sequence and the *uid* of the user who submitted it. Because two attributes are being used to specify which records to retrieve the AND keyword must be used. This means the query will only retrieve data from records that contains both of the specified values.

Using this result of the query in `mysqli_num_rows` the number of rows that already have this name for this particular user is found and stored in `num`. This value is then used on the next row to check if the *name* is valid or not. The *name* is only valid if it has not already been used by the user. Therefore the value of `num` should be 0. This therefore ensures that each nucleosome sequence has a unique value relative to the user.

If any errors are caught, an error message is stored in the `errors` array.

Line 25 checks if the `errors` array is empty. Depending of if this is true of not, one of two things can happen:

- If it is **empty** then the data can be submitted to the database. We first generate a query which will insert a new record into the *nucelosomeSequences* table. This query was explained earlier in the document. This is then executed.

  The value of this record's *nsid* attribute needs to be found so that it can be used when generating the queries for the nucleosomes associated with this sequence. Therefore on line 28 a new query is executed. It is explained below:

  ```
  SELECT nsid FROM nucleosomeSequences ORDER BY nsid DESC LIMIT 1
  ```

  > This query is used to retrieve the *nsid* value of the latest record that was added to the database. Therefore we use a `SELECT` query. The latest nucleosome will have the highest *nsid* value. This is because of the AUTO_INCREMENT feature. Therefore the returned records should be ordered based on the *nsid* value from highest to lowest. This is done by the `ORDER BY` clause using the `DESC` keyword (which stands for descending). There can only be one latest row, therefore onyl one record should be retrieved. This is done by the using `LIMIT 1`. Where `1` represents the number of rows that should be returned and `LIMIT` forces the query to only return a capped amount of queries.

  On line 29 the actual value of *nsid* is retrieved so that it can be used programmatically using `mysqli_fetch_array`. In stead of returning an array I chose to specify which element of the array should be returned and stored in `nsid`. This was done by treating the function as an array which is permitted in PHP. I chose to do this instead of returning an array because the array would only contain one element anyway and would require more operations to store the data in `nsid`. So this method was more efficient.

  On line 31 the histone modifications list that was inputted by the user is split into nucleosomes-sized chunks based on the `|` character. Therefore this character should represent the end of the histone modifications list of one nucleosomes and the beginning of another. This uses the `explode()` function, so an array of this nucleosome-sized chunks are returned.

  Line 32 splits the inputted DNA sequence into nucleosome-sized chunks based on size. Each nucleosome must have a fixed length of DNA of 127bp. Therefore the DNA is split into 127-character chunks. This is achieved by the `str_split()` function which also returns an array of these chunks.

Line 34 checks to see if the number of histone modification chunks is less than the number of DNA chunks. If that is the case than the user has inputted too few histone modification chunks. This must be account for so that the correct number of nucleosomes are a part of the sequence. Therefore using then `array_push()` function a new chunk is added that contains no histone modifications. This will correct this imbalance.

Line 36 checks to see if the number of histone modification chunks is greater than the DNA chunks. If it is then using the `array_push()` function unknown DNA is added. Following genomic standard, that character N is used to represent an unknown base. As there are 127bp wrapped around a nucleosome 127 characters of N are added to the DNA chunks array. This should rebalance the numbers.

It is important to note that each histone modification chunk has a respective DNA chunk. This is defined by each histone modification chunk's index in the `modsArray`'s respective DNA chunk having the same index in `dnaArray`.

After this `i` is initialised with a value of 0. This will be used in the following loop.

On lines 41 to 48 a query is generated for each nucleosome and executed using a do-while loop. I chose to use a do-while loop because `i` is going to be used as an index for each array and will increment with each iteration of the loop. As it increments at the just before the end of the loop and because a do-while checks the condition at the end of a loop (this condition checks if `i` is less then the length of `modsArray`) I chose to use a do-while loop.

The first lines of the loop retrieves the histone modifications chunk and its respective DNA chunk and stores it in `mods` and `dna` variables respectively. The `ndsid` variable finds the *ndsid* value of the record that stores the current DNA chunk using the earlier defined `findNdsid()` function. These variables are then used to generate a query that adds a new record to the *nucleosomes* table which was explained earlier in this document. This query is then executed and the `i` variable increments.

After the loop has finished a message is outputted so that the user knows submission was a success.

- If the *errors* variable is not empty, some errors must have been caught. Therefore each error is outputted to the user using a FOREACH loop as explained in the **Forum** section of this document.

Disease Submission Algorithm

This algorithm is used to when a user is creating a new disease. I will explain it using pseudocode because this algorithm uses a lot of programming constructs and therefore should be explained programmatically.

```
1   IF isset($_POST["submitted"])
2       name = mysqli_real_escape_string(strip_tags($_POST["name"]))
3       notes = mysqli_real_escape_string(strip_tags($_POST["notes"]))
4       errors = []
5
6       IF name == "Name" OR name == ""
7           array_push(errors, "Name is invalid")
8       ENDIF
9
10      IF notes == "Notes" OR notes == ""
11          array_push(errors, "Notes is invalid")
12      ENDIF
13
14      IF empty(errors)
15          query = "INSERT INTO diseases VALUES(NULL, '"+name+"', '"+notes+"', "+uid+")"
16          result = mysqli_query(db, query)
```

```
17          OUTPUT("Success")
18      ELSE
19          FOREACH errors AS error
20              OUTPUT(error)
21          END FOREACH
22      ENDIF
23  ENDIF
```

As in other submission algorithms, this algorithm starts by checking to see if the HTML form was submitted via POST. This means that this code will only execute once the HTML form is submitted.

Lines 2 and 3 retrieve data from these forms. The `name` and `notes` variables are set to data that has been exposed to the `strip_tags` and `mysqli_real_escape_string` functions. This is to prevent any malicious client-side code from being submitted to the database, and to ensure that quotes within the string are properly escaped so that the generated queries are not disrupted, respectively.

The `errors` variable is then initialised so that it has the appropriate scope to be used throughout the algorithm.

Lines 5 to 13 include code that error checks the `name` and `notes` variables. Any error messages are appended to the `errors` array.

Line 14 checks to see if the the `errors` array is empty or not:

- If it is **empty** then there are no errors and the query can be executed successfully. Therefore, a query is generated that adds a new row to the *diseases* table. This query was explained earlier in the document. This query is then executed and a message is outputted to the user so that they know that submission was a success.
- If it is **not empty** then there were some errors caught. Each error is outputted to the the user using a FOREACH loop. This type of loop was already explained in this document.

Histone Modification Interpretation Algorithm

This algorithm is used to interpret the overall change in expression caused by a histone modification sequence on for any sequence of DNA. This overall change is represented as a number and may also be known as the **result**. The more negative the result the more repressive it is. The more positive, the more activational it is. Therefore a result with a positive value represents a increase in expression whereas a result with a negative expression represents a decrease in expression. is based off of the mathematical formula described in my *Analysis*. Therefore this algorithm uses mathematics and is too complex for a flowchart. Therefore I am explaining it using pseudocode:

```
1   FUNCTION findResult(histoneModsArray)
2       result = 0
3       FOREACH histoneModsArray AS mod
4           query = "SELECT effectType, magnitude FROM histoneMods WHERE hmid = "+mod
5           queryResult = mysqli_query(db, query)
6           row = mysqli_fetch_array(result)
7
8           IF row[0] == 1
9               result -= row[1]
10          ELSE
11              result += row[1]
12          ENDIF
```

```
13      END FOREACH
14      RETURN result
15  END FUNCTION
```

The algorithm is contained within a function. This is so that complexity is reduced from the **Sequence Displaying Algorithm**, therefore making that algorithm easier to read and maintain. It has one compulsory parameter called `histoneModsArray`. This contains an array of primary keys for the *histoneMods* table. This is useful when extracting data from that table. The primary key for that table is *hmid*. I chose a function over a subroutine because this algorithm returns a value and a subroutine cannot do that.

Line 2 initialises the `result` variable. This will contain the change in expression caused by a histone modification sequence. It is initialised now so that it has a suitable scope so that it can be accessed by loops and IF statements that have a smaller scope, whilst still being able to return the result.

Lines 3 to 13 defines a FOREACH loop. Here each element of the `histoneModsArray` is stored in the `mod` variable and is iterated through the loop once. This mod contains a single *hmid* value. This value is then used on Line 4 to generate a query. This query is explained below:

> ```
> SELECT effectType, magnitude FROM histoneMods WHERE hmid = 7
> ```
>
> This query is used to retrieve data from the *histoneMods* table. Therefore a `SELECT` query is used. The data being retrieved are those from the *effectType* and *magnitude* attributes. In order to specify a particular record to retrieve data from a `WHERE` clause is used. The primary key of the table is used to specify the record as it is unique so only one record will be selected which allows the user to have an appropriate degree of accuracy when selecting data.

The query is executed on line 5 and data is stored as an array on line 6 in the `row` variable.

Because the order of the data in the `row` array is the same as the the order of the attributes specified in the query, the *effectType* of the histone modification is stored at `row[0]` and the *magnitude* is stored at `row[1]`.

Therefore line 8 determines whether or not the *effectType* has a value of 1 or not:

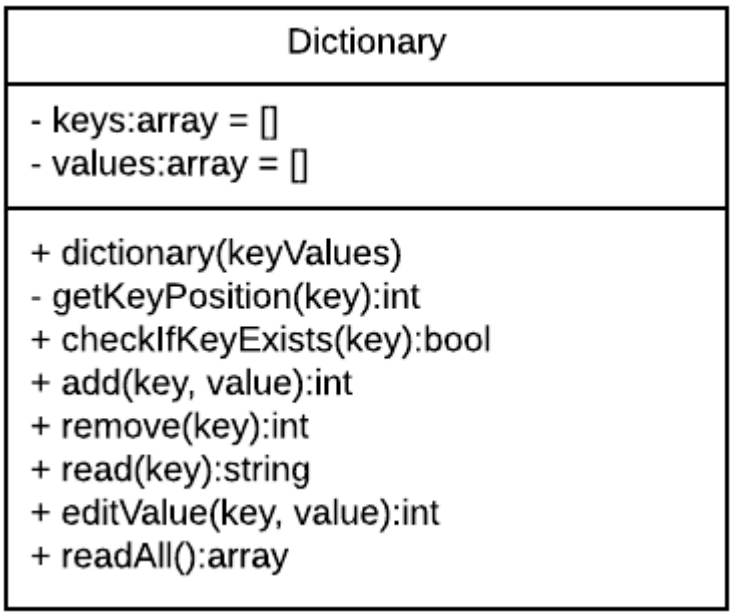
- If it **does** then this histone modification is recessive. The magnitude of this histone modification is subtracted to the result. This is the same as multiplying the magnitude by -1 and then adding it to the result, so it is still uses the formula defined in my *Analysis*.
- If it **doesn't** then this histone modification is activational. The magnitude of this histone modification should be added to the result.

After the loop has finished, all of the histone modifications that were specified by `histoneModsArray` have been accounted for and the result can be returned.

Dictionary

A **dictionary** is a data structure that associates keys with values.For example, a key "word" can be associated with a value "Hello world!". Each key must be unique, but the values do not. This is different from an array because instead of using an index as a key (which is strictly an integer) other datatypes can be used, such as a string.

A dictionary will be used in the **Sequence Displaying Algorithm**. This dictionary will be an instance of a class. I am therefore using object orientated programming to achieve this. I will explain the class using a class diagram:



The symbols used in this diagram were explained earlier in this document as well as the design of this diagram.

A dictionary can be implemented using arrays. The *keys* property contains each key in the dictionary and stores them as elements to an array. The *values* property contains each value in the dictionary and stores them as elements in an array. The way a key is associated with a value is through their shared index. For example if a key "a" had an index of 1 in the *keys* array, then its value "b" also has an index of 1 in the *values* array.

The *dictionary(keyValues)* method is a constructor. This means it sets up the default variables for an object when it is instantiated. In PHP (the programming language that this class will be defined in) a constructor must share the same name as the class itself. It is called when a new object is instantiated. It takes a parameter *keyValues*. *KeyValues* is a single array that contains a key followed by its value followed by another key... For example, key "a" is followed by value "c" which is then followed by key "b" and its value "d". This is optional for the programmer to include when calling this method as there may be no initial values for the dictionary. This array is then split into two separate arrays which are then assigned to the *keys* and *values* properties. It is public so that it can be called by the programmer.

The *getKeyPosition(key)* method finds the index of a specified key (*key*) in the *keys* array. It is only used by other methods within the class so it is private so that the programmer cannot access it. It returns the index as an integer. If the index cannot be found than an erroneous code of -1 is returned to signify that the key does not exist.

The *checkIfKeyExists(key)* method returns true if a key exists within the dictionary. It returns false if a key does not exist. It uses the *getKeyPosition()* method to determine this because this function will return a value of -1 if the key does not exist. It is public so that it can be called by the programmer.

The *add(key, value)* method appends a key/value combination at the end of the dictionary. This is done by appending the key at the end of the *keys* array and likewise for the value and *values* property. As both are appended to the ends of these arrays they will share the same index and thus be related to each other. It is public so that it can be called by the programmer. It will return an integer of 0 if this was a success. if the key already exists, then an error code of 1 is returned.

The *remove(key)* method deletes a key/value combination from the dictionary. This is done by finding the index for the specified *key* in the *keys* property using *getKeyPosition()* method. This index is then used to remove both the value from the *values* array and the key from the *keys* array. It is public so that it can be called by the programmer. It returns an integer value of 0 if the deletion was a success. If the key does not exist than an error code 1 is returned.

The *read(key)* method returns the value associated with the specified *key*. This is done by using the *getKeyPosition()* method to retrieve the index of the key, so that that same index can be used to retrieve the value. This value is returned as a string. If the key does not exists an integer error code of 1 is returned instead.

The *editValue(key, value)* method changes the value associated with the specified *key* to the specified *value* parameter. This done by finding the index of the key using the *getKeyPosition()* method, and then using that index to assign that index the new value in the *values* property. It returns an integer value of 0 of the editing was successful. if the key could not be found an error code of 1 is returned.

The *readAll()* method combines the *keys* and *values* property into a single array. This is so that the keys and values share a similar pattern to the *KeyValues* array in the constructor. This array is then returned.

Sequence Displaying Algorithm

This algorithm is used to display a single sequence. The information displayed should include:

- **Overall** information on the sequence, such as any important information, the name of the sequence and the entire DNA sequence that the nucleosomes contain
- The **change in expression** of the DNA sequence, calculated by the **Histone Modification Interpretation Algorithm**.
- A breakdown of the nucleosome sequence, displaying each nucleosome with its DNA sequence and its histone modifications.

I have decided to explain this algorithm using pseudocode. This the algorithm is very complex so it would be simpler to read the algorithm using pseudocode than a flowchart.

```
1   IF isset($_GET["sequence"])
2       nsid = $_GET["sequence"]
3       overallComponents = new dictionary(["DNA", "", "histoneModifications", ""])
4
5       num = mysqli_num_rows(mysqli_query(db, "SELECT nid FROM nucleosomes WHERE nsid =
"+nsid))[0]+1
6       nucleosomeQueue = new queue(num)
7       query = "SELECT nucleosomeSequences.name, notes, diseases.name, uid, did FROM
nucleosomeSequences INNER JOIN diseases ON disease.did=nucleosomeSequences.did WHERE nsid =
"+nsid
8       result = mysqli_query(db, query)
9       row = mysqli_fetch_array(result)
10      topHTML = generateHTML(row)
11
12      query = "SELECT nid, histoneMods, dnaSequence FROM nucleosomes INNER JOIN
nucleosomes.ndsid = nucleosomesDNASequence.ndisd WHERE nsid = "+nsid+" ORDER BY nid ASC"
13      result = mysqli_query(db, query)
14      WHILE row = mysqli_fetch_array(result)
15          overallDNA = overallComponents.read("DNA")+row[2]
16          overallMods = overallComponents.read("histoneModifications")+row[1]
```

```
17          errorCode = overallComponents.edit("DNA", overallDNA)
18          errorCode = overallComponents.edit("histoneModifications", overallMods)
19          mods = explode(",", row[1])
20          modsNames = ""
21          FOREACH mods AS mod
22              modQuery = "SELECT name FROM histoneMods WHERE hmid = "+mod
23              name = mysqli_fetch_array(mysqli_query(db, modQuery))[0]
24              modsNames += name+", "
25          END FOREACH
26          html = generateHTML(modsNames, row[2])
27          nucleosomeQueue.add(html)
28      END WHILE
29
30      allModsArray = explode(",", overallComponents.read("histoneModifications"))
31      expressionChange = findResult(allModsArray)
32      topHTML = generateHTML(expressionChange, overallComponents.read("allDNA"))
33      OUTPUT(topHTML)
34      DO
35          OUTPUT(nucleosomeQueue.read())
36          nucleosomeQueue.pop()
37      WHILE nucleosomeQueue.isEmpty() == FALSE
38  ENDIF
```

One input is needed in order to display data on a nucleosomes sequence. This data is the *nsid* value. This is the primary key for the *nucleosomeSequence* table. It is submitted to a page via the GET protocol. The value is therefore retrieved using the `$_GET` superglobal. Line 1 checks to see if this value has been submitted. if it has then the rest of the code can execute. Line 2 retrieves this value and stores it in the `nsid` variable so that it can be quickly referenced later in the code.

Line 3 instantiates a new *dictionary* object. This object is then stored in the variable `overallComponents`. This dictionary holds data on the entire DNA sequence that the nucleosomes sequence contains as well as all of the histone modifications in the sequence. The keys for these values are `"DNA"` and `"histoneModifications"` respectively. They are stored in a dictionary so that the two pieces of data can be stored in the same data structure and therefore, accessed as one object rather than multiple. This declutters the code and therefore makes maintenance easier. The array passed as an argument into the constructor of the *dictionary* class is used to initialise the data structure. This was explained more thoroughly in the **Dictionary** section of this document.

A **HTML block** is a block within the display that shows the user a particular piece of information, such as a single nucleosome or overview data.

Line 5 finds the number of HTML blocks that need to be displayed by finding the number of nucleosomes that need to be displayed and adds a value of 21to the resulting integer. This value accounts for one block that contains overview data on the nucleosome sequence, such as the full DNA sequence or the disease that it is associated with. The function `mysqli_num_rows()` finds the number of nucleosomes in the sequence from the result of the following query:

```
SELECT nid FROM nucleosomes WHERE nsid = 5
```

This query is used to retrieve data. Therefore a `SELECT` query is used. The data retrieved is used to validate that a row exists so that it can be counted. Therefore the attribute with the shortest datatype should be retrieved. This is why the `nid` attribute is retrieved.

> The nucleosome records that are retrieved should only be related to a particular nucleosome sequence. This is why a `WHERE` clause is used to specify this using the `nsid` key. In this case the records retrieved have to be associated with a nucleosome sequence with a `nsid` value of 5.

This number is stored in the `num` variable. `num` is then used to find instantiated a new queue object. This object will contain the HTML block of each nucleosome, so that they can be displayed later. The `num` variable is used to set the fixed length of the queue. so that all HTML blocks can be contained in the queue.

On line 7 a query is generated which retrieves overview data on the nucleosome sequence. This query was explained earlier in the document. Line 8 executes this query. Line 9 enables access to the retrieved data by storing it as an array using `mysqli_fetch_array()`. Line 10 generates HTML that contains some of this overview data and stores it in the `topHTML` variable. This variable will contain data that needs to be displayed at the top of the page. This isn't displayed yet because not all of the overview data (i.e. the change in expression or the entire DNA sequence) is retrieved yet.

Line 12 generates another query. This query was explained earlier in this document. It retrieves the *histoneMods* list and DNA sequence for each nucleosome in the sequence. Line 13 executes this query.

Line 14 loops through each nucleosome using a while loop. This works by storing one record's data as an array in `row` and using this variable throughout the current iteration. At the end of this iteration, `row` is set to store the next record's data and so the cycle continues. This is what happens within the loop:

- On line 15 the DNA sequence of the nucleosome is concatenated to the entire DNA sequence. The result is then stored in the `overallDNA` variable. The same happens with the nucleosome's *histoneMods* value. The result here is stored in the `overallMods` variable. The entire DNA sequence and the list of all histone modifications in the sequence are retrieved from the `overallComponents` dictionary.
- The next two lines update the values of the "allDNA" and "allMods" keys in the `overallComponents` dictionary with the values of `overallDNA` and `overallMods` respectively. Therefore these new values have the current nucleosome's DNA and histone mods list added to them. Therefore, by the end of the loop the DNA sequence and histone modification list for the entire sequence have been retrieved.
- Line 19 creates an array of the *hmid*s (*histoneMods* primary keys) that this nucleosome contains using the `explode()` function. This array is then stored in the `mods` variable. Line 20 initialises the `modsNames` variable so that it has the appropriate scope to be modified inside a loop whilst being able to be used outside of the loop.
- Lines 21 to 25 define a FOREACH loop. This loop sets an element of the `mods` array to the `mod` variable. After iteration using this element, the next element in the array is stored in the `mod` variable. This continues until each element in the array has been iterated through the loop once. Within this loop:
    - On line 22 a query is generated and stored in `modQuery`. This query is used to retrieve the name of the histone modification with a *hmid* value of `mod`. It is explained below:

      ```
      SELECT name FROM histoneMods WHERE hmid = 3
      ```

      In order to retrieve data from a table a `SELECT` query must be used. In order to specify which record should have its data retrieved from a `WHERE` clause is used. This clause specifies this record using the primary key of the *histoneMods* table (*hmid*).
    - On line 23 the query is executed and the *name* of the histone modification at hand is accessed using `mysqli_fetch_array()`.

    - Line 24 concatenates the name of this histone modification to the end of the `modNames` variable. It is followed by a `", "` string which is used to separate each histone modification name from each other so that the user can clearly differentiate two names from one another.
- Line 26 generates a HTML block for the current nucleosome using the names of its histone modifications and its DNA sequence. This HTML is then added to the back of the `nucleosomeQueue` queue.

On line 30 an array is created that stores the *hmid*s of the histone modifications used across the entire nucleosome sequence using the `explode()` function. This array is stored in the `allModsArray` variable. The change in expression is found using the `findResult()` function that was defined earlier in this document on line 31. It uses the `allModsArray`.

On line 32, the rest of the HTML block that contains the overview data is generated and concatenated to `topHTML`. This is done because the DNA sequence contained within the entire nucleosome sequence and the change in expression has been retrieved.

On line 33 the contents of the `topHTML` variable are outputted. This means that the overview data is displayed at the top of the page before the nucleosome-by-nucleosome break down.

On lines 34 to 37 a do-while loop is used to output each element stored in `nucleosomeQueue` before removing that element. This therefore displays each nucleosome in the order that they were added - from first to last. Therefore displaying the sequence in the same order that it was submitted. The way this do-while loop works with a similar context was explained in the **Thread Displaying Algorithm** section of this document.

Disease Displaying Algorithm

This algorithm is used to display overall information on a disease and its associated nucleosomes sequences, as well as hyperlinks that redirect to pages that contain further details on those nucleosome sequences.

```
1       IF isset($_GET["disease"])
2          did = $_GET["disease"]
3          query = "SELECT name, notes, uid FROM diseases WHERE did = "+did
4          result = mysqli_query(db, query)
5          row = mysqli_fetch_array(result)
6          html = generateHTML(row)
7
8          query = "SELECT name, notes, nsid  FROM nuceosomeSequences WHERE did = "+did
9          result = mysqli_query(db, query)
10         WHILE row = mysqli_fetch_row(result)
11               html += generateHTML(row)
12         END WHILE
13
14         OUTPUT(html)
15  ENDIF
```

The primary key (*did*) of the disease that the user is trying to retrive data on is stored in the `$_GET` superglobal. Specifically, `$_GET["disease"]`. Therefore, we can tell if the user is attempting to display a disease if `$_GET["disease"]` has been set. This is checked in an IF statement on line 1. If it has been set, the rest of the code can execute.

Line 2 retrieves 'did' and stores it in teh variable `did`.

Line 3 generates a query that retrieves overview data on the disease from the *diseases* table. This equery was expalined earlier in this document. Line 4 executes this query and stores the result in `result`. Line 5 makes this data accessible by storing it as an array using `mysqli_fetch_array()`. Line 6 generates HTML using this array. This HTML is then stored in `html`.

Line 8 generates a query which retrieves overview data on each nucleosome sequence which is associated with this disease. This query was explained earlier in this document. Line 9 eecutes this query and the result is returned to the `result` variable.

Line 10 uses a while loop to iterate through each retrieved row. This is done by assigning `row` an array of data for a single row and iterating through the loop using this data before setting `row` to contain data from the next record that was retrived.

Within this while loop HTML is generated using the data strored in row. This is then concatenated to the `html` variable.

Once the while loop has completed, all of the associated nucleosome sequences will have their HTML concatenated to the `html` variable. The disease overview data is stored at the beginning of this `html` string so that it can be displayed first.

Line 14 outputs the entire `html` string to display the HTML mark up.

Components For Both Solutions

Database Tables

Users Table

Name	Type	Purpose
uid	UNSIGNED INT AUTO_INCRMENT PRIMARY_KEY	%
firstName	VARCHAR 50	This contains the first name of the individual who owns this account.
secondName	VARCHAR 100	This contains the second name of the individual who owns this account.
emailAddress	VARCHAR 150	This contains a unique email address that the individual owns. This will be used to confirm the account's validity via email to prevent bots from spamming the database.
password	VARCHAR 255	This contains a password that has been encrypted (cannot be read by humans) using a hashing algorithm. This prevents malicious users who can see the database from extracting passwords. Once encrypted the password cannot be decrypted, as hashing works only in one direction, making the password more secure. In order to check if an inputted password/emailAddress combination is correct, the inputted password will be hashed using the same algorithm and compared with he stored one.
hash	CHAR 32	Contains a hashed password value using a different hashing algorithm to enhance security. This is set to NULL once the account has been activated.
level	BIT	A value of 0 represents a standard account, 1 represents an admin account.
interest	VARCHAR 50	This contains a string value that holds basic information as to why the user registered an account. this could be sold to large scientific websites.

This table is used to contain user accounts as records.

The *firstName* and *seconName* attributes uses a VARCHAR data type. I chose this because an user's real-world name can have a varying length. VARCHAR only uses as much storage as neccessary upto a maximum. This means that storage for each *firstName* or 8secondName* is always efficient. *secondName* has a larger maximum character length than *firstName* because second names are typically longer than first names.

An email address is a string made up of different chracters. Therefore the *emailAddress* attribute has a VARCHAR data type. This is beneficial for the same reasons as the previous attribute. It has a maximum length of 200 characters. This is because email addresses can be very long, so this is reflected in the maximum capcity of thsi attribute.

Before being stored in the *password* attribute, the password that was inputted is ran through the `crypt(x, y)` algorithm. This is a hashing algorithm in PHP, where `x` is the string that is to be hashed and `y` is the **salt**. The salt is a string that is used to hash `x`. Because PHP's documentation states that the development team is continually changing `crypt()` to be more secure, the length of the returned hash value may vary. They suggest allowing for a maximum of 255 characters to store the output of `crypt()`. This is why the maximum length of the *password* attribute is 255 characters long. Because the returned hash is a mixture of characters, the data type for this attribute is VARCHAR. This carries the same benefits as explained previously. Because `crypt()` is a hashing function it is more secure as the the hash value cannot be reversed using the same algrithm to return the previous password.

The *hash* attribute contains a value that was outputted by my **Hashing Algorithm**. This attribute is used to determine if an account is activated or not. Because this algorithm returns a string value made up of different characters and because it always returns a string of a fixed length of 32 characters, I chose to use the data type CHAR because it is quicker to retrive data from it. I chose a fixed length of 32 characters to keep data storage efficient.

The *level* attribute can have one of two values: 0 or 1. 0 represents a standard account whereas 1 represents an admin account. Therefore, I chose to use a BIT datatype. This is because a bit can also only have two values. There is only a single bit in this datatype so it automatically has a length of 1.

The *interest* attribute stores the reason why this account was created. This reason is described in english and therefore uses characters. This means the data is a string. Therefore, I chose to use the VARCHAR attribute. This has the same benefits as described earlier. I chose a maximum length of 50 characters because the data should be short as it only contains a breif reason as to why the account was created.

Hashing Algorithm

This algorithm is used to generate a hash from a given string. In the context of my solutions, it will create a hash from a new account's *password* and stored it in the *hash* attribute. This attribute is used to determine if an account is activated or not, where if it is NULL an account is activated. If it has a value then it is not activated. This will be explained further at the *Account Activation Algorithms* section. The returned hash must be 32 characters long so that it does overflow the hyperlink that is generated by one of the **Account Activation Algorithms**.

I have described this hashing algorithm using pseudocode as it uses mathimatics which is easieer to explain programatically. Some PHP-specific functions are used. Most of these functions have been defined earlier in the document. Those that havent are defined in the following table: |Function Name|Purpose |-|- |ord(*x*)|This function finds the ASCII decimal value of the character *x*. |dechex(*x*)|This function converts a decimal value into a hexidecimal value. This hexidecmial value is then returned.

```
1   FUNCTION hashing32(starterStr)
2       chars = str_split(starterStr, 1)
3       hash = ""
4       FOREACH chars AS char
5           ascii = ord(char)
6           ascii *= 817513877
7           hex = dechex(ascii)
8           hash += hex
9       END FOREACH
10
11      IF length(hash) > 32
12          hash = str_split(hash, 32)[0]
```

```
13      ELSE IF length(hash) < 32
14          i = 0
15          DO
16              hash += "1"
17              i++
18          WHILE i < (32-length(hash))
19      ENDIF
20      RETURN hash
21  END FUNTION
```

This algorithm is defined as a function. This is because it returns a value and so that the complexity of the **Account Activation Algorithms** is reduced. Therefore, making the algorithm easier to read and therefore easier to maintain. The function takes one paremeter called `starterStr`. This contains the string that needs to be hashed.

Line 2 breaks `starterStr` into an array of single characters using `str_split()`. This array is then stored in the `chars` variable.

Line 3 initilises the `hash` variable. This is so that it has a large variable scope so that it can be modified within loops and if statements whilst still being returned at the end of the function.

Lines 4 to 9 defines a FOREACH loop. This takes an element from the `chars` array and stores it in the `char` variable. This variable is then used through one iteration. After this iteration the next element is assigned to `char`. The cycle repeats until each element has been iterated once. Within this loop:

- Line 5 finds the ASCII decimal value and assigns it to the `ascii` variable.
- Line 6 multiplies `ascii` by a large prime number. This is so that if a malicous user attempts to reverse engineer the algorithm they have to find the prime number that was - this takes a lot of time and a lot of computation power to do (computational security).
- Line 7 converts the decimal value to hexidecimal using `dechex()`. It is then stored in the `hex` variable. `hex` is therefore a string as hexideimal uses non-denary characters to represent numbers.
- Line 8 concatenates the `hex` variable to the `hash` variable.

After this loop a long string of hexidecimal values will be strored in `hash` based on each character.

Line 11 determines if the length of the `hex` value is longer than the 32 characters that we need.

- If it **is** then the first 32 characters of `hash` is retrived. This is done by using the `str_split` function to create an array of 32 character long chunks of data. This function is then reated as an array to return the first chunk.
- If it **is not** then the length of `hash` is checked to see if it is shorter than 32 characters. If it is then the difference between 32 and the length of `hash` is used to find the number of characters needed to be added. `i` is used as a acounter to count from 0 to the number of characters needed. These are then used in a do-while loop to append a character until `i` is no longer less than the difference between 32 and the length of `hash`. This will elongate `hash` to have 32 characters.

On line 20, `hash` is returned.

Account Activation Algorithms

Generating An Activation URL

This algorithm is used to create a URL that redirects the user to a page that activates their account. An account is activated when its *hash* value is `NULL`. When an account is created this *hash* value is filled with a hash value.

A new php-specific function is used. It is outlined in the table below:

Name	Purpose
mail(*x*, *y*, *z*)	This function sends an email to the email addres *x* with the subject *y* with the message *z*.

```
1   FUNCTION urlGenerator(email, hash)
2       url = "index.php?hash="+hash+"&emailAddress="+email
3       mail(email, "Nomios - Account Activiation", url)
4   ENDIF
```

I decided to define this algorithm as a function because it will be called later by the **Sign Up Algorithm**, which decreases that algorithm's code's complexity. This therefore makes the code easier to maintain. This function has two parameters: `email` and `hash`.

`email` contains the account's email address (this is stored in the *users* table's attribute *emailAddress*).

The `hash` parameter contains a hash value generated by the `hashing32()` algorithm that I defined earlier in this document. This algorithm was defined earlier. It will take the unencrypted password that was inputted by the user when it was signed up and return a hash value.

Line 2 generates the URL. I will now explain the different components of this URL:

- `index.php` is the file that is being navigated to.
- `?` indicates the beginning of the URL query. This query can hold data as key/vale pairs. This is in the form of *a*=*b*. In this case both keys are assigned using concatenation:
    - `hash="+hash` sets the "hash" key is assigned the hash value that is stored in the `hash` variable.
    - `&emailAddress="+email` sets the "emailAddress" key to the account's email address, which is stored in the `email` variable. Note that `&` is used to differentiate between different key/value pairs.

Line 3 sends the email via SMTP protocol to the inbox of the email address stored in `email`. The email sent contains the generated URL as the body of the email. This URL can be entered by the user into their web browser so that they can be directed to a web page that activates the account.

Activating An Account

This algorithm is used when a user has been redirected to a web page that activates this account. This is redirection is caused by entering the URL that they were emailed during account creation (more specifically, the `urlGenerator()` function called in the **Sign Up Algorithm**). I will explain it using pseudocode. This is because the algorithm uses many different functions which would make a flowchart incredibly confusing.

```
1   IF isset($_GET["hash"]) && isset($_GET["emailAddress"])
2       hash = $_GET["hash"]
3       email = $_GET["emailAddress"]
4       query = "UPDATE users SET hash=NULL WHERE hash='"+hash+"' AND emailAddress='"+email+"'"
5       result = mysqli_query(db, query)
6       IF result == TRUE
7           OUTPUT("Account activated.")
8       ENDIF
9   ENDIF
```

A particular account from the *users* table can be identified using the *emailAddress* attribute. This is because each email address stored in this column is unique. The *hash* attribute may not be unique as its value is found using a hashing algorithm, which may return the same hash value for two separate strings. However, the hash value is still needed so that URL that is used to activate the account can be verified (i.e. it is authentic and is from the inbox that it was sent to). Otherwise any user could just enter an email address and activate the account!

Before this algorithm can be used both the "hash" and "emailAddress" keys should be present in the URL. If they are in the URL then they have been submitted to the server using the GET protocol. Therefore in order to check if the two keys are in the URL an IF statement is used that does this using the `isset()` function. If they are present, the rest of the code can execute.

Lines 2 and 3 assign the values associated with the keys to the `hash` and `email` variables. They are then used on line 4 to generate a SQL query that changes the *hash* attribute to be NULL for this particular record. This is explained further below:

> ```
> UPDATE users SET hash=NULL WHERE hash='081f80af686cbeac34a556276e0a7d71' AND
> emailAddress="reiss@jones.com"
> ```
>
> This query is intended to edit data for a particular record in the *users* table. Therefore an `UPDATE` query is used on the `users` table. The `SET` clause lists the attributes that are to have their values changed, followed by the value that is to be changed to. In this case it is the *hash* attribute being changed to contain NULL.
>
> In order to specify a particular account, a record should be specified. Because the *emailAddress* attribute contains a unique value for each record it is used in the `WHERE` clause to specify the row. In order to make sure the hash value presented in the URL is authentic it is also specified so that the only record changed is one that contains a matching hash value. The `AND` keyword is used to ensure that both conditions return true before any record is updated.

This query is then executed on line 5 and the result is stored in `result`. If any records were changed, `result` will be true. Otherwise, it will be false. A success message should only be outputted if `result` is true. Tis is what lines 6 to 8 do.

Sign Up Algorithm

This algorithm is used to create a new user account. I have written it in pseudocode as it is quite complex and would be difficult to understand if it was presented as a flowchart. Some new php-specific functions are used. These are defined in the table below:

Function Name	Purpose
array_map(*x*, *y*)	This returns an array that contains elements that have each been processed through function *x*. The previous versions of theses elements were stored in array *y*. Instea do being called *x* is passed as a string
trim(*x*)	This returns a string with no excess whitespace. The original string is *x*.

```
1   IF isset($_POST["submitted"])
2       $_POST = array_map("trim", $_POST)
3       $_POST = array_map([db, "real_escape_string"], $_POST)
4       firstName = strip_tags($_POST["firstname"])
5       secondName = strip_tags($_POST["secondName"])
6       emailAddress = strip_tags($_POST["emailAddress"])
7       password = $_POST["password"]
8       intrest = strip_tags($_POST["intrest"])
9       errors = []
10
11      errors = errorChecking()
12
13      IF empty(errors)
14          hash = hashing32(password)
15          encryptedPassword = crypt(password, "example")
17          query = "INSERT INTO users VALUES(NULL, '"+firstName+"', '"+secondName+"',
'"+emailAddress+"', '"+encryptedPassword+"', '"+hash+"', 0, '"+intrest+"')"
18          result = mysqli_query(db, query)
19          urlGenerator(emailAddress, hash)
20          OUTPUT("Success. Please activate your account by checking your inbox.")
21      ELSE
22          FOREACH errors AS error
23              OUTPUT(error)
24          END FOREACH
25      ENDIF
26  ENDIF
```

As I have previously explained any submission algorithm should check if its form has been submitted before retrieving data (to prevent errors). If the form has been submitted then the rest of the code can be executed.

Line 2 applies the `trim()` function on each piece of data submitted via POST. This means that all data submitted through the form has any excess whitespace removed. This means that data is inserted into the database correctly. Line 3 does the same but using the `mysqli_real_escape_string()` function. It is passed as an array because a data connection is needed when used the function is used in this context. This is done to properly escape any special characters in a string to prevent an error when the later query is executed.

Lines 4 to 8 retrieved data that has now been passed through these functions. Each data is passed through the `strip_tags()` function besides the data for the *password* attribute. This is to prevent malicious client-side code from being submitted. The *password* does not require this as it will be encrypted later and the characters that make up the malicious code will be encrypted anyway. This gives the user more flexibility

when inputting their password.

Line 11 is meant to simplify the pseudocode that checks for invalid data. This pseudocode has been repeated numerous times through this document. This is not a function! It is merely used to represent code that checks for errors.

Line 13 checks if the `errors` variable is empty:

- If it **is** empty, a query that inserts a new record into the *users* table is generated. This is explained further below:

  ```
  INSERT INTO users VALUES(NULL, 'Reiss', 'Jones', 'reiss@jones.com',
  'JKOMEFSI9UY9FUES98EF98EHS9UFH', '081f80af686cbeac34a556276e0a7d71', 0, 'Science!')
  ```

  > As a new record is being added to the *users* table an `INSERT` query is used. Eahc value matches the datatype for each attribute in the *users* table. The value for *uid* is `NULL`. this is because its value is based on the AUTO_INCREMENT feature.

  This query is generated using the concatenation of the variables that hold the data that was retrieved earlier. The query is then executed.

  On line 19, the URL which activates the account is generated and sent to the user via email.

  Line 20 outputs a message which lets the user know that the account creation was successful and that they should activate their account by using the URL that was delivered to their inbox.

- If it **is not** empty each error is outputted using a FOREACH loop. The way this works was explained earlier in the document.

Sign In Algorithm

This algorithm is used to sign in a user. It uses many programming constructs. Therefore I have decided to explain this algorithm using pseudocode as a flowchart would be very complicated to read if it was used instead.

```
1   IF isset($_POST["submitted"])
2       emailAddress = trim($_POST["emailAddress"])
3       password = trim($_POST["password"])
4       errors = []
5
6       IF emailAddress == "" OR emailAddress == "rf@example.com"
7           array_push(errors, "Invalid email address.")
8       ENDIF
9
10      IF password == "" OR password == "Password
11          array_push(errors, "Invalid password.")
12      ENDIF
13
14      IF empty(errors)
15          password = crypt(password, "example")
16          query = "SELECT uid, level FROM users WHERE emailAddress='"+emailAddress+"' AND
password = '"+password+"'"
17          result = mysqli_query(db, query)
18          IF result == TRUE
19              row = mysqli_fetch_array(result)
```

```
20              $_COOKIE["user"] = row[0]
21              global LEVEL = row[1]
22              OUTPUT("Signed in")
23          ELSE
24              OUTPUT("Incorrect combination")
25          ENDIF
26      ELSE
27          FOREACH errors AS error
28              OUITPUT(error)
29          END FOREACH
30      ENDIF
31  ENDIF
```

Line one checks if the sign in HTML form has been submitted or not. The way this works has been explained previously in this document. If it has then the rest of the code can be executed.

Lines 2 and 3 retrieve and apply the `trim()` function to the inputted `"emailAddress"` and `"password"`. Line 4 initilises the `errors` variable so that it has an appropriate scope.

Lines 5 to 13 perform error checking on the inputted data. If an errors are caught, a string describing the error is appended to the `errors` array.

Line 14 checks if the `errors` array is empty or not:

- If it **is** empty then there are no errors. The inputted password is encrypted using the **same** `crypt()` function with the same salt and reassigned to `password`. Because `crypt()` is a hashing algorithm, the returned hash value will be the same as the stored *password*. A query is generated which retrieves the *uid* and *level* of the account which is being singed in with. This query is explained below:

  ```
  SELECT uid, level FROM users WHERE emailAddress = "reiss@jones.com" AND password =
  "FEIU87MCEI893";
  ```

  > Because this query is retrieving data `SELECT` query is used. In order to retrieve the *uid* and *level* of the account, these attributes are specified. The `WHERE` clause is used to specify which record is retrieved using the *emailAddress* and *password* attributes. If the specified values match a row's attribute's values then the user can sign in as this account.

  The query is then executed and the result is stored in `result`. If `result` contains any returned records then it will have a boolean value of true. Otherwise it will be false. An IF statement is used to determine which it is.

  - If it is **true** the email/password combination matches an account's. Line 19 makes this retrieved data accessible using `mysqli_fetch_array()`. The *uid* should be stored as a cookie so that it can be accessed by the webpage to allow the signed in user access to the forum and the HMI. The level should be stored in a global constant variable so that it can be accessed too in case the user is an admin and needs access to admin functions (such as deleting inappropriate posts). These occur on lines 20 and 21 respectively. The `$_COOKIE` superglobal array is used to create a cookie and assign it a value. The `global` keyword gives the following variable a global variable scope. A message is then outputted indicating to the user that they have signed in successfully.
  - If it is **false** the email.password combination did not match an account's. A message indicating this is outputted to the user.

- If it is **not empty** there are some errors. These are outputted to the user in a FOREACH loop as explained previously in this document.

Searching SQL query

This SQL query will be used to search the *nucleosomeSequences* table for records that have a *name* value that matches what the user has inputted. It can be modified to search for records from the *diseases* table too (as both tables use the *name* attribute with the same data type and purpose). It may also be modified to search for records from the *threads* table. So think of this query as more of an example of a general query that can be applied to search fro all types of data.

```
SELECT nsid, name, notes FROM nucleosomeSequences WHERE name LIKE '%APP%'
```

As we are retrieving data a `SELECT` query is used. The `nsid` attribute is retrieved so that a hyperlink can be generated that redirects the user to a page that displays the sequence by itself (the `nsid` is needed so that this page knows which sequence to display). The `name` and `notes` are retrieved so that they can be displayed within that record's HTML block (this is so that the user knows what each nucleosome sequence is about in an overall sense).

The `WHERE` clause is used to specify the records that are retrieved from the table. The `LIKE` keyword is used to compare the *name* attribute with the specified string. `LIKE` lets a record be returned if the *name* attribute matches any capital variant of the specified string. E.g:

- app
- ApP
- APP

The wildcard character `%` is used to match any set of characters of any length. In this case:

- APP Expression Increase
- Expression For APP Increases

Combining both of these features causes for more varied results to be returned. This means that the results that are returned are less strict, which means that the user is more likely to find the nucleosome sequence they are looking for.

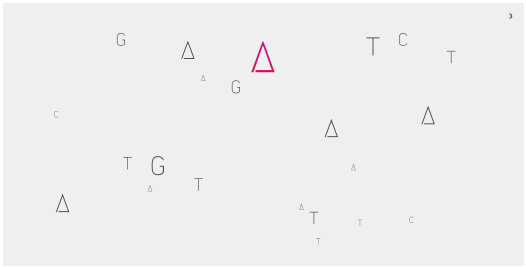
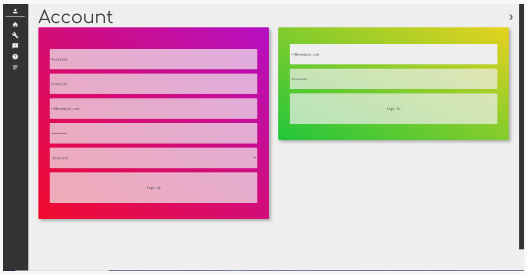
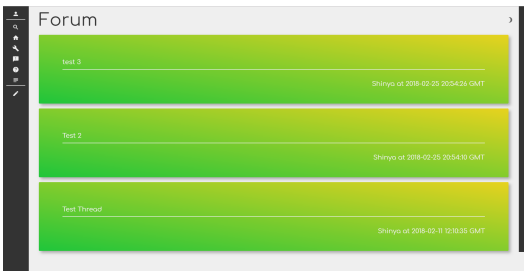
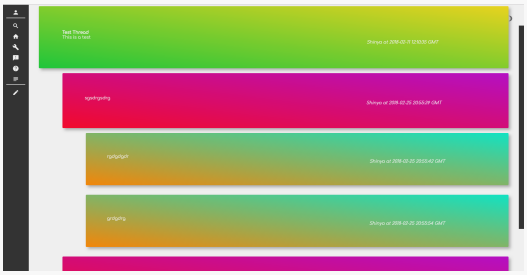
Modifications that change the function of this query could be:

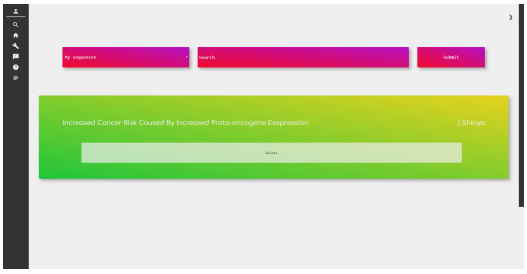
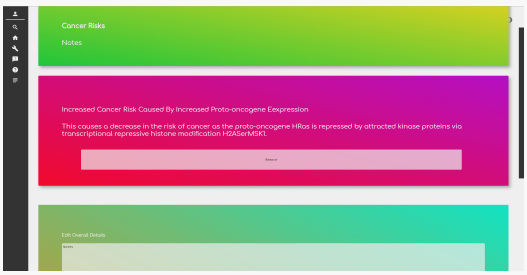
- `did` instead of `nsid` and the table that data is retrieved from could be set to `diseases`. This would search for *diseases* records that have a *name* similar to the string specified.
- `tid`, `subject` replacing the attributes. `subject` replacing `name` in the `WHERE` clause. The table that is having data retrieved from it set to `threads`. This lets the user search for threads that have a subject that is similar to the specified string.
- A `AND uid =` portion added to the `WHERE` clause to search for data that belongs to the current user.

User Interface

In order to explain the UI for each component of my solutions I have used a table that shows a screenshot of the UI I designed on screen using CSS along with a brief explanation of why that specific part of the UI is good. I chose to design my UI in CSS so that I could use the code directly in my project and modify it if appropriate. Note that as my entire project uses a "single webpage" (as outlined in my objectives) the word "page" here means HTML element that contains a particular piece of information or HTML form that also

take sup most of the viewport. The is colourful. This maintains a fresh look and feel for the website, which make sit more enjoyable to use.

Name	Screenshot	Explanation
First-load Entry Page		This is a landing page for first-time users. It is meant to offer a "wow-factor" that intrigues the user to interact with the page and enter the website proper. This is done by clicking on the red triangle.
Account Sign In/Sign Up Page		This is a page that displays HTML forms for signing in and sign up for a new account. The inputs are highlighted when the cursor hover's over it to make the UI more responsive.
Account Sign In/Sign Up Page Night Mode		This showcases Night mode. This is where the colours used in the UI are much darker, which dulls the light of the screen dramatically. This makes the website suitable to use in the dark and prevents eye strain.
Account Details/Sign Out Page		This contains a form to change to user's password (incase their account has been hacked) and a button which allows them to sign out . This button is red to indicate a warning to the user that once sign out they must sign back in again.
Main Forum Page		This contains a list (ordered from latest to earliest) of threads that are in the forum. Each thread can be clicked on to redirect to the **Forum Single View Page** for that particular thread. The transition between pages is seamless creating a better user experience.
Forum Single View Page		This page contains a list of HTML blocks, each displaying a single thread within a post. Each type of post has a different colour and different indentation so that the user can differentiate between posts.

Name	Screenshot	Explanation
HTML Form Forum Example		This form is an example of creating a new thread or posting a response to another a thread or message. The background is dimmed so that the user can focus on the form only, letting them focus on writing.
Disease Creation HTML Form		The part of the form with the pink background is part of the disease creating form. The inputs are very large, letting the user focus on writing and avoiding typing errors.
Sequence Creation Form		This is part of the form for creating a new nucleosome sequence. Each input is very wide and is highlighted when in use. This lets the user focus on the data they are inputting. Decreasing the amount of mistakes in the data.
Sequence Display		This is an example of a nucleosome sequence being displayed. The overview data has a green background, whereas any data about a nucleosome has a pink one. This differentiates the two. Nucleosomes are differentiated by having different HTML blocks (gaps are between each block).
Search and Search Result		The UI for the search form is simple - it is all on a single line and each input is of the same colour. This reduces confusion for the user.
Disease Display		This is an example of a disease being displayed with its associated sequences. Each sequence is displayed with a pink background and when clicked on redirects the user to a sequence display page (seamless experience).

Name	Screenshot	Explanation
Editing Form Example		This is an example of an editing form for the disease and sequence display pages. It is in a different colour to other elements on these pages so that it can be clearly recognised to prevent accidental edits.

Nomios -Technical Solution

This document contains the code that makes up my solution. The code is explained using the comments that are within the code for each file. Please remember that each reference to "page" is to a HTML element that takes up most of the screen and contains a specific category of information. It is not a webpage!

For a more clear read of the code, the files themselves should be opened.

My solution can be split into different pages:

- The **account page** is the page for dealing with user accounts.
- The **forum page** is the page for viewing all threads or creating a new one
- The **forum single view page** is the page for viewing a single thread.
- The **tool page** is the page for creating a new disease or nucleosome sequence.
- The **tool single view page** is the page for viewing a single nucleosome sequence or a single disease.
- The **search page** is the page for searching threads, diseases or nucleosome sequences. The results can be further filtered to those created by the user or to all records.
- The **about page** is where an explanation of what the website is about and the help guide are located.
- The **entry page** is the landing page for new users.
- The **home page** is where the user is directed to after they pass the **onload page**.

The project has been modularized into different files depending on the page their code is relating to and when that code is executed. This makes maintenace iof the code easier, as it is less complicated to read.

index.php

This is the webpage that the user accesses. All other resources are loaded onto this page, potentially changing it to suit the circumstances of the user. This is necessary because my website uses a single web page as explain in my *Analysis*.

```
<html>
  <?php
    #error_reporting(0); # Turn off error reporting.
    $db = mysqli_connect("localhost", "root", "", "hmi"); # Connect to the MySQL database. All
queries use this conenction to communicate to the database.
    $admin = FALSE; # Default this variable to be false for when the user is not signed in.
    if(isset($_COOKIE["user"])){ # If the user has signed in.
      $uid = $_COOKIE["user"]; # Extract the ID data from the cookie and store it as a variable.
      $userRow = mysqli_fetch_array(mysqli_query($db, "SELECT * FROM users WHERE uid=".$uid)); #
Select all data associated witht his particular user and store it as an array.
      if($userRow[7] == "1"){ # This fetches the user level. If it is set to a value of one the
the user must be an admin.
        $admin = TRUE;
      }
      global $uid, $userRow;
    }
    else { # If the user is not signed in we should change the UI so that the user knows that
they cannot enter the forum or tool pages.

      echo('<style>#forumBtn, #toolBtn {
```

```
        cursor: not-allowed;
      }</style>');
    }
  ?>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Nomios</title>
    <link href="https://fonts.googleapis.com/css?family=Megrim|Comfortaa" rel="stylesheet">
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
    <link rel="stylesheet" href="css/animations.css">
    <link rel="stylesheet" href="css/general.css">
    <link rel="stylesheet" href="css/inputStyles.css">
    <link rel="stylesheet" href="css/mediaQueries.css">
    <link rel="stylesheet" href="css/pages.css">
    <link rel="stylesheet" href="css/accountStyles.css">
    <link rel="stylesheet" href="css/popupStyles.css">
    <link rel="stylesheet" href="css/forumStyles.css">
    <link rel="stylesheet" href="css/toolStyles.css">
    <link rel="stylesheet" href="css/homeStyles.css">
    <link rel="stylesheet" href="css/entryStyles.css">
    <link rel="stylesheet" href="css/searchStyles.css">
    <!-- The above imports all of the css resources neccessary for the page. The first two links
link to external resources, the first to retrive the fonts used and the second t retrive the
icons packs that is needed for the UI. -->
    <?php
      if(isset($_COOKIE["accessedBefore"])){ # Disables the first-load entry screen before
init() is triggered so that it does not display.
        echo('<style> #bgObjCon, #enterBtnCon {
          display: none;
        }
        #mainCon {
          animation-duration: 1s;
        }</style>');
      }
    ?>
  </head>
  <body>
    <script src="https://www.w3schools.com/lib/w3.js"></script>
    <script src="js/entryScript.js"></script>
    <script src="js/menuScript.js"></script>
    <script src="js/inputScript.js"></script>
    <script src="js/toolScript.js"></script>
    <script src="js/accountScript.js"></script>
    <!-- Above are all of the JavaScript scripts neccessary for the site to function. -->
    <div id="dayChangeBtn" onclick="dayChangeClick();"><i class="material-
icons">brightness_3</i></div> <!-- The button that switches between day and night mode. -->
    <div id="enterBtnCon"><h1 id="enterBtn" class="coloredText redPurple">A</h1></div> <!--
First-load button to enter the website. -->
    <div id="bgObjCon"> <!-- Contains all of the elements that make up the 3D-like effect of the
first-load page. -->
      <div name="bgObj" class="bgObjBig" style="position: absolute; top: 69%; left: 10%;
animation-fill-mode: backwards;">A</div>

      <div name="bgObj" class="bgObjBig" style="position: absolute; top: 11%; left:
```

```
34%;">A</div>
      <div name="bgObj" class="bgObjBig" style="position: absolute; top: 43%; left: 62%;
animation-name: floatTwo;">A</div>
      <div name="bgObj" class="bgObjBig" style="position: absolute; top: 55%; left: 28%;
animation-fill-mode: backwards;">G</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 29%; left: 44%;
animation-name: floatTwo;">G</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 11%; left: 22%;
animation-name: floatTwo; animation-fill-mode: backwards;">G</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 66%; left: 37%;
animation-name: floatTwo;">T</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 77%; left: 59%;
animation-name: floatThree;">T</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 58%; left: 23%;
animation-name: floatFour; animation-fill-mode: backwards;">T</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 82%; left: 68%;
animation-name: floatFour;">T</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 68%; left: 28%;
animation-name: floatThree;">A</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 27%; left: 38%;
animation-name: floatFour; animation-fill-mode: backwards;">A</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 60%; left: 67%;
animation-name: floatThree;">A</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 89%; left: 60%;
animation-name: floatFour;">T</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 75%; left: 57%;
animation-name: floatThree; animation-fill-mode: backwards;">A</div>
      <div name="bgObj" class="bgObjBig" style="position: absolute; top: 12%; left: 70%;
animation-name: floatTwo;">T</div>
      <div name="bgObj" class="bgObjBig" style="position: absolute; top: 38%; left: 80%;
animation-name: floatFour;">A</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 11%; left: 76%;
animation-name: floatTwo; animation-fill-mode: backwards;">C</div>
      <div name="bgObj" class="bgObjMed" style="position: absolute; top: 16%; left:
85%;">T</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 80%; left: 78%;
animation-name: floatThree; animation-fill-mode: backwards;">C</div>
      <div name="bgObj" class="bgObjSmall" style="position: absolute; top: 40%; left: 10%;
animation-name: floatThree; animation-fill-mode: backwards;">C</div>
    </div>
    <div id="mainCon">
      <div id="menuCon"> <!-- Contains all of the links to different "pages" within the site.
menuBtnClick() is defined in js/entryScript.php -->
        <p onclick="menuBtnClick('account');"><i class="material-icons">person</i></p>
        <hr class="menuRule">
        <?php if(isset($_COOKIE["user"])){echo('<p onclick="menuBtnClick(\'search\');"><i
class="material-icons">search</i></p>');}?>
        <p onclick="menuBtnClick('home');"><i class="material-icons">home</i></p>
        <p onclick="menuBtnClick('tool');"><i id="toolBtn" class="material-icons"
onclick="menuBtnClick('tool');">build</i></p>
        <p onclick="menuBtnClick('news');"><i class="material-icons">announcement</i></p>
        <p onclick="menuBtnClick('about');"><i class="material-icons">help</i></p>

        <p onclick="menuBtnClick('forum');"><i id="forumBtn" class="material-icons">subject</i>
```

```
</p>
        <hr class="menuRule postBtn">
        <p id="forumPostMenuBtn" class="postBtn" onclick="displayPU('postingPU');"><i
class="material-icons">create</i></p> <!-- This will only display when the forumPage or
forumSinglePage is being viewed -->
      </div>
      <div id="searchPage" class="page">
        <div id="searchFormCon">
        </div>
        <div id="searchResultsCon" class="textCon"> <!-- This is where the results show up from
a search. -->
        </div>
      </div>
      <div id="homePage" class="page">
        <p id="homeTitle" class="coloredText redPurple">Nomios</p>
        <br>
        <p id="homeSubTitle">A tool for histone modifications, a forum for scientists.</p>
        <div class="textCon">
        </div>
      </div>
      <div id="toolPage" class="page">
        <div id="toolPageContent" class="textCon">
        </div>
      </div>
      <div id="toolSingleViewPage" class="page">
        <div class="textCon">
          <p class="subTitle"><?php isset($_GET["disease"]) ? $t = "Disease" : $t = "Sequence";
echo ($t); #This prints out a title $t depending on whether or not `disease` is set in the url.
If so we can print out the name of the disease. Otherwise we use a generic , descriptive title.
?></p>
          <div id="toolSingleViewPageContent">
            <br><br>
          </div>
        </div>
      </div>
      <div id="newsPage" class="page">
        <div class="textCon">
          <p class="subTitle">News</p>
          <div class="strip redPurple">
            <br>
            <p class="text">No news!</p>
          </div>
          <br><br>
        </div>
      </div>
      <div id="aboutPage" class="page">
        <div class="textCon">
          <p class="subTitle">About</p>
          <br>




          <div class="strip redPurple"<p class="text"><b>Nomios</b> is a single platform for two
```

```
services: </p><ulclass="text"><li>A forum for scientists. No matter what specialists field you are in, you can meet others, collaborate and learn on a single platform.</li><li>A histone modification interpreter for epigeneticists. Simulate your experiments and save time and resources. Find what the overall change in expression is for a particular histone modification sequence.</li></ul><p class="text">Founded in 2018, this platform was designed to encorage global research and quick access to knowledge concerning any questions that a scientist today might have.</p></div>
          <br>
          <p class="subTitle">help</p>
          <br>
          <div class="strip greenYellow">The following contains a simple guide for use of the <b>forum</b>: <ul><li>In order to enter the forum first click on the <i class="material-icons">subject</i> icon.<li>To start a new thread click on the <i class="material-icons">create</i> icon.</li><li>To view a thread click on one of the thread items that are displayed.</li><li>Once viewing a thread click on the <i class="material-icons">create</i> icon to reply to the original post, or click on a response to reply to it.</li><li>Mark-up is supported. <b>[b][/b]</b> emboldens text. <em>[i][/i]</em> creates italics. [l][/l] specifies a list, with [li][/li] being the items on this list. Mark-up can only be used in the <em>Message</em> components of a post.</li></ul></div>
          <br>
          <div class="strip orangeBlue">The following contains a simple guide for use of the <b>histone modfication interpreter</b>:<ul><li>Click the <i class="material-icons">build</i> icon to create a disease or nucleosome sequence.</li><li>Click the <i class="material-icons">search</i> icon to search for a particular nucleosome sequence or disease.</li><li>On a Disease page, click the nucleosome sequence block to view that sequence.</li><li>On a Sequence page, click the disease block to view the associated disease's page.</li><li>Submit the edit forms on both types of page to confirm edits to a sequence or disease that you have created.</li></ul></div>
          <br><br>
        </div>
      </div>
      <div id="forumPage" class="page"> <!-- For viewing links to different threads and posting a new thread. -->
        <div class="textCon">
          <p class="subTitle">Forum</p>
            <div id="forumPageContent">
            </div>
          </div>
        </div>
        <div id="forumSingleViewPage" class="page"> <!-- For viewing one particular thread and posting messeges and replies to messages to that thread. -->
        <div class="textCon">
          <p class="subTitle">Forum</p>
          <div id="forumSingleViewPageContent">
            </div>
          </div>
        </div>
        <div id="accountPage" class="page">
          <div class="textCon">
            <p class="subTitle" id="accountSubTitleStrip">Account</p>
            <div id="accountPageContent">
            </div>

          </div>
```

```
        </div>
      </div>
      <?php
        $trim = array_map('trim', $_POST); # Each data in the $_POST superglobal array
(submiited via POST) has excess spaces removed to lower the amount of storage used by the
database.
        $trim = array_map(array($db, 'real_escape_string'), $trim); # Each data submitted via
POST now escapes any problematic characters, sucha as quotiation marks that could distort
queries in the from of strings.
        $popupTop = '<div class="boardConPU"><div class="popUpBox redPurple"><div
class="textConPU"><p class="titlePU">Notification<span class="crossPU">X</span></p>';
        $altPopupTop = substr($popupTop, 0, 23).' style="display: none"
id="postingPU">'.substr($popupTop, 24); # For forum POST forms.
        $popupBottom = '</div></div></div>';
        # Loads the PHP scripts that are neccessary for website function.
        # Require_once makes sure that only one version of the file is included and prevents the
execution of the rest of this script if it is not present - preventing a user from running an
errornous version of this program.
        require_once("php/toolFunctions.php");
        require_once("php/dictionary.php");
        require_once("php/hashing32.php");
        require_once("php/queue.php");
        require_once("php/onload/onloadLOAD.php");
        require_once("php/submission/submissionLOAD.php");
        # Tags that need to be loaded on start but require PHP to display the correct
information.
        if(isset($_COOKIE["user"])){ # If signed into an account.
          if(isset($_GET["thread"])){ # If we are displaying the forumSingleViewPage we must
have a thread variable in the query string of the URL in order to view a single thread. If so
the forms for posting will be different to that of the forumPage.
              echo($altPopupTop.'<br>Reply To The Original Post<br><br><form action="index.php?
page=forum&thread='.$_GET["thread"].'" method="post"><textarea name="messagePM"
class="textareaPU PUInput" info="Message" onfocus="clearValue(this); selected(this);"
onblur="restoreValue(this); deselected(this);">Message</textarea><br><br><input
name="submittedPM" type="submit" class="button btnPU PUInput"/></form>'.$popupBottom);
              # The above line create a form that is used for posting messages to the original
post. Submittted via POST.
              echo(substr($popupTop, 0, 23).'style="display: none"
id="replyingPU">'.substr($popupTop, 24).'<br>Reply To A Message<br><br><form action="index.php?
page=forum&thread='.$_GET["thread"].'" method="post"><textarea name="messagePR"
class="textareaPU PUInput" info="Message" onfocus="clearValue(this); selected(this);"
onblur="restoreValue(this); deselected(this);">Message</textarea><br><br><input id="midPR"
name="midPR" type="hidden" value=\'\'/><input name="submittedPR" type="submit" class="button
btnPU PUInput"/></form>'.$popupBottom); # Take the first part of the popup and then concatenate
HTML makes up a form for posting replies to messages.  Submittted via POST.
          }
          else { # If not displaying the forumSingleView (therefore no thread be displyed).
              $subject = isset($trim["subjectPT"]) ? $trim["subjectPT"] : "Subject"; # If the
thread post was tried and failed, retain the data so that it can be corrected and resubmitted.
              $message = isset($trim["messagePT"]) ? $trim["messagePT"] : "Message"; # ^ Else
print the default value.


              echo($altPopupTop.'<br>Start A Thread<br><br><form action="index.php?page=forum"
```

```
method="post"><textarea class="textareaPU subjectTextareaPU PUInput" name="subjectPT"
info="Subject" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this);
deselected(this);">'.$subject.'</textarea><br><br><textarea name="messagePT" class="textareaPU
PUInput" info="Message" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this);
deselected(this);">'.$message.'</textarea><br><br><input name="submittedPT" type="submit"
class="button btnPU large"/></form>'.$popupBottom);
              # The above line of code prints out a form for posting a thread. Submittted via
POST.
          }
        }
      ?>
      </div>
   </body>
</html>
```

css/accountStyles.css

This contain the CSS code that stylises the account page.

```
/*IDs*/
  #signInForm, #signUpForm {
    min-width: 49%;
    max-width: 49%;
    height: auto;
    max-height: none;
    float: left;  /* This aligns the element either to the left or right. */
    border-radius: 3px; /* This rounds the corners of elements. */
    border-style: none; /* Changes the style-type of the border to be not exist. */
  }

  #signInForm > form, #signUpForm > form { /* The forms within the #signUpForm and #signInForm
elements. */
    margin: 5%; /* Adds space to the outside of the element to adjust the element itself. */
  }

  #signInForm {
    float: right;
    vertical-align: top; /* Sets the element to be inline with the top of its adjacent element.
*/
  }

  #con { /*Is a container for the sigg in and sign up forms.*/
    min-width: 100%;
    max-width: 100%;
    min-height: 100%;
    max-height: 100%;
  }

  #downButton {
    display: block;


    margin-right: auto; /* Automatically sets both left and right margins of the element to be
```

```
equal. */
    margin-left: auto; /* ^ */
    cursor: pointer; /* Changes the texture of the cursor. */
  }

  #accountDetailsForm {
    border-radius: 3px;
    border-style: none; /* Border is reduced to minium thickness */
    padding: 5%; /* Adjusts space inside an element to move elements inside the element. */
  }
```

css/animations.css

This contains the CSS animations that are used in other CSS files.

```
/* On first load */
  @keyframes bounce {
    50% {transform: translateY(-5%);}
    100% {transform: translateY(5%);}
  }

  @keyframes flyAway {
    100% {transform: translateY(2000px);}
  }

  @keyframes floatOne {
    50% {transform: translate(10%, 10%);}
  }

  @keyframes floatTwo {
    50% {transform: translate(-10%, -10%);}
  }

  @keyframes floatThree {
    50% {transform: translate(-10%, 10%);}
  }

  @keyframes floatFour {
    50% {transform: translate(10%, -10%);}
  }

  /*For page switching*/
  @keyframes fadeIn {
      100% {opacity: 1;}  /* 'opacity' sets how opaque the element is. */
  }
  @keyframes fadeOut {
      100% {opacity: 0;}
  }
```

css/entryStyles.css

This is the CSS code that stylises the landing page (this is a page for a first-time user).

```
/*IDs*/
  #enterBtn {
    display: block;
    cursor: pointer;
    transition-timing-function: ease-in-out; /* Specifies the speed curve. Slow start, quick
end. */
    position: relative;
    z-index: 2; /* Specifies the z component of an element. The higher the 'closer' it is to the
viewport. */
  }

  #bgObjCon {
    font-family: 'Megrim', cursive;
  }

    #bgObjCon > div {
      animation-name: floatOne;
      animation-duration: 5s;
      animation-fill-mode: forwards;
      animation-iteration-count: infinite; /* Constantly repeats. */
      transition-timing-function: ease-in-out;
      color: #333; /* Sets the color of the font. */
    }
/*Elements*/
  h1 {
    font-family: 'Megrim', cursive; /* Sets the font to be used. */
    font-size: 1000%;
    text-align: center;
    padding-left: auto;
    padding-right: auto;
    box-sizing: border-box; /* Icludes the padding and border in the total width of the element.
*/
  }
/*Classes*/
  /*On first load*/
    .bgObjBig {
      font-size: 600%;
    }

    .bgObjMed {
      font-size: 400%;
    }

    .bgObjSmall {
      font-size: 200%;
    }
```

css/forumStyles.css

This is the CSS code that stylises the forum.

```
/*Classes*/
  .forumObj { /*This applies to any type of post in a thread.*/
       border-style: none;
       border-radius: 3px;
     display: flex; /* Displays as a flexbox. This element is now flexible, whcich gives it access to the flex properties. */
   }

   .forumCon > em, .forumCon > b, .forumCon > ul, .forumCon > t, .forumCon > br, ul > li, em > b, b > em, t > em, t > b, t > ul, t, em { /*t = I created this tag element.*/
     color: #EFEFEF; /*Makes sure that the rule is directly taken by each element in a forumObj ready for nightmode. */
   }

   .forumCon {
     min-width: 90%;
     max-width: 90%;
     padding: 5% 5% 5% 5%; /* All elements inside leave a border around the edge of the menubar and the rest of the viewport. */
   }

   .forumNameTag {
     float: right;
   }
   /*Post types*/
       .message { /*Reply to an original post.*/
            cursor: pointer;
            margin-left: 5%;
            width: 95%;
       }

       .reply { /*Reply to a message.*/
            float: right;
            margin-left: 10%;
            width: 90%;
          margin-bottom: 2%;
        }
```

css/general.css

This is the CSS code that stylises HTML elements from different types of pages.

```
/*IDs*/
  #dayChangeBtn {
    position: fixed; /* Stays at a specifed location and does not move based on the nearest, upper nested relative element.*/
    right: 2vw;
    top: 2vw;
    z-index: 1000; /* Stays on top of all elements so that it can always be accessed. */
    color: #666;
  }
```

```
/*Elements*/
  ::selection { /* The style of highlighted text. */
    background-color: inherit; /* No change in style to keep the website style continous. */
    color: inherit;
  }

  body {
    background-color: #EFEFEF;
    color: #333;
    font-family: 'Comfortaa', cursive;
    overflow: hidden; /* Any excess width or height of an element to the outside of the viewport
is hidden away from view. */
    box-sizing: border-box;
    cursor: default;
  }

  div {
    display: block; /* Displayed as a section of the webpage. */
    box-sizing: border-box;
  }

  p {
    margin-bottom: -1%; /* Pushes text downwards. */
  }

  hr {
    border-width: thin;
    color: #333;
    border-style: solid;
  }
/*Classes*/
  .floatAesthetic {
    box-shadow: 7px 7px 17px #AAA; /* Creates a shadow effect from an element. */
  }

  .text {
    font-size: 100%;
    max-width: 100%;
    min-width: 100%;
    overflow-wrap: break-word; /* Defines where breaks are places in words when overflowing.
This will only break a word if a word cannot be placed on its own line without overflowing. */
    color: #EFEFEF;
  }

  .text > b, text > em {
    color: #EFEFEF;
  }
  /*Coloured text classes*/
    .coloredText {
      -webkit-text-fill-color: transparent; /* Lets us see the background behind a peice of
text. */


      -webkit-background-clip: text !important; /* Sets the background of an element to only be
```

```
displayed where the tet is present. */
    }

    /*Colors*/
      .redPurple { /*xY: starts with x colour ends with Y colour.*/
        background: linear-gradient(to top right, #F2092C 0%, #B50FC4 100%); /* Creates a color
gradient from the bottom left corner to the top right corner.*/
      }

      .greenYellow {
        background: linear-gradient(to top right, #1FC63A 0%, #E8D31E 100%);
      }

      .orangeBlue {
        background: linear-gradient(to top right, #F2870A 0%, #11E3C3 100%);
      }

  /*Other*/
    .large{ /* Occupies all space availabale to the element in the x-axis.*/
      min-width: 100%;
      max-width: 100%;
      max-height: 10%;
      min-height: 10%;
      font-size: 100%;
      margin-left: 0;
    }
  /*Results from queries*/
    .strip {
      min-width: 100%;
      max-width: 100%;
      border-radius: 3px;
      border-style: none;
      color: #EFEFEF;
      font-size: 150%;
      cursor: pointer;
      padding: 5%;
    }

      .strip > hr { /* Sets the colour of the horizontal rule within a strip element. */
        color: #EFEFEF;
      }
```

css/homeStyles.css

This contains the CSS code that is specific to the home page.

```
/*IDs*/
  #homeTitle {
    font-size:  1000%;
    font-weight: 700; /* Sets the boldness of the text. */
    margin-top: -2%; /* Pushes the element upwards. */
  }

  #homeSubTitle {
    font-size:  300%;
    font-weight: 400;
    margin-top: -2%;
    color: #333;
  }
```

css/inputStyles.css

This contains the CSS code specific to stylising input elements of HTML forms.

```
/* IDs */
#dnaSequenceT , #dnaSequenceTE { /* Sets the DNA sequence to its typical, capitalised form. */
  text-transform: uppercase;
}
/* Elements */
  select, input, textarea, .button {
    background: rgba(238, 238, 238, 0.7); /* Sets the background to be partially transparent. */
    color: #333;
    border-style: none;
    border-radius: 3px;
    cursor: pointer;
    width: 100%;
    min-height: 8%;
    font-family: "Roboto Mono", monospace;
    margin-right: auto;
    margin-left: auto;
    margin-bottom: 2%;
    padding: 5px;
  }

  select > option { /* This makes sure that the optiosn are always visible regaerdless of colour
mode. */
    color: #333;
  }

  form {
    max-width: 90%;
    min-width: 90%;
    margin: 0;
    margin-top: 2%;
  }

  textarea {
    min-width: 100%;
```

```
    max-width: 100%;
    min-height: 40%;
    max-height: 40%;
  }

    textarea::-webkit-scrollbar {
      background-color: #EFEFEF;
    }

    textarea::-webkit-scrollbar-thumb {
      background-color: #333;
    }
    /*Red button*/
      .redBtn {
        background-color: #E50000 !important;
        border-color: #E50000 !important;
        color: #EFEFEF !important;
      }

      .redBtn:hover, #signOutBtn:hover { /*Added specifity to overide button hover for
signOutBtn.*/
        background-color: #CC0000 !important;
        border-color: #CC0000 !important;
      }

      .redBtn > p {
        transform: translateY(50%); /* Moves the element downwards by half of the space
available to it. */
      }
  /*Button*/
    .button {
      text-align: center;
      min-height: 12%;
      max-height: 12%;
      cursor: pointer;
      p {
        display: block;
        height: 50%;
        transform: translateY(-50%);
        font-size: 120%;
      }
    }

    button:hover, input:hover, select:hover, textarea:hover  { /*Used hover here AND javaScript
for input hovering so as */
      background-color: #EFEFEF !important;     /*to include submit inout buttons. Also to
achieve hover effect.*/
      border-color: #EFEFEF !important;
    }
  .removeDeleteButton {
    margin-left: 5%;
    margin-right: 5%;

    margin-top: 2%;
```

```
    margin-bottom: -2%;
  }
  /*Removes chrome slection effects.*/
    textarea:hover,
    input:hover,
    textarea:active,
    input:active,
    textarea:focus,
    input:focus,
    button:focus,
    button:active,
    button:hover,
    label:focus,
    .btn:active,
    .btn.active
    {
      outline:0px !important;
      -webkit-appearance:none;
    }
```

css/mediaQueries.css

This contains CSS code changes depending on the dimensions of the screen of the device that the user is using.

```
@media only screen and (max-width: 500px) { /*For mobile displays (or any other display that ahs
a maximum width of 500 pixels. This is NOT the width of the viewport.)*/
  #menuCon {
    max-height: 5%;
    min-height: 5%;
    max-width: 100%;
    min-width: 100%;
    display: block;
    margin-bottom: 5%;
    z-index: 101;
  }

  #menuCon p {
    display: inline-block;
    top: 50%;
    transform: translateY(-50%);
    margin-left: 2%;
    margin-right: 2%;
  }

  #dayChangeBtn { /* Moves the button the chnage the color scheme to a more suitable location.
*/
    top: 0.5%;
    right: 1%;
  }


  #accountPage, #homePage, #toolPage, #newsPage, #aboutPage, #forumPage, #forumSingleViewPage,
```

```
#toolSingleViewPage, #searchPage {
    margin-left: 5%; /* Lets us occupy slightly more room now that the menu bar has gone, and
keeps a margin for easier reading.*/
    margin-right: 5%; /* ^ */
    margin-top: 12%; /* Shifts all pages downwards to make room for the new postion of the menu
bar. */
  }

  .doubleForm {
    margin-bottom: 5%;
    min-width: 100%;
    max-width: 100%;
  }

  .menuRule {
    display: none; /* No horizontal rules in the menu bar in the new position. */
  }
}
```

css/pages.css

This is the CSS code that is specific to stylising the fundamental parts of the pages.

```
/*IDs*/
  #mainCon { /*Where all non-menu content is.*/
    opacity: 0;
    position: fixed; /* Occupys the entire viewport. */
    top: 0px; /* ^ */
    left: 0px; /* ^ */
    width: 100%;
    height: 100%;
    min-width: 1px; /*Fills up if nothing is in it.*/
    min-height: 1px; /* ^ */
    animation-duration: 4s; /*For first load.*/
    animation-iteration-count: 1;
    animation-fill-mode: forwards;
    overflow-x: hidden; /*No horizontal scrolling*/
  }

  #mainCon::-webkit-scrollbar { /* Sets the properties of the scrollbar background. */
    background-color: #EFEFEF;
    width: 1%;
  }

  #mainCon::-webkit-scrollbar-thumb { /* Sets the properties of the scrollbar 'thumb' (the bit
that we use to scroll). */
    background-color: #333;
  }

  #menuCon { /*Where all menu content is.*/
    position: fixed;

    width: 5%;
```

```
    height: 100%;
    min-width: 1px;
    min-height: 1px;
    background-color: #333;
    text-align: center;
    color: #EFEFEF;
  }

  #homePage { /*This is the first page to load so it needs to be already viewable when the page loads.*/
    opacity: 1;
  }

/*Classes*/
  /*Menu-specific classes*/
    .material-icons {
      cursor: pointer;
    }

    .menuRule {
      border-color: #EFEFEF;
      width: 70%;
    }

    .postBtn { /*This is the button for posting a thread/message in a thread.*/
      display: none;
    }

  .page { /*A page is where specific content. E.g. All tool content is in the tool page. Not to be confused with a web page.*/
    opacity: 0;
    position: absolute; /* Specifies a particular location within the nearest, upper nested relative element. This is usually the body. */
    top: 0px;
    left: 0px;
    width: 91%;
    height: 96%;
    min-width: 1px;
    min-height: 1px;
    margin-left: 7%;
    margin-right: 2%;
    margin-bottom: 4%;
    margin-top: 2%;
    animation-iteration-count: 1;
    animation-duration: 1s;
    animation-fill-mode: forwards;
    transition-timing-function: ease-in-out;
  }

  /*Classes found within pages.*/
    .textCon { /*Puts elements below the subtitle*/
      width: 100%;

      height: 100%;
```

```
      min-width: 1px;
      min-height: 1px;
      margin-top: -5%;
    }

    .subTitle {
      font-size: 400%;
    }
```

css/popUp.css

This CSS code is used to style any "pop ups". These are HTML blocks that contain notifications or forms for the user to fill out. They are useful at drawing attention.

```
/*Classes*/
  .boardConPU{ /*This is what dims the background*/
    position: absolute;
    top: 0px;
    left: 0px;
    min-width: 100vw; /* 1vw = viewport width/100 */
    min-height: 100vh; /* 1vh = viewport height/100 */
    max-width: 100vw;
    max-height: 100vh;
    background-color: rgba(64, 64, 64, 0.6);
  }

  .popUpBox { /*This is what makes the actual pop up.*/
    position: relative;
    border-radius: 3px;
    border-style: none;
    border-width: 5px;
    max-width: 70%;
    min-width: 70%;
    max-height: 70%;
    min-height: 70%;
    margin-left: auto;
    margin-right: auto;
    margin-top: 7.5%;
    box-sizing: border-box;
    z-index: 999;
    box-shadow: 2px 2px 10px #222;
    overflow-y: auto; /* Lets us scroll through the content of the notification. */
  }

    .popUpBox::-webkit-scrollbar { /* Sets the properties of the scrollbar backgound. */
      background-color: transparent;
      width: 1%;
    }

    .popUpBox::-webkit-scrollbar-thumb {  /* Sets the properties of the scrollbar draggable
part. */

      background-color: #EFEFEF;
```

```
    }

    .textConPU { /*This is where the body of the pop up goes.*/
      color: #EFEFEF;
      margin-top: 5%;
      margin-left: 5%;
      margin-bottom: 5%;
      margin-right: 5%;
    }

    .titlePU {
      color: #EFEFEF;
      font-size: 300%;
      font-weight: bolder;
    }

    .crossPU { /*For closing the pop up*/
      float: right;
      cursor: pointer;
    }

    /*Pop up inputs*/

      .textareaPU {
        background-color: #EFEFEF;
        border-color: #EFEFEF;
        text-transform: none; /* Keeps the user inputted text as unchnaged in term sof
capitalisation. */
        color: #333;
      }

      .subjectTextareaPU {
        max-height: 20%;
        min-height: 20%;
      }
```

css/search.css

This CSS code stylises the page that contains the searching function.

```
/*IDs*/
#searchBoxForm {
  text-align: center;
  padding: 5% 5% 5% 5%;
}

#inputsCon {
  max-width: 100%;
  min-width: 100%;
}


  #searchType { /* The part of the search form which lets us specify what type of data we are
```

```
looking for. */
    color: #EFEFEF;
    max-width: 30%;
    min-width: 30%;
    margin-right: 2%;
    float: left;
    font-size: 110%;
  }

  #searchBox { /* The part of the search form which we type into. */
    color: #EFEFEF;
    max-width: 50%;
    min-width: 50%;
    font-size: 110%;
    margin-right: 2%;
  }

  #searchBtn {
    color: #EFEFEF;
    max-width: 16%;
    min-width: 16%;
    float: right;
    max-height: 8%;
    min-height: 8%;
    font-size: 110%;
  }
```

css/toolStyles.css

The "tool" is the histone modification interpreter. This file contains the CSS code that stylises the tool page.

```
/*IDs*/
  #histoneModDiv { /*For containing the histone modification buttons.*/
    border-width: 5px;
    border-radius: 3px;
    background-color: rgba(238, 238, 238, 0.7);
  }

  #hmodBg { /*For displaying the mod sequence*/
    border-style: none;
    border-radius: 3px;
    color: #333;
    min-width: 100%;
    max-width: 100%;
    min-height: 40%;
    overflow-y: auto;
    background: #EFEFEF;
  }

  #diseaseForm, #toolForm {
    border-radius: 3px;

    padding: 5% 5% 5% 5%;
```

```
    color: #EFEFEF;
  }

  #removeHistoneBtn { /* For creating a sequence. */
    margin-right: 0%;
  }

  #removeHistoneBtnTE { /* For editing a sequence */
    margin-right: 1%;
  }
/*Classes*/
  .histoneMod {
    max-width: 47%;
    min-width: 47%;
    margin: 1% 1% 1% 0%; /*Top right bottom left*/
    max-height: 10%;
    min-height: 10%;
    display: inline-block;
    box-sizing: content-box; /* Sets the width of the element to not include the border and
margin. Lets us space out elements using margin. */
    border-style: none;
    border-radius: 3px;
  }

  .histoneMod:hover {
    background-color: #EFEFEF;
    border-color: #EFEFEF;
    color: #333;
  }

  .toolCon {
    margin: 5% 5% 5% 5%;
  }

  .sequenceDiseaseTitle {
    font-size: 120%;
    color: #333;
    font-weight: bolder;
  }
```

php/onload/account.php

This file contains code to do with the account page that is executed once `index.php` is loaded.

```
<?php
  if(isset($_GET["hash"]) && isset($_GET["email"])){ // If the user is activating their account
    $hash = $_GET["hash"]; // Get the hash from the URL query
    $ea = $_GET["email"];
    $q = "UPDATE users SET hash = NULL WHERE emailAddress='$ea' AND hash='$hash'"; // Set the
hash to NULL so that the user can log in.
    $r = mysqli_query($db, $q); // Processes the query $q and returns the result.

    if($r){ // If the result actually returns a result.
```

```
      echo($popupTop);
      echo("<p>You can now sign in.</p>");
      echo($popupBottom);
      // Display a notification letting them know that activation was successfull to the user.
    }
  }

  $emailAddress = isset($trim["emailAddressSI"]) ? $trim["emailAddressSI"] : "rf@example.com"; # If the user has tried to sign in, save the inputted email address so that they can correct their mistakes.
  # w = x ? y : z ; Means w is set to y if x is true. If x is false set w to z.
  if(isset($_COOKIE["user"])){ // If the user is signed in.
    $q = "SELECT * FROM users WHERE uid=".$_COOKIE['user'];
    $r = mysqli_query($db, $q);
    $num = mysqli_num_rows($r); // Get the number of returned rows.
    if($num == 1){ // If user account is returned (each user row is unqiue so it must be equal to 1).
      $name = ($userRow[1]." ".$userRow[2]); // Get the name.
      $ea = $userRow[3]; // Get the email address.
      $a = 'Standard account'; // Initialise the admin string variable.
      if($admin == TRUE){ // If the user is an admin
        $a = 'Admin account';
      }
      echo('<script>loadUserData("'.$name.'", "'.$ea.'", "'.$a.'");</script>'); // Display the user account data and password form.
    }
    else if($num != 1) { // If the account is not unqiue, must be malicous. Or if it does not exist we need to display the correct account page.
      echo('<script>document.cookie = "user=;expires='.(time()-100).'"; document.getElementById("accountPage").innerHTML = \''.$accountMenu.'\';</script>'); # Delete user cookie if user not found in database.
      echo('<script>loadAccountPage('.$emailAddress.');</script>'); # Display the sign in and sign up forum
    }
    else {
      echo('<script>loadAccountPage("'.$emailAddress.'");</script>');
    }
  }
  else {
    echo('<script>loadAccountPage("'.$emailAddress.'");</script>');
  }
?>
```

php/onload/forum.php

This forum-page related code is executed once `index.php` is loaded.

```
<?php
    function convert($m) { # Converts artificial mark-up to HTML.
        $chars = ['[i]', '[/i]', '[b]', '[/b]', '[l]', '[/l]', '[li]', '[/li]']; # All of thw artificial mark up tags.

        $replaceChars = ['<em>', '</em>', '<b>', '</b>', '<ul>', '</ul>', '<li>', '</li>']; #
```

```
HTMl tags.
        for($i=0;$i<count($chars);$i++){
            $m = str_replace($chars[$i], $replaceChars[$i], $m);
            # Replace any sub-string within the message $m, that matches the artifical mark up
tag, with the actual HTML mark up. DO this for each artificial mark up tag.
        }
        return $m;
    }

    # Displays all threads.
    $q = "SELECT tid, subject, uid, dateTime FROM thread ORDER BY dateTime DESC"; # Fetches all
threads and order them from newest to oldest to keep the forum relevent to current times.
    $r = mysqli_query($db, $q);

    $num = mysqli_num_rows($r); # Gets the number of rows that were returned.

    echo("<script>document.getElementById('forumPageContent').innerHTML =\"<br><br>"); # Print
some javascript that executes once the page has loaded. This javascript prints out HTML mark-up.
    if($num < 1){ # If no threads exist, display a message to indicate this to the user.
        echo("Nothing here! Try posting a new thread by clicking on the 'write' icon at the
bottom of the bar.");
    }
    else {
        $threads = new queue($num); # Create a new thread object.

        # Note that the order of the array that is reutrned by mysqli_fetch_array is in the
order of the table attributes specified in the query, or in the order of the attributes of the
table if they are not specified in the query.
        while($row = mysqli_fetch_array($r)){ # For each thread.
            $fn = mysqli_fetch_array(mysqli_query($db, "SELECT firstName FROM users WHERE
uid=".$row[2]))[0]; #  Get the first name of the user who posted it.
            $s = $row[1]; # Fetches the subject of the thread.
            if(strlen($s) > 25){ # This makes sure that only a set amount of the subject text is
printed so as not to overflow the strip.
                $s = substr($s, 0, 25)."...";
            }
            $html = "<div class='strip greenYellow floatAesthetic' onclick='window.location.href
= `index.php?page=forum&thread=$row[0]`;'><a>$s</a><hr><br><a style='float: right;'>$fn at
".$row[3]." GMT</a>";
            # Each thread displayed as a strip with a link to the forum sing view so that we can
view a single thread.
            if($admin == TRUE){ # If the user is an admin, they should have the power to delete
inappropriate threads. Display a form to do that.
                $html .= "<form action='".$_SERVER["REQUEST_URI"]."' method='post'><input
name='deleteTid' type='hidden' value='".$row[0]."'/><input class='removeDeleteButton'
type='submit' value='Delete'/></b></form>";
            }
            $html .= "</div><br>";
            $threads->append($html); # Adds the HTML markup as an element to the back of the
queue.
        }


        do {
```

```
            echo($threads->get()); # Display the element at the front of the queue.
            $threads->pop(); # Remove that that element.
        } while ($threads->isEmpty() == FALSE); # Repeat until the queue is empty.
    }
    echo("\";</script>");
    # ---

    if(isset($_GET["thread"]) && isset($_COOKIE["user"])){ # If we are in are viewing a single
thread on forumSingleViewPage.
        $tid = $_GET["thread"]; # Get the thread id.
        $r = mysqli_query($db, "SELECT mid FROM message WHERE tid = ".$tid); # Select all
messages
        $num = mysqli_num_rows($r); # Find the number of returned rows for part of the length of
the queue.
        while($mid = mysqli_fetch_array($r)[0]){
            $num += mysqli_num_rows(mysqli_query($db, "SELECT rid FROM replies WHERE mid =
".$mid)); # Adds the number of replies for each number to the total length of the queue.
        }

        $posts = new queue($num+1); # This will contain the html with the original post first,
then each message followed by their replies in order of oldest to newest. We add one to the
queue length to account for the original post.

        echo("<script>menuBtnClick('forumSingleView');
document.getElementById('forumSingleViewPageContent').innerHTML=`"); # Begin the JavaScript
script.
        $r = mysqli_query($db, "SELECT subject, message, firstName, dateTime FROM thread INNER
JOIN users ON thread.uid = users.uid WHERE tid=".$tid); # Fetch the original post of the thread
and the user's first name who psoetd it.
        $rowT = mysqli_fetch_array($r); # Store the extracted data in an array.
        $m = convert($rowT[1]); # Converts artificial mark up to valid HTML in the message.
        $OPHTML = "<br><br><div class='forumObj greenYellow floatAesthetic'><div
class='forumCon'><b>$rowT[0]</b><br><t>$m</t><br><em class='forumNameTag'>$rowT[2] at $rowT[3]
GMT</em></div></div>"; # Displays the original post.

        $posts->append($OPHTML);

        $r = mysqli_query($db, "SELECT message, firstName, mid, dateTime FROM message INNER JOIN
users ON message.uid = users.uid WHERE tid=".$tid); # Fetch data on all messages to the original
post including the first name of the use who posted it.
        while($rowM = mysqli_fetch_array($r)){ # For each message.
            $m = convert($rowM[0]); # Retrive the message body.
            $mHTML = "<br><div class='forumObj message redPurple floatAesthetic'><div
class='forumCon' onclick='document.getElementById(\"replyingPU\").style.display = \"block\";
document.getElementById(\"midPR\").value = \"".$rowM[2]."\";'><div><t>$m</t><br><em
class='forumNameTag'>$rowM[1] at $rowM[3] GMT</em></div>";
            # The above line displays the message.
            if($admin == TRUE){ # Dispaly a form to delete the message if the user is an admin.
                $mHTML .= '<form action="'.$_SERVER['REQUEST_URI'].'" method="post"><input
name="deleteMid" type="hidden" value="'.$rowM[2].'"/><input class="removeDeleteButton"
type="submit" value="Delete" onclick="document.getElementById(\'replyingPU\').style.display =
\'none\';"/></b></form>';

            }
```

```
            $mHTML .= "</div></div><br>";
            $posts->append($mHTML);
            $rR = mysqli_query($db, "SELECT message, firstName, rid, dateTime FROM replies INNER
JOIN users ON replies.uid = users.uid WHERE replies.mid=".$rowM[2]); # Fetch all the repleis
foreach message, aswell as the firstname of the suer who posted it.
            while($rowR = mysqli_fetch_array($rR)){
                $m = convert($rowR[0]);
                $rHTML = "<div class='forumObj reply orangeBlue floatAesthetic'><div
class='forumCon'><t>$m</t><br><em class='forumNameTag'>".$rowR[1]." at ".$rowR[3]." GMT</em>";
                #Display the reply.
                if($admin == TRUE){ # Dispaly a form to delete the repy if the user is an admin.
                    $rHTML .= '<form action="'.$_SERVER['REQUEST_URI'].'" method="post"><input
name="deleteRid" type="hidden" value="'.$rowR[2].'"/><input class="removeDeleteButton"
type="submit" value="Delete"/></b></form>';
                }
                $rHTML .= "</div></div><br>";
                $posts->append($rHTML);
            }
        }

        do {
            echo($posts->get()); # Print the post
            $posts->pop(); # Remove it from the queue.
        } while($posts->isEmpty() == FALSE); # Repeat until the queue is empty.
        echo("<br><br>`;</script>"); # End the JavaScript script.
    }
?>
```

php/onload/search.php

This search-page related code is only executed once `index.php` has loaded.

```
<?php
  if(isset($_COOKIE["user"])){ # Only display the page if the user has signed in.
    # For searching through forum posts and sequences. selected="selected" is used instead of
just selected as it is XHTML compliant. This also indicated the default option of select.
    $search = isset($trim["searchValue"]) ? $trim["searchValue"] : "Search"; # Get the
previously entered value searched for if the user has searched before. Otherwise set toa
defualt value.
    $searchType = isset($trim["searchType"]) ? $trim["searchType"] : "";
    echo('<script>document.getElementById("searchFormCon").innerHTML=`'); # Start the JavaScript
script.
    echo('<form id="searchBoxForm" action="index.php?page=search" method="post">
      <div id="inputsCon">
        <select id="searchType" name="searchType" class="redPurple floatAesthetic">'); # Begin
displaying the search form.
    $optionKeyValues = ['currentUserSequences', 'My sequences', 'currentUserDiseases', 'My
diseases', 'currentUserThreads', 'My threads', 'sequences', 'All sequences', 'diseases', 'All
diseases', 'threads', 'All theads'];



    # Above line creates an array that holds the value (the key) and the actual text (the value)
```

```
displayed to the user, for each user in the respective format. E.g. w, x, y, z where w is a
value and x is a peice of text and w,x and y,z are pairs of values.
    $optionDictionary = new dictionary($optionKeyValues); # Create a dictionary object using the
constructor defined in dictionary.php and the array previosuly defined.
    if($searchType != ""){ # If the user has entered a search term before, display that as the
default type of search to be conducted.
      $submittedValue = $optionDictionary->read($searchType); # Gets the text to display to the
user.
      echo('<option value="'.$searchType.'" selected="selected">'.$submittedValue.'</option>');
# Prints the option.
      $result = $optionDictionary->remove($searchType); # Removes the option from the dictionary
so it is not reprinted.
    }
    $optionKeys = $optionDictionary->keys; # Fetches all keys from the dictionary.
    for($i=0;$i<count($optionKeys);$i++){ # For each key.
      $currentKey = $optionKeys[$i];
      $currentValue = $optionDictionary->read($currentKey); # Fetch the value associated with
that key.
      echo('<option value="'.$currentKey.'">'.$currentValue.'</option>'); # Print the option.
    }
    echo('</select>
        <input id="searchBox" name="searchValue" class="redPurple floatAesthetic" type="text"
value="'.$search.'" info="Search" onfocus="clearValue(this); selected(this);"
onblur="restoreValue(this); deselected(this);"/>
        <input id="searchBtn" class="redPurple floatAesthetic" type="submit"/>
      </div>
      <input name="page" value="tool" type="hidden"/>
    </form><br><br><br><br>'); # Print the rest of the form. $search holds the default value of
the search term input.
    echo('`;</script>'); # End the JavaScript script.
  }
?>
```

php/onload/tool.php

The code in this file is executed once `index.php` loads. It deals with loading the tool page.

```
<?php
  if(isset($_COOKIE["user"])){ # Only display the page if the user has signed in.
    # For searching through forum posts and sequences. selected="selected" is used instead of
just selected as it is XHTML compliant. This also indicated the default option of select.
    $search = isset($trim["searchValue"]) ? $trim["searchValue"] : "Search"; # Get the
previously entered value searched for if the user has searched before. Otherwise set toa
defualt value.
    $searchType = isset($trim["searchType"]) ? $trim["searchType"] : "";
    echo('<script>document.getElementById("searchFormCon").innerHTML=`'); # Start the JavaScript
script.
    echo('<form id="searchBoxForm" action="index.php?page=search" method="post">
      <div id="inputsCon">
        <select id="searchType" name="searchType" class="redPurple floatAesthetic">'); # Begin
displaying the search form.

    $optionKeyValues = ['currentUserSequences', 'My sequences', 'currentUserDiseases', 'My
```

```
diseases', 'currentUserThreads', 'My threads', 'sequences', 'All sequences', 'diseases', 'All diseases', 'threads', 'All theads'];
    # Above line creates an array that holds the value (the key) and the actual text (the value) displayed to the user, for each user in the respective format. E.g. w, x, y, z where w is a value and x is a peice of text and w,x and y,z are pairs of values.
    $optionDictionary = new dictionary($optionKeyValues); # Create a dictionary object using the constructor defined in dictionary.php and the array previosuly defined.
    if($searchType != ""){ # If the user has entered a search term before, display that as the default type of search to be conducted.
      $submittedValue = $optionDictionary->read($searchType); # Gets the text to display to the user.
      echo('<option value="'.$searchType.'" selected="selected">'.$submittedValue.'</option>'); # Prints the option.
      $result = $optionDictionary->remove($searchType); # Removes the option from the dictionary so it is not reprinted.
    }
    $optionKeys = $optionDictionary->keys; # Fetches all keys from the dictionary.
    for($i=0;$i<count($optionKeys);$i++){ # For each key.
      $currentKey = $optionKeys[$i];
      $currentValue = $optionDictionary->read($currentKey); # Fetch the value associated with that key.
      echo('<option value="'.$currentKey.'">'.$currentValue.'</option>'); # Print the option.
    }
    echo('</select>
        <input id="searchBox" name="searchValue" class="redPurple floatAesthetic" type="text" value="'.$search.'" info="Search" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/>
        <input id="searchBtn" class="redPurple floatAesthetic" type="submit"/>
      </div>
      <input name="page" value="tool" type="hidden"/>
    </form><br><br><br><br>'); # Print the rest of the form. $search holds the default value of the search term input.
    echo('`;</script>'); # End the JavaScript script.
  }
?>
```

php/onload/toolsingleview.php

The code here executes once `index.php` is loaded. It deals with loading the tool single view page.

```
<?php
  if(isset($_GET["sequence"])){ # If we are retriving a specific sequence.
    echo("<script>document.getElementById('toolSingleViewPageContent').innerHTML='");
    $nsid = $_GET["sequence"];
    $row = mysqli_fetch_array(mysqli_query($db, "SELECT name, notes, uid FROM nucelosomesequence WHERE nsid=$nsid"));
    $name = $row[0]; # Fetch the sequence name.
    $notes = $row[1];
    $ownerid = $row[2];
    echo('<p class="sequenceDiseaseTitle">'.$name.'</p><br><hr>'); # And display it.


    $r = mysqli_query($db, "SELECT histoneMods, ndsid, nid FROM nucleosome WHERE nsid=$nsid
```

```
ORDER BY nsid"); # Fetch data on all nucleosome related to this sequence.
    $counter = 0; # Defines the nucleosome number (to give it an arbitary name). Orders from
first to last (ascending by default).
    $num = mysqli_num_rows($r); # Length of the queue.
    $nucleosomesHTML = new queue($num); # Contains a queue of nucleosome HTML blocks.
    $allComponents = new dictionary(["allDNA", "", "allMods", ""]);
    $nucleosomes = []; # Two dimensional array that will contain the name number and the
nucelosome ID for each nucleosome. Name Number (counter) : nucleosome id.
    while($row = mysqli_fetch_array($r)){
      $nid = $row[2];
      $html = '<div class="strip redPurple floatAesthetic">';
      $counter++;
      $html .= '<p class="text"><b>Nucleosome '.$counter.'</b><p><br><hr>'; # Print the next
nucleosome number in the sequence.
      $dnaSequence = mysqli_fetch_array(mysqli_query($db, "SELECT DNASequence FROM
nucleosomednasequence WHERE ndsid=".$row[1]))[0];
      $allDNA = $allComponents->read("allDNA").$dnaSequence;
      $errorCode = $allComponents->editValue("allDNA", $allDNA);
      $html .= '<p class="text"><em>DNA Sequence:</em><br>'.$dnaSequence.'</p><p class="text">
<em>Histone Mods:</em><br>';
      $allMods = $allComponents->read("allMods").$row[0];
      $errorCode = $allComponents->editValue("allMods", $allMods);
      $mods = explode(",", $row[0]);
      for($i=0;$i<count($mods);$i++){
        $mod = (int)$mods[$i]; # Select the current histone modification ID (hmid).
        $modName = mysqli_fetch_array(mysqli_query($db, "SELECT name FROM histonemods WHERE
hmid=".$mod))[0]; # Select the histone modification name
        $final = $modName;
        $value = $i <=> (count($mods)-2); # Returns -1 if $i is less than count($mods)-1, 0 if
equal two or 1 if greater than. This is called the spaceship operator.
        if($value == -1){ # Concatenate a comma if this ISN'T the last mod in the sequence.
          $final .= " , ";
        }
        $html .= $final;
      }
      if($uid == $ownerid){ # Checks to see if the user owns this sequence.
        $html .= '<form action="index.php?page=tool&sequence='.$nsid.'" method="post"><input
name="nidTD" type="hidden" value="'.$nid.'"/><input name="submitTD" class="removeDeleteButton"
type="submit" value="Delete" onfocus="selected(this);" onblur="deselected(this);"/></form>';
        # Displays the form for deleting a nucleosome from the sequence if the user owns this
sequence.
      }
      $html .= '</div><br><br>';
      $nucleosomesHTML->append($html);
      array_push($nucleosomes, [$counter, $nid]); # Append the array.
    }
    $r = mysqli_query($db, "SELECT disease.did, disease.name FROM disease INNER JOIN
nucelosomesequence ON disease.did=nucelosomesequence.did WHERE nsid=$nsid"); # Selects the
disease associated with the sequence if such a relationship exists.
    $row = mysqli_fetch_array($r);
    $did = $row[0]; # Fetch the disease ID.
    $diseaseName = "";

    if($r){ # If the sequence is related to a disease, fetch the disease's name.
```

```
      $diseaseName = $row[1];
    }
    $allMods = explode(",", $allComponents->read("allMods"));
    $result = interpretHistoneModSequence($allMods, $db); # Determine the change in expression
of the DNA sequence at hand.
    if($result > 0){ # If the overall change in expression activates the expression, display
this to the user using the + sign.
      $result = "+".$result;
    }

    if($r){ # Only display a disease association of one exists.
      echo('<div class="strip greenYellow floatAesthetic"
onclick="window.location.href=`index.php?page=tool&disease='.$did.'`"><p class="text">This
sequence is derived from: <b>'.$diseaseName.'</b></p></div><br><br>');
    }
    echo('<div class="strip greenYellow floatAesthetic"><p class="text"><b>Overall</b></p><p
class="text"><em>Full DNA Sequence: <br></em>'.$allComponents->read("allDNA").'</p><p
class="text"><em>Expression Change Result: <br></em>'.$result.'</p><br><p class="text">
<em>Notes: </em><br>'.$notes.'</p></div><br><br>');
    # This must be printed first before everthing else so that it stays at the top of the page.
It displays data specific to the overall sequence.
    # === Now print the breakdown of the sequence.
    do {
      echo($nucleosomesHTML->get());
      $nucleosomesHTML->pop();
    } while($nucleosomesHTML->isEmpty() == FALSE);
    # ===
    if($uid == mysqli_fetch_array(mysqli_query($db, "SELECT uid FROM nucelosomesequence WHERE
nsid=$nsid"))[0] || $admin == TRUE){ # Editing for owner and admin only.
      if($notes == ""){
        $notes = isset($trim["notesTEO"]) ? $trim["notesTEO"] : "Notes"; # Determine the default
value of the notes using the tenary operator.
      }
      echo ('<form id="toolForm" class="orangeBlue floatingAesthetic" action="index.php?
page=tool&sequence='.$nsid.'" method="post"><p class="text">Edit Overall Details</p><br><br>
<select name="diseaseAssociationTEO">'); # Start displaying the edit form for the overall
sequence.
      if($did){ # If the disease-sequence relationship exists, set that to be the first option
in the drop down input.
        echo('<option selected="selected" value="'.$did.'">'.$diseaseName.'</option>');
        $q = "SELECT did, name FROM disease WHERE did!=$did"; # Then fetch all diseases that are
not the disease just displayed.
      }
      else { # Print all diseases in alphabetical order.
        $q = "SELECT did, name FROM disease ORDER BY name DESC";
      }
      global $q; # Get the query we just determiend.
      echo('<option value="NULL">No association</option>'); # Display No association as the
second/first option depending on whether a disease association already existed or not.
      $r = mysqli_query($db, $q);



      while($row = mysqli_fetch_array($r)){ # Print each remaining/all diseases as options for
```

```
the disease association of the sequence depending on whether a disease association already
existed or not.
        echo('<option value="'.$row[0].'">'.$row[1].'</option>');
      }
      echo('</select><textarea name="notesTEO" info="Notes" onfocus="clearValue(this);
selected(this);" onblur="restoreValue(this); deselected(this);">'.$notes.'</textarea>');
      echo('<input name="nameTEO" type="text" value="'.$name.'" info="Name"
onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/>');
      echo('<input type="hidden" name="submittedTEO" value="TRUE"/><input type="submit""
name="submitTEO" value="Edit" onfocus="selected(this);" onblur="deselected(this);"/></form>');
      # Display the rest of the form with the deault values determined earlier.
      echo('<form id="toolForm" class="orangeBlue floatingAesthetic" action="index.php?
page=tool&sequence='.$nsid.'" method="post"><p class="text">Edit</p><br><br><select
name="nucleosomeTE" onfocus="selected(this);" onblur="deselected(this);"><option
value="NULL">New nucleosome</option>');
      # Begin displaying the nucleosome-specific edit form.
      foreach($nucleosomes as $n){ # Print each nuclesome as an option to edit. Before we
printed an option to create a new nucleosome.
        echo('<option value="'.$n[1].'"> Nucleosome '.$n[0].'</option>');
      }
      $dnaTE = isset($trim["dnaSequenceTE"]) ? $trim["dnaSequenceTE"] : "ATCG";
      $modsStrTE = isset($trim["histoneModsTE"]) ? $trim["histoneModsTE"] : "";
      $viewingModsTE = "";
      $mods = explode(",", $modsStrTE);
      foreach($mods as $mod){
        $name = mysqli_fetch_array(mysqli_query($db, "SELECT name FROM histonemods WHERE
hmid=".intval($mod)))[0];
        $viewingModsTE .= $name." ";
      }
      echo('</select><textarea id="dnaSequenceTE" name="dnaSequenceTE" maxlength="127"
info="ATCG" onkeydown="dnaInputCheck(this);" onfocus="clearValue(this); selected(this);"
onblur="restoreValue(this); deselected(this);">'.$dnaTE.'</textarea><br><br><div id="hmodBg">
<div id="hmodSequenceTE" class="toolCon">'.$viewingModsTE.'</div></div><br><br>');
      echo('<div id="histoneModDiv"><div class="toolCon"><div id="removeHistoneBtnTE"
class="button histoneMod" onclick="removeLastHmod(`hmodSequenceTE`,
`histoneModsTE`);">Remove</div>');
              $q = "SELECT hmid, name FROM histonemods";
              $r = mysqli_query($db, $q);
              while($row = mysqli_fetch_array($r)){
                echo('<div class="button histoneMod" onclick="addHmod(`'.$row[1].'`,
`'.$row[0].'`, `hmodSequenceTE`, `histoneModsTE`);">'.$row[1].'</div>');
              }
      echo('</div></div><input id="histoneModsTE" name="histoneModsTE" type="hidden"
value="'.$modsStrTE.'"/>');
      echo('<input type="submit" name="submitTE" value="Edit"/><input type="hidden"
name="submittedTE" value="TRUE"/>');
    }
    echo("';</script>");
  }
  else if(isset($_GET["disease"])){ # If we are retriving the data on a specific disease.
    echo("<script>document.getElementById('toolSingleViewPageContent').innerHTML='");
    $did = $_GET["disease"]; # Fetch the ID of the disease from the URL query.

    $row = mysqli_fetch_array(mysqli_query($db, "SELECT name, notes, uid FROM disease WHERE
```

```
did=$did"));
    $name = $row[0];
    $notes = $row[1];
    if(is_null($notes)){
      $notes = isset($trim["notesTEO"]) ? $trim["notesTEO"] : "Notes";
    }
    $ownerid = $row[2];
    echo('<br><div class="strip greenYellow floatAesthetic"><p class="text"><b>'.$name.'</b></p>
<br><p>'.$notes.'</p></div><br><br>');
    # Thhe above line displays the overview information on the disease.
    $r = mysqli_query($db, "SELECT nsid, name, notes FROM nucelosomesequence WHERE did=$did"); #
Fetches all seqeunces associated with this disease.
    while($row = mysqli_fetch_array($r)){ # Ieterates through each sequence associated with this
disease.
      $nsid = $row[0];
      echo('<div class="strip redPurple floatAesthetic"
onclick="window.location.href=`index.php?page=tool&sequence='.$nsid.'`;"><p
class="text">'.$row[1].'</p><br><p class="text">'.$row[2].'</p>');
      # Above line displays the currnent sequence.
      if($uid == $ownerid){ # Checks to see if the user owns this sequence.
        echo('<form action="index.php?page=tool&disease='.$did.'" method="post"><input
name="nsidTD" type="hidden" value="'.$nsid.'"/><input name="submitTD" class="removeDeleteButton"
type="submit" value="Remove" onfocus="selected(this);" onblur="deselected(this);"/></form>');
        # Above line displays a form to remove association of the sequence to the disease.
      }
      echo('</div><br><br>');
    }

    if($uid == $ownerid){ # Editing for owner only.
      # This displays the form for editing the disease, usign default values determined earlier.
      echo ('<form id="toolForm" class="orangeBlue floatingAesthetic" action="index.php?
page=tool&disease='.$did.'" method="post"><p class="text">Edit Overall Details</p><br><br>');
      echo('<textarea name="notesTEO" info="Notes" onfocus="clearValue(this); selected(this);"
onblur="restoreValue(this); deselected(this);">'.$notes.'</textarea>');
      echo('<input name="nameTEO" type="text" value="'.$name.'" info="Name"
onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/>');
      echo('<input type="hidden" name="submittedTEO" value="TRUE"/><input type="submit""
name="submitTEO" value="Edit" onfocus="selected(this);" onblur="deselected(this);"/></form>');
    }
    echo("';</script>");
  }
?>
```

php/onload/onloadLOAD.php

This file groups together all of the other file in the **/onload** directory and loads them within this file. This file can then be loaded itself into `index.php`. This simplifies the code in `index.php` so that it can be more easily maintained.

```
<?php


  # Require_once makes sure that only one version of the file is included and prevents the
```

```
execution of the rest of this script if it is not present - preventing a user from running an
errornous version of this program.
  require_once("account.php"); # Always load the account page PHP script.
  if(isset($_COOKIE["user"])){ # Only load these PHP scripts if the user is signed in, so that
only activated (therefore valid) accounts have access to the main website, so that actions the
user does can be attributed to the account.
    require_once("forum.php");
    require_once("tool.php");
    require_once("toolSingleView.php");
    require_once("search.php");
  }
  echo("<script>"); # Start the JavaScript script.
  if(isset($_GET["page"])){ # If the user is being redirected to a particular page.
    if(isset($_COOKIE["user"])){
      switch($_GET["page"]){ # Checks the value of the page URL query.
        case "account": # If this value is account.
          echo("menuBtnClick('account');"); # Redirect to the account page.
          break; # Break out of the switch statement.
        case "forum":
          if(isset($_GET["thread"])){ # If we are displaying a particular thread then we need to
pop up the single view page.
            echo("menuBtnClick('forumSingleView');");
          }
          else{
            echo("menuBtnClick('forum');");
          }
          break;
        case "help":
          echo("menuBtnClick('help');");
          break;
        case "news":
          echo("menuBtnClick('news');");
          break;
        case "tool":
          if(isset($_GET["sequence"]) || isset($_GET["disease"])){ # Same goes for histone
modification sequences. A disease will contain many smaller sequences, and this information
should be displayed on one page, so we need to display on the single view page.
            echo("menuBtnClick('toolSingleView');");
          }
          else{
            echo("menuBtnClick('tool');");
          }
          break;
        case "search":
          echo("menuBtnClick('search');");
          break;
        default: # If there is an invalid value, redirect to the home page.
          echo("menuBtnClick('home');");
          break;
      }
    }


    else if($_GET["page"] == "account"){ # Let the user switch between the account and home
```

```
pages via the URL query, but nothing else.
      echo("menuBtnClick('account');");
    }
    else {
      echo("menuBtnClick('home');");
    }
  }
  echo("</script>"); # Ends the JavaScript script.
?>
```

php/submission/account.php

The code here executes once a form from the account page is submitted.

```
<?php
  if(isset($trim["submittedSU"])){ # If the user wants to sign up a new account.
    $fn = strip_tags($trim["firstNameSU"]); # Strip tags removes any HTMl tags, preventing XSS
attacks. $trim is where all posted data is located.
    $sn = strip_tags($trim["secondNameSU"]); # We use names of inputs to extract data specific
to a given input. Eg, $trim["x"] extracts data from the input with name x.
    $ea = strip_tags($trim["emailAddressSU"]);
    $pw = $trim["passwordConSU"];
    $in = $trim["interestSU"];
    $hash = hashing32($pw); # Uses my hash algorithm to produce a 32 character string.
    $errors = [];
    if(mysqli_num_rows(mysqli_query($db, "SELECT * FROM users WHERE emailAddress='$ea'")) > 0){
# If the email address submitted is not unqiue.
      $errors = ["Email addresss already in use."]; # Throw an error.
    }

    if($pw == "Password"){
      $errors = ["Password cannot be default."];
    }

    if($in == "Interest"){
      $errors = ["Please select and interest."];
    }

    if(empty($errors)){ # If the array contaisn no errors.
      $q = "INSERT INTO users VALUES(NULL, '$fn', '$sn', '$ea', '".crypt($pw, 'iN5@n1tY')."',
'$in', '$hash', 0)"; # Create a new account, that is unactivated as it contains a hash value.
      # In the above line, crypt() is a hash algorithm that uses a salt to encrypt the password.
      $r = mysqli_query($db, $q);
      mail($ea, "Nomios - Please Confirm Your Account", "Confirm your account by clicking copy
and pasting the following url: http://127.0.0.1/index.php?page=account&email=$ea&hash=$hash",
"FROM: reiss1999@gmail.com");
      # Above line sends and email to the user's address.
      echo($popupTop);
      echo("<p>Take a look at your inbox for a confirmation before you sign in.</p>");
      echo($popupBottom);
      # Above three lines indicate to the user that they must activate their account.

    }
```

```
    else { # If there are errors, display them to the user so that they can correct them.
      echo($popupTop);
      foreach($errors as $error){
        echo("<p>".$error."</p>");
      }
      echo($popupBottom);
    }
  }
  else if(isset($trim["submittedSI"])){ # If the user is trying to sign in.
    $ea = $trim["emailAddressSI"];
    $pw = $trim["passwordSI"];
    $q = "SELECT uid FROM users WHERE emailAddress='$ea' AND password='".crypt($pw,
'iN5@n1tY')."'"; # Attempt to fetch the uid of a user that mathces the mail address and
encrypted password entered.
    $r = mysqli_query($db, $q);
    if(mysqli_num_rows($r) == 1){ # If there is a one, unique result then it means that the
account exists and is valid (a valid account is one where only one copy of it exists because
malicous attakcs could bemade through duplicate accounts).
      if(mysqli_num_rows(mysqli_query($db, "SELECT uid FROM users WHERE emailAddress='$ea' AND
hash IS NULL")) == 1){ # If the user's account ahs been activated.
        echo('<script>document.cookie = "user='.mysqli_fetch_array($r)[0].';expires='.(time()+
(3600*24)*30).'"; reload();</script>'); # Create a cookie storing the user ID using JavaScript
and refresh the page to prevent form resubmission.
      }
      else { # Otherwise, the user's account is not activated.
        echo($popupTop);
        echo("<p>The account has not yet been confirmed. Please check your inbox.</p>");
        echo($popupBottom);
      }
    }
    else { # No account mathces the encrypted-password and email address combination.
      echo($popupTop);
      echo("<p>Either the account does not exist or the password was incorrect.</p>");
      echo($popupBottom);
    }
  }
  else if(isset($trim["submittedAE"])){ # If the user is editing their password
    $pw = $trim["passwordAE"];
    $pc = $trim["passwordConAE"]; # Password confirmation
    $pwc = crypt($pw, 'iN5@n1tY'); # Encrypt the password
    $errors = [];
    if($pw !== $pc){ # !== not same type or same value.
      $errors = ["Passwords do not match."];
    }

    if($pw == "Password" || $pwc == "password" || $pwc == mysqli_fetch_array(mysqli_query($db,
"SELECT password FROM users WHERE uid = ".$_COOKIE["user"][0]))){
      # If the password has already been entered or it was left at its default value throw an
error. Don't need to check $pw to see if it matches the already entered password as if it
mathces $pwc, then $pwc will throw an error for it.
      $errors = ["Password is invalid."];
    }
```

```
    if(empty($errors)){ # If no errors are found.
      $q = "UPDATE users SET password = '".$pwc."' WHERE uid = ".$_COOKIE["user"];
      $r = mysqli_query($db, $q); # Update the password.
      echo($popupTop);
      echo("<p>Password updated.</p>");
      echo($popupBottom);
    }
    else {
      echo($popupTop);
      foreach($errors as $error){
        echo("<p>".$error."</p>");
      }
      echo($popupBottom);
    }
  }
?>
```

php/submission/deletingScripts.php

This code is executed when a delete HTML form is submitted from the tool/tool single vie pages.

```
<?php
  if(isset($trim["deleteNsid"])){
    $nsid = $trim["deleteNsid"]; # Fetch ID for the nucleosome sequence from the form.
    $r = mysqli_query($db, "DELETE FROM nucelosomesequence WHERE nsid=$nsid"); # Delete the row
that contains this id from the database
    echo('<script>window.location.href="index.php?page=search";</script>'); # Redirect to the
search page.
  }
  else if(isset($trim["deleteDid"])){
    $did = $trim["deleteDid"]; # Fetch ID for the disease from the form.
    $r = mysqli_query($db, "DELETE FROM disease WHERE did=$did"); # Delete this disease row from
the database using the ID.
    echo('<script>window.location.href="index.php?page=search";</script>');
  }
  else if(isset($trim["nidTD"]) && isset($_GET["sequence"])){
    $nid = $trim["nidTD"];
    $r = mysqli_query($db, "DELETE FROM nucleosome WHERE nid=$nid");
    echo('<script>window.location.href="index.php?page=tool&sequence='.$nsid.'";</script>');
    # Redirect to the sequence from which the nucelosome was deleted from to prevent accidental
resubmission.
  }

  # All forum deleting stuff is handled in submission/forum.php
?>
```

php/submission/forum.php

This code is executed when a HTML form from the forum is submitted.

```
<?php
```

```
  # Deleting forum items
  if(isset($trim["deleteMid"])){
    # $trim["deleteMid"] holds the ID for the message.
    $r = mysqli_query($db, "DELETE FROM message WHERE mid=".$trim['deleteMid']); # Delete the
message we want to delete.
    $r = mysqli_query($db, "DELETE FROM replies WHERE mid=".$trim['deleteMid']); # Delete all
attached replies to that message.
    echo('<script>window.location.href = "index.php?'.$_SERVER['QUERY_STRING'].'";</script>'); #
Redirected to the same thread on the forum single view page.
  }
  else if(isset($trim["deleteTid"])){ # Delete an entire thread.
    $r = mysqli_query($db, "DELETE FROM thread WHERE tid=".$trim['deleteTid']); # Delete the
thread.
    $r = mysqli_query($db, "SELECT mid FROM message WHERE tid=".$trim['deleteTid']); # Select
all messages in this thread.
    while($row = mysqli_fetch_array($r)){ # For each message
      $mid = $row[0]; # Fetch the message ID for deleting the replies associated with each
message because the replies do not contain a tid column.
      $r = mysqli_query($db, "DELETE FROM message WHERE mid=".$mid); # Delete the message using
this ID.
      $r = mysqli_query($db, "DELETE FROM replies WHERE mid=".$mid); # Delete replies associated
with this message.
    }
    echo('<script>window.location.href = "index.php?page=forum";</script>'); # Redirect to the
forum to view all threads.
  }
  else if(isset($trim["deleteRid"])){ # Delete a reply.
    $r = mysqli_query($db, "DELETE FROM replies WHERE rid=".$trim['deleteRid']);
    echo('<script>window.location.href = "index.php?'.$_SERVER['QUERY_STRING'].'";</script>'); #
Redirected to the same thread on the forum single view page.
  }
  # ===

  if(isset($trim["submittedPT"])){ # If the user is posting a new thread.
    $s = strip_tags($trim["subjectPT"]);
    $m = strip_tags($trim["messagePT"]);
    $errors = [];
    if($s == "Subject" || $m == "Message"){
      $errors = ["Cannot post default values."];
    }
    if(empty($errors)){ # If there are no errors.
      $q = "INSERT INTO thread VALUES(0, $uid, '$s', '$m', CURRENT_TIMESTAMP)"; # Add a new row
to the thread table. CURRENT_TIMESTAMP gets the current time and date of the server.
      $r = mysqli_query($db, $q);
      $tid  = mysqli_fetch_array(mysqli_query($db, "SELECT tid FROM thread ORDER BY tid DESC
LIMIT 1"))[0]; # Fetch the ID of the latest thread and therefore, the one we just submitted.
      echo('<script>window.location.href = "index.php?page=forum&thread='.$tid.'";</script>'); #
Redirect to the singleViewPage to view on this particular thread.
    }
    else {
      echo($popupTop);
      foreach($errors as $error){

        echo("<p>".$error."</p>");
```

```
      }
      echo($popupBottom);
    }
  }
  else if(isset($trim["submittedPM"])){ # If the user is posting a message to a thread.
    $m = strip_tags($trim["messagePM"]);
    $tid = $_GET["thread"]; # Get the id of the thread we are currently viewing (must be viewing
a thread on a singleViewPage to post a message to it).
    $errors = [];
    if($m == "Message"){
      $errors = ["Cannot post a default value."];
    }

    if(empty($errors)){
      $r = mysqli_query($db, "INSERT INTO message VALUES(0, $tid, $uid, '$m',
CURRENT_TIMESTAMP)"); # Insert a new row to the message table.
      echo('<script>window.location.href = "index.php?page=forum&thread='.$tid.'";</script>');#
Reload the page to prevent resubmission.
    }
    else {
      echo($popupTop);
      foreach($errors as $error){
        echo("<p>".$error."</p>");
      }
      echo($popupBottom);
    }
  }
  else if(isset($trim["midPR"])){ # If the user is posting a reply to a message.
    $m = strip_tags($trim["messagePR"]);
    $tid = $_GET["thread"];
    $errors = [];
    if($m == "Message"){
      $errors = ["Cannot post a default value."];
    }

    if(empty($errors)){
      echo('<script>window.location.href = "index.php?page=forum&thread='.$tid.'";</script>'); #
Reload the page to prevent resubmission.
      $r = mysqli_query($db, "INSERT INTO replies VALUES(NULL, ".$trim['midPR']."., $uid, '$m',
CURRENT_TIMESTAMP)"); # Insert a new row to the replies table.
    }
    else {
      echo($popupTop);
      foreach($errors as $error){
        echo("<p>".$error."</p>");
      }
      echo($popupBottom);
    }
  }
?>
```

php/submission/search.php

This code is executed when one the search page form is submitted.

```
<?php
  if(isset($trim["searchValue"]) && isset($trim["searchType"])){
    $searchTerm = $trim["searchValue"]; # This contains the text that the user is actually
searching for.
    $searchType = $trim["searchType"]; # This tells us what filter to apply to the search.
    if($searchTerm == "Search"){ # We cannot search for the text 'Search'.
      $searchTerm = ""; # This sets the value so that it lists all records.
    }

    $q = "";
    switch($searchType){ # This is used to determine which query to process based on the filter.
% is a wildcard character.
      case "threads":
        $q = "SELECT tid, subject, firstName FROM thread INNER JOIN users ON
thread.uid=users.uid WHERE subject LIKE '%$searchTerm%' OR message LIKE '%".$searchTerm."%'
ORDER BY tid DESC";
        break;
      case "sequences":
        $q = "SELECT nsid, name, firstName FROM nucelosomesequence INNER JOIN users ON
nucelosomesequence.uid=users.uid WHERE name LIKE '%".$searchTerm."%'";
        break;
      case "diseases":
        $q = "SELECT * FROM disease WHERE name LIKE '%".$searchTerm."%'";
        break;
      case "currentUserSequences":
        $q = "SELECT nsid, name, firstName FROM nucelosomesequence INNER JOIN users ON
nucelosomesequence.uid=users.uid WHERE users.uid=$uid AND name LIKE '%".$searchTerm."%'";
        break;
      case "currentUserThreads":
        $q = "SELECT tid, subject, firstName FROM thread INNER JOIN users ON
thread.uid=users.uid WHERE subject LIKE '%$searchTerm%' OR message LIKE '%".$searchTerm."%' AND
users.uid=".$uid." ORDER BY tid DESC";
        break;
      case "currentUserDiseases":
        $q = "SELECT * FROM disease WHERE name LIKE '%".$searchTerm."%' AND uid=".$uid;
        break;
      default:
        $errors = ["No search type indicated."];
        break;
    } # All queries will return records created by the user. The 'currentUser' queries return
only records created by the current user.
    $r = mysqli_query($db, $q);
    echo("<script>document.getElementById('searchResultsCon').innerHTML=\"");
    if(mysqli_num_rows($r) > 0){ # Only display results if any rows exist from the query.
      while($row = mysqli_fetch_array($r)){
        if($searchType == "threads" || $searchType == "currentUserThreads"){ # Searching for
threads.
          $s = $row[1];
          if(strlen($s) > 25){
            $s = substr($s, 0, 25)."...";

          }
```

```
          # This displays the results of the retrived records, when one is clicked it will
redirect to a page displaying all of the records='s information.'
          echo("<div class='strip greenYellow floatAesthetic' onclick='window.location.href =
`index.php?page=forum&thread=".$row[0]."`;'><a>$s</a><a style='float: right;'> | ".$row[2]."
</a>"); # Each thread displayed.
          if($admin == TRUE){ # Allows admins to delete any thread.
            echo("<form action='".$_SERVER["REQUEST_URI"]."' method='post'><input
name='deleteTid' type='hidden' value='".$row[0]."'/><input class='removeDeleteButton'
type='submit' value='Delete'/></b></form>");
          }
          echo("</div><br>");
        }
        else if($searchType == "sequences" || $searchType == "currentUserSequences"){ #
Searching for histone modification sequences.
          # This displays the results of the retrived records, when one is clicked it will
redirect to a page displaying all of the records='s information.'
          echo("<div class='strip greenYellow floatAesthetic' onclick='window.location.href =
`index.php?page=tool&sequence=".$row[0]."`;'><a>$row[1]</a><a style='float: right;'> |
".$row[2]."</a>"); # Each thread displayed.
          if($uid == mysqli_fetch_array(mysqli_query($db, "SELECT uid FROM nucelosomesequence
WHERE nsid=".$row[0]))[0]){ # Allows the user to delete their own sequences but no-one elses.
            echo("<form action='".$_SERVER["REQUEST_URI"]."' method='post'><input
name='deleteNsid' type='hidden' value='".$row[0]."'/><input class='removeDeleteButton'
type='submit' value='Delete'/></b></form>");
          }
          echo("</div><br>");
        }
        else { # If we are searching diseases. Else can only be this as any other value of
$searchType is an input error.
          echo("<div class='strip greenYellow floatAesthetic' onclick='window.location.href =
`index.php?page=tool&disease=".$row[0]."`;'><a>".$row[1]."</a></a>"); //Each thread displayed.
          if($uid == mysqli_fetch_array(mysqli_query($db, "SELECT uid FROM disease WHERE
did=".$row[0]))[0]){ # Allows the owner to delete their own diseases.
            echo("<form action='".$_SERVER["REQUEST_URI"]."' method='post'><input
name='deleteDid' type='hidden' value='".$row[0]."'/><input class='removeDeleteButton'
type='submit' value='Delete'/></b></form>");
          }
          echo("</div><br>");
        }
      }
    }
    else { # If no results display a message to indicate this.
      echo("No results.");
    }
    echo("\";</script>");
  }
?>
```

php/submission/tool.php

This code is executed when one of the HTML forms from the tool page are submitted.

```php
<?php
  if(isset($trim["submittedD"])){
    $name = strip_tags($trim["diseaseNameD"]);
    $notes = checkNotes(strip_tags($trim["notesD"])); # Sets notes to be SQL compliant by encapsulating them with quotation marks if not default value and if a default value set it to be NULL.
    $errors = [];
    if($name == "Disease name"){
      $errors = ["Name cannot be default value"];
    }
    else { # This checks to see if the disease is already in the database. One or the other as 'Disease Name' cannot be submitted as a value so it wont be in the database.
      $r = mysqli_query($db, "SELECT name FROM disease"); # Selects all of the name in the disease table.
      while($row = mysqli_fetch_array($r)){ # For each row in the table.
        if($row[0] == $name){ # Checks if the user-inputted name is the same as the current row.
          $errors = ["Disease already in the database"];
        }
      }
    }

    if(empty($errors)){
      $q = "INSERT INTO disease VALUES(NULL, '$name', $notes, $uid)"; # Insert a new row into the disease table.
      $r = mysqli_query($db, $q);
      echo("<script>window.location.href = 'index.php?page=search&diseaseSubmitted=TRUE';</script>"); # Reload the page to prevent resubmission.
    }
    else {
      echo($popupTop);
      foreach($errors as $error){
        echo('<p>'.$error.'</p>');
      }
      echo($popupBottom);
    }
  }
  if(isset($trim["submittedT"])){ # This is for submitting a new sequence.
    $dnaStr = strtoupper($trim["dnaSequenceT"]); # Convert text to be capitalised (it is not changed by CSS).
    $modsStr = $trim["histoneModsT"];
    $name = strip_tags($trim["sequenceNameT"]);
    $notes = checkNotes(strip_tags($trim["notesT"]));
    $diseaseAssoc = $trim["diseaseAssociationT"];
    $errors = [];
    if($modsStr == "" || $dnaStr == "ATCG" || $name == "Name"){
      $errors = ["Cannot use default values."];
    }
    $r = mysqli_query($db, "SELECT name FROM nucelosomesequence WHERE uid=$uid"); # Select the names from all nucleosomesequences.
    while($row = mysqli_fetch_array($r)){ # For each returned row
      if($row[0] == $name){ # Check if the name matches with name of a sequence that the user already owns.
        $errors = ["You already have a sequence with that name."];
```

```
          break;
        }
      }
      if(empty($errors)){
        $dnaStrs = str_split($dnaStr, 127); # Produces nucleosome-sized chunks of DNA.
        $modsStrs = explode("|", $modsStr); # Produces a chunk of histone modifications for each
specified nucleosome.
        $did = "";
        if($diseaseAssoc != "NULL"){
          $did = (int)$diseaseAssoc; # Cast the string value of the disease association as an
integer.
        }
        else {
          $did = "NULL";
        }
        $dnaStrsLen = count($dnaStrs); # Length of the $dnaStrs array.
        $modsStrsLen = count($modsStrs); # Length of the $modsStrs array.
        $difference = $dnaStrsLen - $modsStrsLen; # Difference between the two.
        if($dnaStrsLen > $modsStrsLen){ # If we have more DNA chunks than histone modifications
          for($i=0;$i<abs($difference);$i++){ # Difference between the two values (abs() returns
the magnitude of a number).
            array_push($modsStrs, " "); # Append spaces so that each nucleosome has "mods" so that
it can be inserted into the database.
          }
        }
        else if($dnaStrsLen < $modsStrsLen){ # If we have more histone chunks than DNA chunks, add
a new DNA chunk of unknown bases.
          for($i=0;$i<abs($difference);$i++){ # Difference between the two values (abs() returns
the magnitude of a number).
            array_push($dnaStrs,
"NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN"); # Append unknown DNA sequence N signifies unknown, 127 Ns in
each append.
          }
        }
        $queries = []; # Array that holds all of our nucleosome queries ONLY.
        $r = mysqli_query($db, "INSERT INTO nucelosomesequence VALUES(NULL, $uid, $did,
'".$name."', $notes)"); # Inserts the nucleosome sequence into its table.
        $nsid = mysqli_fetch_array(mysqli_query($db, "SELECT nsid FROM nucelosomesequence WHERE
uid=$uid AND name='".$name."'"))[0]; # Fetches the ID of this new record.
        for($i=0;$i<count($dnaStrs);$i++){ # For each nucleosome (defined by number of nucleosome
DNA chunks).
          $dna = $dnaStrs[$i]; # Current DNA.
          $mods = $modsStrs[$i]; # Current histone mods.
          $ndsid = -1; # Initialise the ID of the nucleosome DNA sequence.
          $dnaFound = checkDNA($dna); # Check to see if DNA is already in the database.
          if($dnaFound == TRUE){
            $ndsid = mysqli_fetch_array(mysqli_query($db, "SELECT ndsid FROM nucleosomednasequence
WHERE DNASequence='".$dna."'"))[0]; # Fetch the ndsid of the mathcing DNA sequence.
          }
          else{ # If no DNA match is found


            $r = mysqli_query($db, "INSERT INTO nucleosomednasequence VALUES(NULL, '".$dna."')");
```

```
# Insert a new DNA sequencn into the databse.
          $r = mysqli_query($db, "SELECT ndsid FROM nucleosomednasequence ORDER BY ndsid DESC
LIMIT 1");
          $ndsid = mysqli_fetch_array($r)[0]; # Fetches the id of the dna sequence we just
inserted.
        }
        array_push($queries, "INSERT INTO nucleosome VALUES(NULL, $ndsid, '".$mods."', $nsid)");
# Adds a query to the queries array that inserts a new record to the nucelosome.
      }
      foreach($queries as $query){ # Execute all nucleosome insertion queries.
        $r = mysqli_query($db, $query);
      }
      #echo('<script>window.location.href="index.php?page=search&sequenceSubmitted=TRUE";
</script>'); # Redirect so that the user does not resubmit the data.
    }
    else{
      echo($popupTop);
      foreach($errors as $error){
        echo('<p>'.$error.'</p>');
      }
      echo($popupBottom);
    }
  }
?>
```

php/submission/toolSingleView.php

This code is executed when a HTML form from the tool single view page is submitted.

```
<?php
  if(isset($trim["submittedTE"]) && isset($_GET["sequence"])){ # Editing a sequence.
    $nucleosome = $trim["nucleosomeTE"];
    $mods = $trim["histoneModsTE"];
    $dna = strtoupper($trim["dnaSequenceTE"]);
    $q = "";
    $matchResult = checkDNA($dna); # Check to see if DNA is already in the database.
    $ndsid = -1; # Not yet assigned with an actual ID value, but needs to be decared so that it
has the correct variable scope.
    if($matchResult == FALSE) {
      $r = mysqli_query($db, "INSERT INTO nucleosomednasequence VALUES(NULL, '$dna')"); # Insert
a new record into the nucleosomednasequence table.
      $ndsid = mysqli_fetch_array(mysqli_query($db, "SELECT ndsid FROM nucleosomednasequence
ORDER BY ndsid DESC LIMIT 1"))[0]; # Fetch the ID from the latest record (the one we just
submitted).
    }
    else {
      $ndsid = mysqli_fetch_array(mysqli_query($db, "SELECT ndsid FROM nucleosomednasequence
WHERE DNASequence='".$dna."'"))[0]; # Fetch the nucelsome DNA sequence ID
    }
    if($nucleosome == "NULL"){ # This means that we need to create a new record.


      $q = "INSERT INTO nucleosome VALUES(NULL, $ndsid, '$mods', ".$_GET["sequence"].")"; #
```

```
Query for creating a new nucleosome record.
    }
    else { # This means we need to update a record.
      $q = "UPDATE nucleosome SET ndsid=$ndsid, histoneMods='$mods' WHERE nid=$nucleosome"; #
Update the values of an already existing nucleosome.
    }
    $r = mysqli_query($db, $q);
    echo('<script>window.location.href="index.php?page=tool&sequence='.$nsid.'";</script>'); #
Reload the page to see the differences.
  }
  else if(isset($trim["submittedTEO"]) && isset($_GET["sequence"])){ # If the overview details
of a sequence is being edited.
    $nsid = $_GET["sequence"];
    $notes = checkNotes(strip_tags($trim["notesTEO"]));
    $name = strip_tags($trim["nameTEO"]);
    $did = $trim["diseaseAssociationTEO"];
    $errors = [];

    if($name == "Name"){
      $errors = ["Cannot use default values."];
    }

    if(empty($errors)){
      $r = mysqli_query($db, "UPDATE nucelosomesequence SET notes=$notes, name='$name', did=$did
WHERE nsid=$nsid"); # Update the nucleosome sequence record with the new overview data.
      echo('<script>window.location.href="index.php?page=tool&sequence='.$nsid.'";</script>'); #
Reload the page to see the differences.
    }
    else {
      echo($popupTop);
      foreach($errors as $error){
        echo('<p>'.$error.'</p>');
      }
      echo($popupBottom);
    }
  }
  else if(isset($trim["submittedTEO"]) && isset($_GET["disease"])){ # If the user is editing the
disease overview.
    $did = $_GET["disease"];
    $notes = strip_tags($trim["notesTEO"]);
    $name = strip_tags($trim["nameTEO"]);
    $errors = [];

    if($name == "Name"){
      $errors = ["Cannot use default values."];
    }

    if(empty($errors)){
      $r = mysqli_query($db, "UPDATE disease SET notes='$notes', name='$name' WHERE did=$did");
# Update the overview values of the disease.
      echo('<script>window.location.href="index.php?page=tool&disease='.$did.'";</script>'); #
Reload the page so that we can see the differences.

    }
```

```
    else {
      echo($popupTop);
      foreach($errors as $error){
        echo('<p>'.$error.'</p>');
      }
      echo($popupBottom);
    }
  }
  else if(isset($trim["nsidTD"]) && isset($_GET["disease"])){ # If the user is removing a
sequence-disease association.
    $nsid = $trim["nsidTD"]; # Fetch the ID of the sequence.
    $r = mysqli_query($db, "UPDATE nucelosomesequence SET did IS NULL WHERE nsid=$nsid"); # Stop
it from being related to disease record.
    echo('<script>window.location.href="index.php?page=tool&disease='.$did.'";</script>'); #
Reload the disease page to see the differences.
  }
?>
```

php/submission/submissionONLOAD.php

This code is executed when `index.php` loads. It performs the same function to the `onloadLOAD.php` script but deals with scripts that are executed when a HTML form is submitted instead.

```
<?php
  if(isset($_COOKIE["user"])){ # Submission scripts to be loaded for signed in users only.
    require_once("forum.php");
    require_once("tool.php");
    require_once("search.php");
    require_once("toolSingleView.php");
    require_once("deletionScripts.php");
  }
  require_once("account.php"); # The page that anyone can submit too.
?>
```

php/dictionary.php

This code contains the class definition for the dictionary data structure.

```
<?php
  class dictionary{
    public function dictionary($keyValues = []){ # This constructor is called when a dictionary
object is instantiated.
      $this->keys = [];
      $this->values = [];
      for($i=0;$i<count($keyValues);$i++){
        if(gettype($i/2) == "double"){ # If integer is not formed when didvided by two, must be
an odd number.
          array_push($this->values, $keyValues[$i]);
        }
        else{ # Else must be even so a key.
          array_push($this->keys, $keyValues[$i]);
```

```
        }
      }
    }

    private function getKeyPosition($key){ # Fetch the position of the key. From thsi we can
tell if a key exists or not. Only the class definition can call it.
      if(!in_array($key, $this->keys)){ # If the key is not in the array.
        return -1; # Because if no position is found, an error code of -1 s thrown.
      }
      else {
        $pos = array_search($key, $this->keys); # Fetch the position of the key.
        return $pos;
      }
    }

    public function checkKeyExists($key){ # Made a new function with a new name to be more
developer friendly as the method name is more relevant.
      $val = dictionary::getKeyPosition($key);
      return ($val = -1 ? FALSE : TRUE); # Return TRUE if a key does have a positiona dn
therefore exists. Else, false.
    }

    public function add($key, $value){
      array_push($this->keys, $key); # Append the new key to the end of the $keys array.
      array_push($this->values, $value); # Append the new value to the end of the $values array.
      return 0;
    }

    public function remove($key){
      $pos = dictionary::getKeyPosition($key);
      if($pos != -1){ # Make sure the getKeyPosition() functions has not returned an error.
        array_splice($this->keys, $pos, 1); # Remove the key and value pair from the arrays at
the specified postion, $pos. This function shifts indexes too if an element is removed in the
middle of an array.
        array_splice($this->values, $pos, 1); # ^
        return 0; # 0 is the success code.
      }
      else {
        return 1;
      }
    }

    public function read($key){ # Read a value froma key.
      $pos = dictionary::getKeyPosition($key);
      if($pos != -1){
        $value = $this->values[$pos]; # Retrive the value at the specified location.
        return $value;
      }
      else {
        return 1;
      }
    }
```

```
    public function editValue($key, $value){ # Edit the value as the specified key.
      $pos = dictionary::getKeyPosition($key);
      if($pos != -1){
        $this->values[$pos] = $value; # Set value at the corressponding point of its key in the
$values array.
        return 0;
      }
      else {
        return 1;
      }
    }

    public function readAll(){
      $finalArray = []; # Initialise the array.
      for($i=0;$i<count($this->keys);$i++){
        array_push($finalArray, $this->keys[$i]);
        array_push($finalArray, $this->values[$i]);
        # Creates an array in the format x, y where x is the key and y is the value.
      }
      return $finalArray;
    }
  }
?>
```

php/hashing32.php

This code is the hashing algorithm that I created called `hashing32`.

```
<?php
   function hashing32($str){ # Takes a given string (str) and returns a hashed string with a
length of 256 charcaters.
     $chars = str_split($str); # Splits the string into individual charcaters (as the default
chunk length of each element of the returned array is 1).
     $finalStr = "";
     foreach ($chars as $char) {
       $ascii = ord($char); # Retrives the ascii denary value of the character.
       $ascii *= 817513877; # ascii denary multiplied by the a large prime number. Malicous
users must divide the string by this value in order to find the charcter of the string used
accurately,
       # as the reurned value of this statement could be a multiple of many different
characters. Therefore malicous users cannot be certain (without enough computational power and
time) what the charcater is.
       $hex = dechex($ascii); # Converts the denary value to a hexidecimal form.
       $finalStr .= $hex;
     }
     if(strlen($finalStr) < 32){
       for($i = 0; $i < (32 - strlen($finalStr)); $i++){
         $finalStr .= "1";
       }
     }
     else if(strlen($finalStr) > 32){

       $finalStr = str_split($finalStr, 32)[0]; # Retrive the first 32 charcters of the string.
```

```
    }
    return $finalStr;
  }
?>
```

php/queue.php

This code contains the class definition for the queue data structure.

```
<?php
  class queue { # Defines the circular queue object class.
    public function queue($length = 100){ # This defines the constructor for instatiations of
the class.
      $this->elements = []; # Contains the array of elements in the current queue. Each element
contains a value.
      $this->headerPointer = 0; # Contains the index of the elment at the beginning of the
queue.
      $this->backPointer = 0; # Contaisn the index for the last element in the queue.
      $length <= 0 ? $this->length = 100 : $this->length = $length; # If the length is lower
than or equal to 0, set the length of the queue to a default. Otherwise it is set to a user
defiend length.
      for($i = 0; $i < $length; $i++){ # Fills the array with "empty" elements so that the array
if fixed in size from instantiation.
        $this->elements[$i] = "";
      }
    }

    public function get(){ # Gets the first value in the queue.
      return $this->elements[$this->headerPointer];
    }

    public function pop(){ # Removes the first element in the queue from the array.
      $this->elements[$this->headerPointer] = ""; # An "empty" element.
      $this->headerPointer++; # Increments the headerPointer to the next element in the queue.
      if(($this->headerPointer + 1) > $this->length){ # If the header pointer is beyond the
length of the queue, we move it back to point to the beginning of the queue.
        $this->headerPointer= 0;
      }
    }

    public function append($value){ # Adds an element to back of a queue.
      $this->elements[$this->backPointer] = $value; # Sets the value at the back of the queue.
      if(($this->backPointer + 1) == $this->length){ # If the next element would otherwsie be
added to the beyond the queue. Set the backPointer to the front.
        $this->backPointer = 0;
      }
      else {
        $this->backPointer++; # otherwise increment the back pointer so that we can add to the
next element.
      }
    }
```

```
    public function isEmpty(){
      if($this->get() == ""){ # If the first element in the queue is empty then the entire queue
must be empty.
        return TRUE;
      }
      else {
        return FALSE;
      }
    }
  }
?>
```

php/toolFunctions.php

This file contains functions that are used in either **submission/tool.php** or **onload/tool.php**.

```
<?php
  # Functions for both the edit tool form and the create tool form, for both submission and on-
load pages.
  function interpretHistoneModSequence($mods, $db){
    $result = 0; # Initilaise the result varibale.
    foreach($mods as $mod){ # For each mod in the sequence.
      $q = "SELECT effectType, magnitude FROM histonemods WHERE hmid=".intval($mod);
      $r = mysqli_query($db, $q); # Fetch the magnitude of its effect and whether or not it is
repressive.
      $row = mysqli_fetch_array($r); # Fetch the actualy values of these.
      if($row[0] == "1"){ # If a repressive effect, decrease $result.
        $result -= $row[1];
      }
      else { # If an activator effect, increase $result.
        $result += $row[1];
      }
    }
    return $result;
  }

  function checkDNA($dna){ # This checks to see if a DNA sequence has already been added to the
database.
    global $db;
    $r = mysqli_query($db, "SELECT DNASequence FROM nucleosomednasequence"); # Fetches all
current DNA sequences form the database.
    while($row = mysqli_fetch_array($r)){ # Checks each row to see if it has a dna sequence that
matches our one ($dna).
      if($dna == $row[0]){
        return TRUE; # Return the boolean true if so.
      }
    }
    return FALSE; # If no match is found return false.
  }

  function checkNotes($notes){ # Allows for notes to be NULL to save database storage.

    if($notes == "Notes"){
```

```
      $notes = "NULL";
    }
    else{
      $notes = "'".$notes."'"; # SQL-friendly string.
    }
    return $notes;
  }
  # ===
?>
```

js/accountScript.js

This contains JavaScript scripts that are used by the account page.

```
function loadUserData(name, ea, admin){ // Displays the user's name, if they are an admin or not, and email address. It also displays a form for changing the password. Displays the sign out button aswell.
  document.getElementById("accountPageContent").innerHTML = '<form id="accountDetailsForm" action="index.php?page=account" method="post" class="greenYellow floatAesthetic"><p class="text">'+name+'<br><br>'+ea+'<br><br>'+admin+'</p><br><br><input name="passwordAE" type="password" value="Password" info="Password" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input name="passwordConAE" type="password" value="Password" info="Password" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input name="submitAE" type="submit" value="Submit"/><input name="submittedAE" type="hidden" value="TRUE"></type><br><div id="signOutBtn" class="button redBtn large" onclick="document.cookie = \'user=;expires=Thu, 01 Jan 1970 00:00:00 UTC\'; window.location.href=\'http://www.google.co.uk\';"><p>Sign out</p></div><br><br>';
}

function loadAccountPage(emailAddress){ // This displays the sign in and sign up forms that are processed by PHP when submitted.
  document.getElementById("accountPageContent").innerHTML = '<br><div id="con"><div id="signUpForm" class="redPurple floatAesthetic"><form action="index.php?page=account" method="post"><br><br><input name="firstNameSU" type="text" value="Rosalind" info="Rosalind" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input name="secondNameSU" type="text" value="Franklin" info="Franklin" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input name="emailAddressSU" type="email" value="rf@example.com" info="rf@example.com" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input id="passwordConSU" name="passwordConSU" type="password" value="Password" info="Password" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><select name="interestSU"><option value="NULL">Interest</option><option value="Scientist">For science!</option><option value="Company">For a company</option><option value="Student">I\'m a student :)</option><option value="Curious">Just curious</option></select><input type="submit" value="Sign up" class="button"/><input name="submittedSU" type="hidden" value="TRUE"/></form></div><div id="signInForm" class="greenYellow floatAesthetic"><form action="./index.php?page=account" method="post"><br><input name="emailAddressSI" type="email" value="'+emailAddress+'" info="rf@example.com" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input name="passwordSI" type="password" value="Password" info="Password" onfocus="clearValue(this); selected(this);" onblur="restoreValue(this); deselected(this);"/><input type="submit" value="Sign in" class="button"/><input name="submittedSI" type="hidden" value="TRUE"/></form></div></div><br><br>';
}
```

js/entryScript.js

This contain JavaScript code that is used by the entry page.

```
window.onload = init; // Once the window has loaded call the function init().

function init() {
  pauseSelect = false; // This pauses the select() and deSelect() functions when set to true.

  enterBtn = document.getElementById("enterBtn"); // Get the values of the attributes on the
```

```
element with ID "enterBtn".
  bgObjs = document.getElementsByName("bgObj");
  bgObjCon = document.getElementById("bgObjCon");
  menu = document.getElementById("mainCon");
  body = document.getElementsByTagName("BODY")[0]; // Gets the values of the attributes of the
body element.
  if(document.cookie.substring(document.cookie.indexOf("accessedBefore=TRUE"),
document.cookie.indexOf("accessedBefore=TRUE")+19) === "accessedBefore=TRUE"){
    // The above line of code checks if accessedBefore=True is in the cookie string set by the
website by taking a substring, defined by two indicies.
    // x.indexOf(y) fetches the index of the beginning character of the string y, if y can be
found as a substring within the string x.
    menuPopUp(enterBtn, bgObjs, bgObjCon, menu); // If the above is true show the main screen.
This is used in conjunction with some code on index.php that prevents the first-load screen from
displaying.
  }
  else { //If this is the first time we have accessed the site.
    document.cookie = "colorMode=day;"; // Set a cookie to set the theme of the website to
daytime.
  }
  enterBtn.onclick = function(){ // When the first-load enterBtn is clicked (only displayed if
this is the first time we have accessed the website so no need to put it in an if statement.)
    menuPopUp(enterBtn, bgObjs, bgObjCon, menu); // Displays the main screen.
    document.cookie = "accessedBefore=TRUE;"; // Set a new cookie that teels us to automatically
move pass the first-load screne
  }
  btn = document.getElementById("dayChangeBtn");
  loadColorScheme(btn);
  closeBtnClick();
}

function randint(min, max) { // Generate a random positive integer between the minimum and
maximum values.
  return Math.floor(Math.random() * ((max-min) +1)) + min;
  /* Above line rounds down the number given. Math.random() generates a random float between 0
and 1.
     We find the difference of the two ranges. We add one so that we do not have a floor result
of zero and therfore have a value that is always greater than the minimum value.
     We then add the minimum value to the floored result so that it is within the range we
wanted.*/
}

function menuPopUp(enterBtn, bgObjs, bgObjCon, menu) { // Displays the main screen.
  enterBtn.style.animationName = "flyAway"; // Sets the button neccessary for entering the
website, to have an animation. This triggers the animation to start automatically.
  enterBtn.style.animationDuration = "7s";
  enterBtn.style.animationIterationCount = "1";
  for(i = 0; i < bgObjs.length; i++) { // For all of the objects BESIDES the enterBtn that makes
up our first load screen.
    bgObjs[i].style.animationFillMode = "forwards";
    bgObjs[i].style.animationIterationCount = "1";


    bgObjs[i].style.animationDuration = String(randint(3, 12))+'s'; // Gives each one a random
```

```
animation duration.
    bgObjs[i].style.animationName = "flyAway";
  }
  menu.style.animationName = "fadeIn"; // This lets the main screen fade into being displayed.
  setTimeout(function(){
    bgObjCon.innerHTML = ""; // Remove all of the elements within the first load screen so less
memory is used.
    enterBtn.style.display = "None"; // Prevent the enter button from being displaued/
  }, 3000); // setTimeOut executes a function after an amoutn of time in milliseconds.
}

function setColorScheme(mode){ // The sets the theme of the website. mode=the colorscheme we
want to switch to.
  sheetEle = document.createElement("style"); // This creates a sttyle element.
  document.head.appendChild(sheetEle); // Append the element to be nested within the head tag.
  sheet = sheetEle.sheet; // Get the sheet element that has been nested within the head element.
  elements = document.getElementsByTagName("*"); // Fetch ALL elements.
  colorsTypes = ["color", "backgroundColor", "boxShadow"];
  menuCon = document.getElementById("menuCon").style; // Get the style attribute content of the
element with ID menuCon.
  menuIcons = document.getElementsByTagName("i"); // Gets the menu icons as they use the Google
icons syntax which requires being wrapped in i tags.
  mainCon = document.getElementById("mainCon");
  if(mode === "night"){
    sheet.insertRule("button:hover, input:hover, select:hover, textarea:hover  {background-
color: #333 !important;border-color: #333 !important;}",0); // This adds a new rule to the sheet
eleemnt we created earlier.
    sheet.insertRule("#mainCon::-webkit-scrollbar {background-color: rgb(51, 51, 51);}", 1); //
The scollbar is a pseudo-element so I had to insert a new rule.)
    sheet.insertRule("#mainCon::-webkit-scrollbar-thumb {background-color: rgb(239, 239,
239);}", 2); // Ditto
    pauseSelect = true; // This means that select and deSelect are now disabled.
    for(i=0;i<elements.length;i++){ // For each element.
      colors = [window.getComputedStyle(elements[i]).getPropertyValue("color"),
window.getComputedStyle(elements[i]).getPropertyValue("background-color"),
window.getComputedStyle(elements[i]).getPropertyValue("box-shadow")];
      // window.getComputedStyle(x).getPropertyValue(y) fetches the value of style property y,
from the element x.
      for(j=0;j<colors.length;j++){ // Go through the colors for each element.
        switch(colors[j]){ // Check the value of the computed style.
          case "rgb(239, 239, 239)":
            eval('elements[i].style.'+colorsTypes[j]+' = "rgb(51, 51, 51)";'); // We use the
colorTypes array to fetch the current attribute name so that we can assign a value to it
            break;                                                                  // on the fly
instead of hardcoding each possibility of attribute/value pair (125 pairs).
          case "rgba(239, 239, 239, 0.7)":
            eval('elements[i].style.'+colorsTypes[j]+' = "rgba(51, 51, 51, 0.7)";'); // eval(x)
execute the string x as if it was JavaScipt code.
            break;
          case "rgba(238, 238, 238, 0.7)":
            eval('elements[i].style.'+colorsTypes[j]+' = "rgba(51, 51, 51, 0.7)";');
            break;

          case "rgb(51, 51, 51)":
```

```
            eval('elements[i].style.'+colorsTypes[j]+' = "rgb(239, 239, 239)";');
            break;
          case "rgb(170, 170, 170) 7px 7px 17px 0px": /*For any box-shadows.*/
            eval('elements[i].style.'+colorsTypes[j]+' = "7px 7px 17px rgb(0, 0, 0)";');
            break;
          default:
            break;
        }
      }
    }
    menuCon.backgroundColor = "rgb(64, 64, 64)"; // Menu bar is set tot be slightly lighter than the backgorund so that it can be clearly seen.
  }
  else if(mode === "day"){
    pauseSelect = false; // Enables the select and deselct functions to be used.
    sheet.insertRule("button:hover, input:hover, select:hover, textarea:hover  {background-color: #EFEFEF !important;border-color: #444 !important;}",0);
    sheet.insertRule("#mainCon::-webkit-scrollbar {background-color: rgb(239, 239, 239);}", 1);
    sheet.insertRule("#mainCon::-webkit-scrollbar-thumb {background-color: rgb(51, 51, 51);}", 2);
    for(i=0;i<elements.length;i++){
      colors = [window.getComputedStyle(elements[i]).getPropertyValue("color"), window.getComputedStyle(elements[i]).getPropertyValue("background-color"), window.getComputedStyle(elements[i]).getPropertyValue("box-shadow")];
      for(j=0;j<colors.length;j++){
        switch(colors[j]){
          case "rgb(239, 239, 239)":
            eval('elements[i].style.'+colorsTypes[j]+' = "rgb(51, 51, 51)";');
            break;
          case "rgba(51, 51, 51, 0.7)":
            eval('elements[i].style.'+colorsTypes[j]+' = "rgba(239, 239, 239, 0.7)";');
            break;
          case "rgb(51, 51, 51)":
            eval('elements[i].style.'+colorsTypes[j]+' = "rgb(239, 239, 239)";');
            break;
          case "rgb(0, 0, 0) 7px 7px 17px 0px":
            eval('elements[i].style.'+colorsTypes[j]+' = "7px 7px 17px rgb(170, 170, 170)";');
            break;
          default:
            break;
        }
      }
    }
    menuCon.backgroundColor = "rgb(51, 51, 51)";
  }
  signOutBtn = document.getElementById("signOutBtn");
  if(signOutBtn){
    signOutBtn.innerHTML = "<p style='color:#EFEFEF;'>Sign out</p>"; // Maintains the colour of the text within the sign out button.
  }
  for(i=0;i<menuIcons.length;i++){ // Maintains the colour for the icons withinnthe menu.
    menuIcons[i].style.color = "rgb(239, 239, 239)";

  }
```

```
}

function loadColorScheme(btn){ // Used when index.php loads to load the currently set colour scheme.
  colourMode = document.cookie.substring(document.cookie.indexOf("colorMode"), document.cookie.indexOf("colorMode")+11); // Get the value of the cookie that is set to define the colour mode.
  if(colourMode === "colorMode=n") { //If it is in night mode, set the dayChange icon to a picture of a sun and change the theme of the website to night mode.
    btn.innerHTML = '<i class="material-icons">brightness_1</i>';
    setColorScheme("night");
  }
  // If it is inday mode, we do not need to load a new colour scheme as it the website is in day mode by default.
}

function dayChangeClick(){ // Used after index.php has loaded. loadColorScheme cannot be used for a simple solution as it only reads the cookie and does not set it as well as the fact that it cannot deal with loading a colour change to day mode.
  btn = document.getElementById("dayChangeBtn");
  if(document.cookie.substring(document.cookie.indexOf("colorMode"), document.cookie.indexOf("colorMode")+11) === "colorMode=n"){ // Get the colour mode cookie.
    document.cookie = "colorMode=day;"; // Set that cookie to the new value.
    setColorScheme("day")
    btn.innerHTML = '<i class="material-icons">brightness_3</i>';
  }
  else {
    document.cookie = "colorMode=night;";
    setColorScheme("night");
    btn.innerHTML = '<i class="material-icons">brightness_1</i>';
  }
}

function closeBtnClick(){ // Closes all notification tabs when clicked. Not possible for more than one pop up, so closing all notifixation is sensible.
  crosses = document.getElementsByClassName("crossPU");
  tabs = document.getElementsByClassName("boardConPU");
  for(i=0;i < crosses.length; i++){
    crosses[i].onclick = function(){ // If any one of the crosses are clicked
      for(j=0;j < tabs.length; j++){ // Every tab is...
        tabs[j].style.display = "none"; // set to not display.
      }
    }
  }
}

function reload(){ // One method of reloading a page. Reload the page if the laoded hash query is not in the URL, but before reloading add that #loaded to prevent further reloads.
  if(!window.location.hash){
    window.location += "#loaded";
    window.location.reload();
  }

}
```

js/inputScript.js

This file contains JavaScript code that is used by HTML form inputs.

```
function dnaInputCheck(element){ // Ensures inputted DNA character is valid.
  value = element.value[element.value.length-1];
  value = value.toUpperCase(); // Converts all values to be upper case, as text-transform:
capitalise does not change the value of the input to be capitalised. Capital form allows for
less variety in the potential values for 'value'.
  if (value !== "A" && value !== "G" && value !== "T" && value !== "C" && value !== "N") { // N
is for unidentified bases.
    element.value = element.value.substr(0, element.value.length-1); // If the character is not
valid then we remove it.
  }
}
// Functions for selecting and delecting an input.
  function selected(element){
    if(pauseSelect === false){ // Only works if night mode is not enabled.
      element.style.backgroundColor = "#EFEFEF";
    }
  }

  function deselected(element){
    if(pauseSelect === false){
      element.style.backgroundColor = "rgba(238, 238, 238, 0.7)";
    }
  }
 // Functions for removing and setting the default value for an input
  function clearValue(element){ // Called when the user is focusing on the input.
    if(element.value == element.getAttribute("info")){ // Checks if the value of input is the
same as its default value, denoted buy the attribute 'info'.
      element.value = "";
    }
  }

  function restoreValue(element){ // Called when the user is not focusing on the input.
    if(element.value == ""){
      element.value = element.getAttribute("info"); // Resets the default value if not data has
been inputted.
    }
  }
```

js/menuScript.js

This JavaScript code is used for the menu sidebar that is found in the user interface.

```
function menuBtnClick(value) { // Used to switch between pages. value represents the page the
user has clicked on.


  document.getElementById("homePage").style.opacity = '0'; // Prevents the homePage from popping
```

```
up everytime index.php loads to create a more seamless transition.
  postEle = document.getElementsByClassName("postBtn");
  if((value === "forum" || value === "forumSingleView") &&
document.cookie.substring(document.cookie.indexOf("user"), document.cookie.indexOf("user")+4)
=== "user"){ // If the user is logged in and one of the forumpages are being accessed.
    for(i=0;i < postEle.length; i++){
      postEle[i].style.display = "block"; // Display all post-related menu elements.
    }
  }
  else { // If any other page is being switched to.
    for(i=0;i < postEle.length; i++){
      postEle[i].style.display = "none";
    }
  }

  if(value === "forum" && window.location.href.indexOf("thread") != -1){ // If we are switching
to the main forum page when threads are listed and the thread query is in our url.
    window.location.href = "index.php?page=forum"; // We reload the page because the user
singleview page cannot have its contents altered unless we lod back into it AFTER we load into
the forum page.
  }
  if((value === "forum" || value === "tool") &&
document.cookie.substring(document.cookie.indexOf("user"), document.cookie.indexOf("user")+4)
!== "user"){
    // If the user is not signed in and is trying to access one of the restricted pages, break
out of the fucntion so tjhat they cannot switch to those pages.
    return 0;
  }
  value += "Page";
  page = document.getElementById(value).style; // Get the style attributes of the page we are
trying to switch to.
  pages = ["homePage", "accountPage", "toolPage", "newsPage", "aboutPage", "forumPage",
"forumSingleViewPage", "toolSingleViewPage", "searchPage"];
  pages.splice(pages.indexOf(value), 1); // Remove the page we are trying to switch to from the
pages array.
  for(i = 0; i < pages.length; i++){
    document.getElementById(pages[i]).style.animationName = "fadeOut"; // Animates the pages we
are not switching to, to fade out.
  }
  setTimeout(function () {
    for(i = 0; i < pages.length; i++){
      document.getElementById(pages[i]).style.display = "none";
    }
  },1000); // After they all have stopped animating stop displaying those aforementioned pages.
  page.display = "block"; // Display the page we ae switching too.
  page.animationName = "fadeIn"; // Animate it to smoothen the switch.
}

function displayPU(id){ // Displays a pop up to the user.
  document.getElementById(id).style.display = "block";
}
```

js/toolScript.js

This javaScript code is by the tool page.

```
function addHmod(name, id, display = "hmodSequence", list = "histoneModsT"){
  // Defaults for output: display is where the list of mods is shown to the user. list is the
actual list of ids that we will use in processing.
  document.getElementById(display).innerHTML += String(name)+" "; // Add the histone
modification name to the display div.
  document.getElementById(list).value += String(id)+","; // Append the id of the histone
modification tothe hidden input list.
}

function removeLastHmod(display = "hmodSequence", list = "histoneModsT"){
  text = document.getElementById(display);
  list = document.getElementById(list);
  mods = text.innerHTML.split(" "); // Splits up all of the dsiplay mod names.
  ids = list.value.split(","); // Splits up the list of ids.
  mods.splice(mods.lengths-1, 1); // Removes the last name from the display.
  ids.splice(ids.lengths-1, 1); // Removes the last id from the list.
  final = "";
  for(i=0;i<mods.length;i++){
    final += String(mods[i])+" "; // Append each mod name with a space at the end to create the
name name display.
  }
  text.innerHTML = final; // Set the HTML content of the display to the new sequence of mod
names.
  list.value = String(ids);
}

function split(display = "hmodSequence", list = "histoneModsT"){ // | Signifies the end of one
nucleosome and the beginning of another.
  text = document.getElementById(display);
  list = document.getElementById(list);
  text.innerHTML += " | ";
  list.value += "|";
}
```

Nomios - Test Plan

For any application it is important to test it in order to make sure that it meets its specified objectives and that it can catch and deal with any unexpected errors to prevent the application from halting. Therefore, my tests should test the application to ensure that the objectives are met and test the application using extreme, expected and erroneous data to ensure that the application can handle errors correctly.

My program is a website and therefore is run in a web browser. In order to serve the majority of users I must therefore serve those web browsers that the majority of users use. I selected the top three web browsers ordered by market share by calculating a mean average of the figures presented by this Wikipedia page. The top three web browsers in order of highest market share are:

1. Google Chrome at 50.1%
2. Safari at 19.3%
3. UC Browser at 8.32%

Combined, these three browsers make up 77.7% of the market share. These browsers therefore serve the majority of users, so to make sure that my testing is valid (by which I mean that most users can actually use the website) I decided to carry one set of tests per browser.

Each test is designed to help prove that each objective listed in my analysis is met

Below are the test that I will conduct. Remember, each of these tests will be repeated for each browser to make sure that my test is valid for the majority of users and that all for these tests are conducted within the website user interface, rather than any other interface that works with a database (such as a *PHPMyAdmin* - a popular web interface for MySQL databases).

Below are some definitions specific to this document. All other definitions that are related to this document can be found in my Analysis and Documented Design Documents. Note that the *Reference Number* is in hexadecimal and that each number is a key for a particular screenshot in the *Test Results* file. Evidence that a test was successful or unsuccessful will also be entered into this file with he evidence being in the form of screenshots. Each screenshot will also have attatched a short peice of text explaining what the results mean and wheteher or not they satisfy the expected result.

> A **thread** is the set of messages based around a particular topic or question in a forum.
>
> A **message** is a response to the original post that started the thread.
>
> A **reply** is a response to a message within a thread.
>
> **Test** data consists of *extreme* (very ends of the range of acceptable values for an input), *expected* (a typical value to be accepted in an input) and *erroneous* (unexpected and invalid value for an input) data that together work to test the effectiveness of the catching of errors in a program.
>
> A **proto-oncogene** is a normal gene that has mutated and now has the potential to cause a cancerous tumour.
>
> A **sequence** is a DNA sequence associated with a histone modification sequence that alter the expression of the DNA sequence.

Test	Purpose	Expected Result	Reference Number
Start a thread.	Checks if a thread start message can be successfully added to the database.	A record is added to the *threads* table.	0
Post a message.	Checks if a message can be successfully added to the database.	A record is added to the *messages* table.	1
Post a reply.	Checks if a reply can be successfully added to the database.	A record is added to the *replies* table.	2
View a thread from another user.	Checks if a user can view posts by another user. If successful than a user can communicate with other users.	The thread of another user is displayed.	3
Post a message in response to another user's start thread.	Confirms that one user can communicate with another within the forum.	A record is added to the *messages* table.	4
Post a message using HTML mark-up.	To make sure that HTML mark-up is filtered out to prevent a XSS attack, such as UI distortion.	The HTML mark-up is removed from the text before it is submitted to the database.	5
Use an admin user to delete a reply.	Confirms that an admin can remove replies within a thread, including inappropriate ones.	A row is deleted from the *replies* table.	6
Attempt to use a standard user to delete a reply, message or thread.	Confirms that a non-admin user cannot delete their own data within the forum.	The user fails to delete messages, replies and threads from the database .	7
Post a reply, message and thread to the forum that includes stylised text.	This checks to see if the program-defined mark-up is working. This mark-up stylises and structures the text to make it more readable and emphatic.	A reply, message and thread row are added to the *replies*, *messages* and *threads* tables respectively. The text can then be viewed as being styled.	8
Ensure that the website, including the text designed to help the user to use the website is correctly displayed.	This checks the user interface, to make sure it is correctly interpreted and therefore displayed by the browser. This ensure that the majority of users can successfully navigate and use the website including through the use of the help text.	The HTML mark-up and CSS is interpreted correctly and displayed as intended.	9
Create a new user account.	This makes sure that a user can own their own account and therefore, have their own data attributed to it.	A new record is inserted to the *users* table.	A
Attempt to create a new reply, message, thread, sequence and disease when not signed into an account.	This ensures that a user can only have their data attributed to their account by preventing any sort of data submission (besides the creation of a new account) without being signed into an account.	The user fails to submit any new data to the database.	B

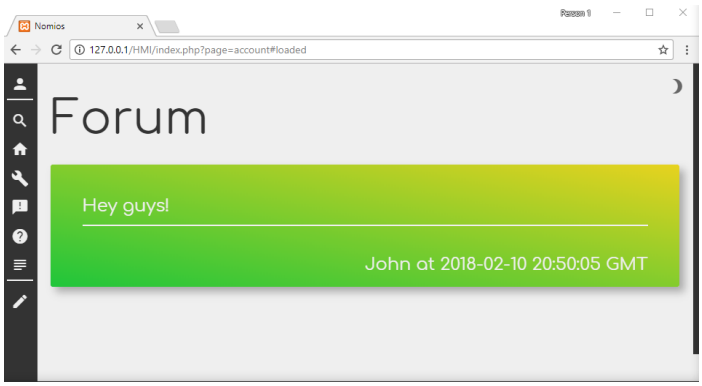
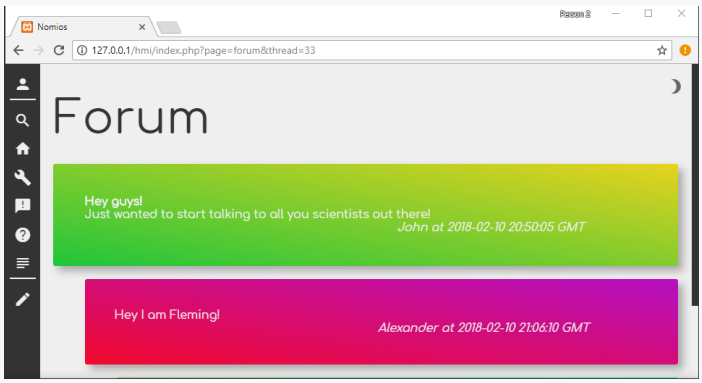
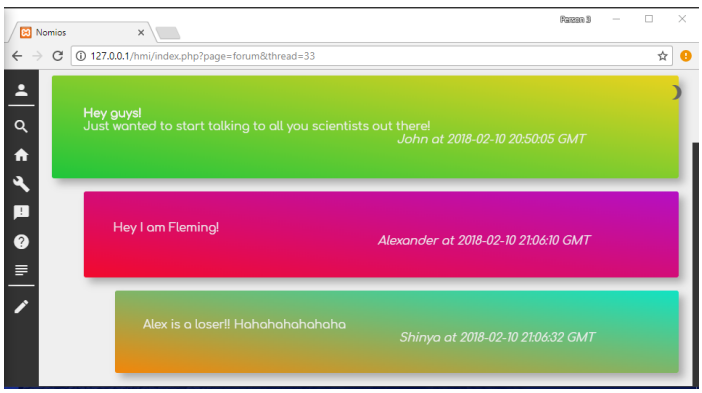
Test	Purpose	Expected Result	Reference Number
Create a new disease.	Checks if a disease can be created and submitted to the database. Also checks to see if a unique name is checked for when creating a *disease* record so that it can be differentiated from others.	A new record is inserted into the *disease* table with a unique *name* attribute.	C
Create a new sequence.	Checks if a sequence can be created and submitted to a database by breaking it down into chunks called *nucleosomes* and storing the sequence as a relationship between these nucleosomes. Each nucleosome also has an associated DNA sequence which is check to be stored successfully in the database too. Also checks to see if a unique name is checked for when creating a *nucelosomesequence* record so that it can be differentiated from others.	New records are inserted into the *nucleosome*, *nucelosomesequence* and *nucleosomednasequence* tables. With the new *nucelosomesequence* record having a unique *name* attribute relative to the user.	D
Attribute the sequence to the disease. Then check if it is displayed as part of the disease.	Checks if a relationship between a disease and a sequence can be created and is successfully recognised by the program.	A change to the *nucelosomesequence* record is detected and is subsequently displayed when viewing the sequences specific to a particular disease.	E
Create another sequence and attribute it to the same disease as previously using another users' account.	Confirms that users can collaborate with each other to work on a set of sequences that are attributed to the same disease.	Both sequences are listed under the same disease, and each can be viewed by the same user.	F
Add a new nucleosome to the sequence using the same user account that created the sequence.	Confirms that a sequence can have new data added to it. It aids the confirmation that the user that created the sequence can edit it.	A new record is added to the *nucleosome* table and can be viewed within the page for a specific sequence.	11
Edit the histone sequence and DNA sequence for a specific nucleosome using the same user account that created the sequence.	This confirms that a nucleosome (and therefore an entire sequence) can be edited by replacing old data with new data. It also confirms that a DNA sequence and histone sequence can be edited by a user. It aids the confirmation that the user that created the sequence can edit it.	A record in the *nucleosome* table is updated. The changes can be viewed within the page for a specific sequence.	12
Delete a nucleosome and a sequence from the database using the same user account that created the sequence.	This confirms that a sequence can have data removed from it (i.e. a nucleosome). It aids the confirmation that the user that created the sequence can edit it.	A record from the *nucleosome* table is removed and its absence noted by the user when reading the sequence's display page. The specified record from *nucelsomeseqence* is removed. All records associated with that *nucelosomesequence* record are also removed.	13

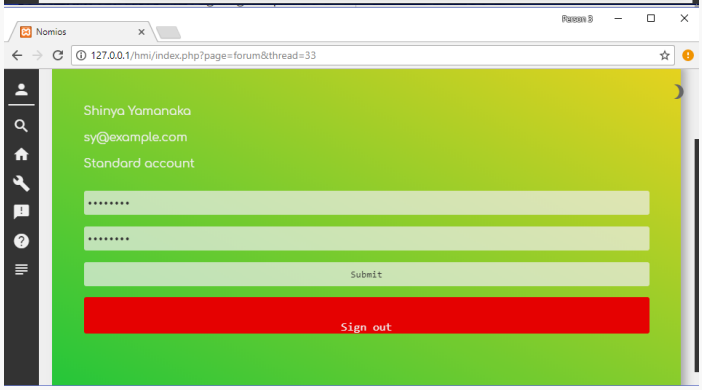
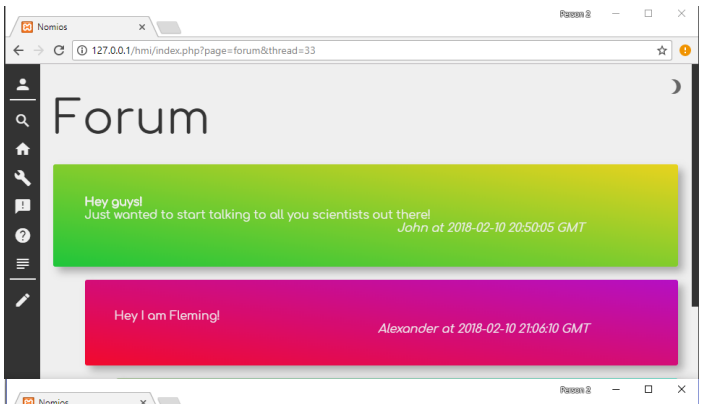
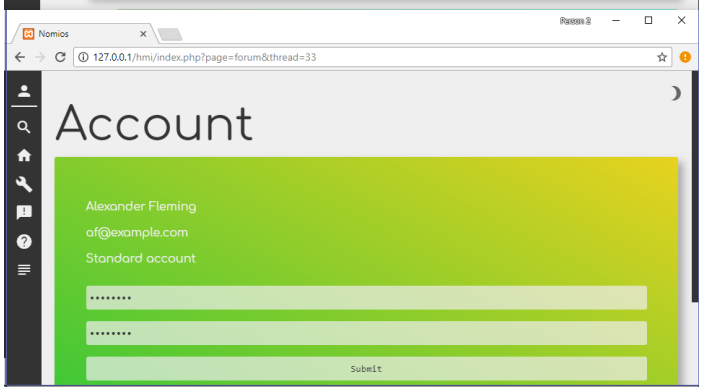
Test	Purpose	Expected Result	Reference Number
Perform the previous three tests using a different account to that which was used to create the sequence.	This will confirm that only the user that creates a sequence can edit it, and that any other users cannot. Therefore preventing malicious attacks between users.	The user fails to update, add or delete any data in the database.	14
Edit the *notes* section of a disease and a sequence.	This will confirm that a user can add a description or note any important information on a particular disease or sequence.	The *notes* attribute of a disease record and a sequence record are both updated in the *disease* and *nucelosomesequence* tables respectively.	15
Interpret a sequence that uses a proto-oncogene that when paired with activator histone modifications causes the risk of cancer to increase.	This will confirm that a result is produced and presented to the user after a sequence is interpreted. It will also confirm that the interpreter is accurate if the result matches the expected outcome by using this real-life example.	A positive result is displayed to the user in the form of a positive integer, denoting an increase in expression.	16
Interpret a sequence that uses a proto-oncogene that when paired with repressive histone modifications causes the risk of cancer to decrease.	This will confirm that a result is produced and presented to the user after a sequence is interpreted. It will also confirm that the interpreter is accurate if the result matches the expected outcome by using this real-life example.	A negative result is displayed to the user in the form of a negative integer, denoting a decrease in expression.	17
Interpret a sequence that uses the APP gene that when paired with activator histone modifications causes the progression of Alzheimer's Disease to increase.	This will confirm that a result is produced and presented to the user after a sequence is interpreted. It will also confirm that the interpreter is accurate if the result matches the expected outcome by using this real-life example.	A positive result is displayed to the user in the form of a positive integer, denoting an increase in expression.	18
Search for a specific sequence and disease that the user has created.	This will confirm that the search function is accurate by returning results specific to what they user has specified. It also confirms that the user can aggregate their sequences and diseases on one page so that they can quickly navigate them.	The sequence and disease searched for is displayed to the user and is owned by the user.	19
Search for a specific sequence and disease that another user has created.	This will confirm that the search function is accurate by returning results specific to what they user has specified. It also confirms that the user can locate another users' sequence or disease and view it for collaborative purposes.	The sequence and disease searched for is displayed to the user.	1A

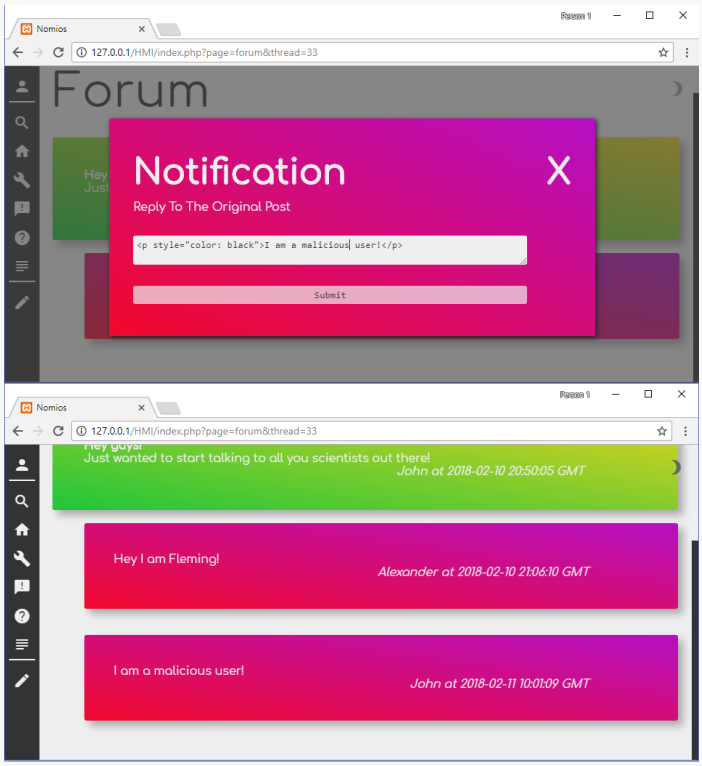
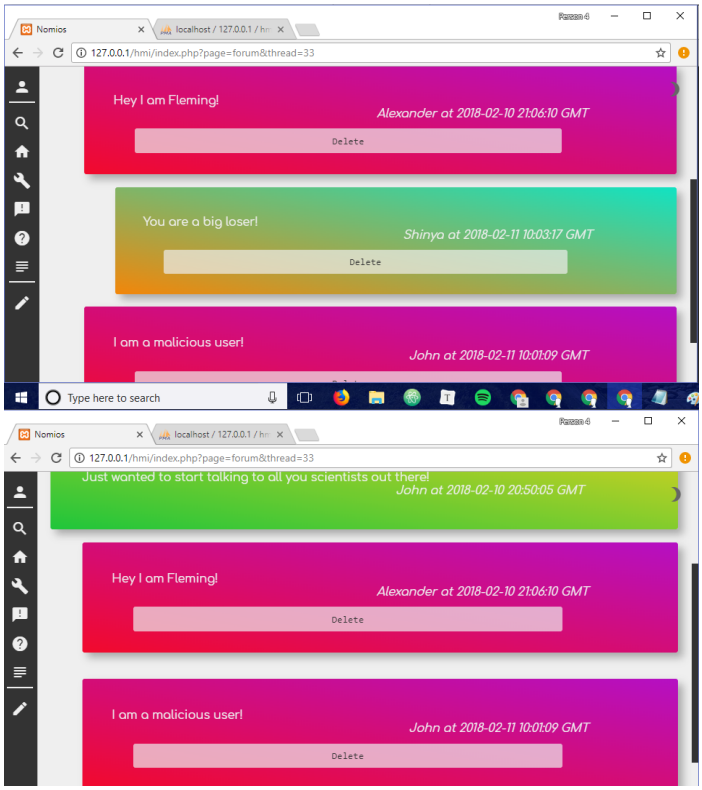
Test	Purpose	Expected Result	Reference Number
Use an admin user to delete a message.	Confirms that an admin can remove messages within a thread, including inappropriate ones.	A row is deleted from the *messages* table and all of the associated *replies* rows are deleted from it too.	1B
Use an admin user to delete a thread.	Confirms that an admin can remove thread within the forum, including inappropriate ones.	A row is deleted from the *thread* table and all of the *message* rows associated with it are delete too, as well as the *replies* rows that are associated with each message.	1C

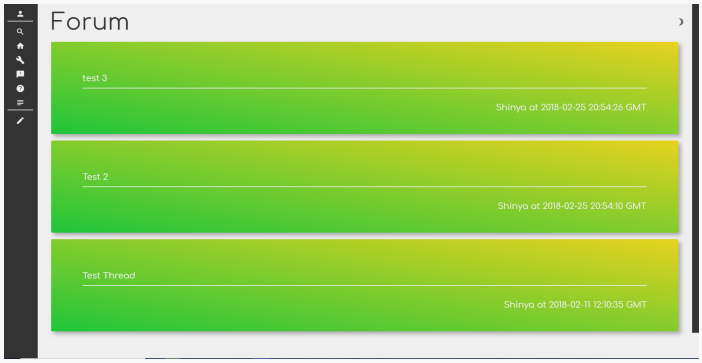
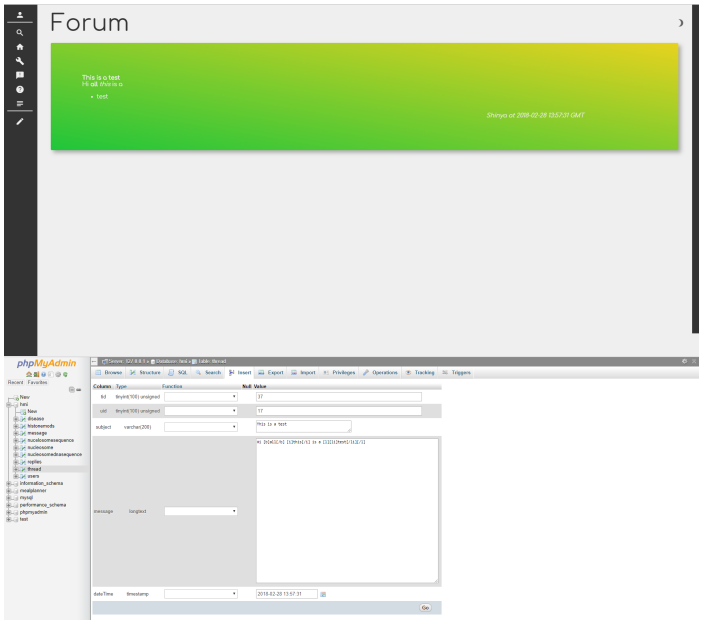
Nomios - Test Results

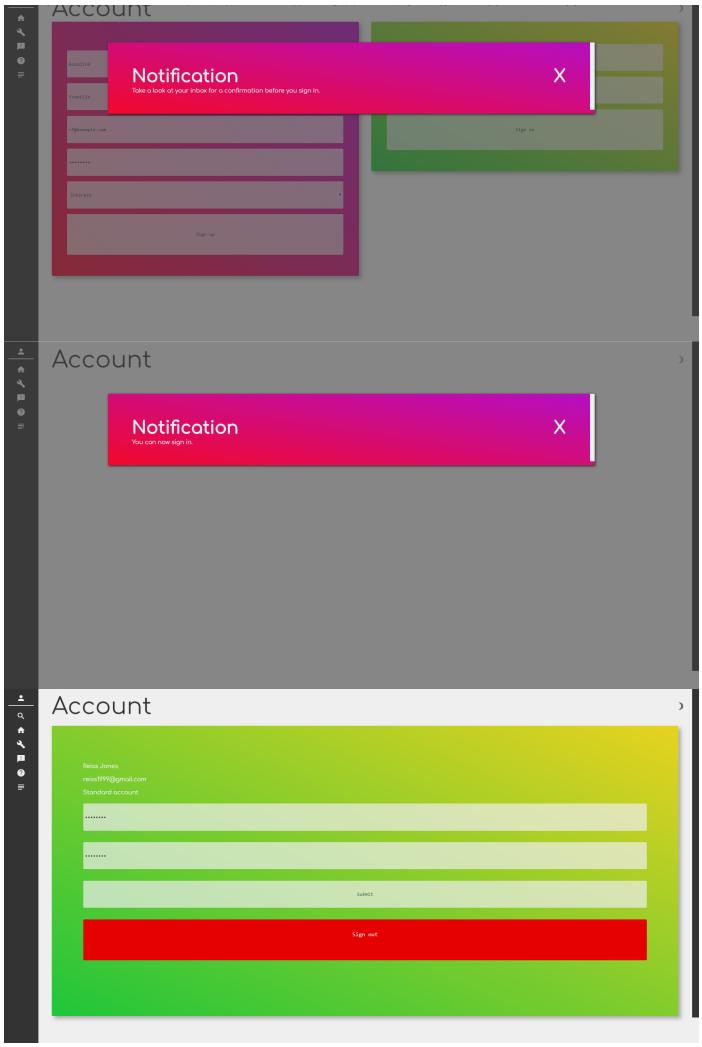
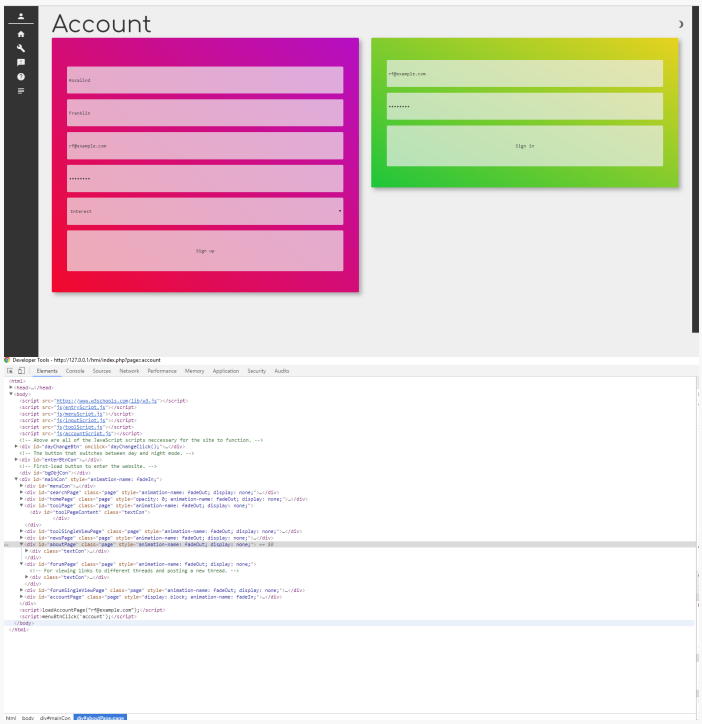
The following table holds the result, the evidence of a result and a short explanation as to what it shows. Each row relates to a test and its expected result as shown in the *Test Plan*. This relation is made by the hexadecimal *Reference Number*.

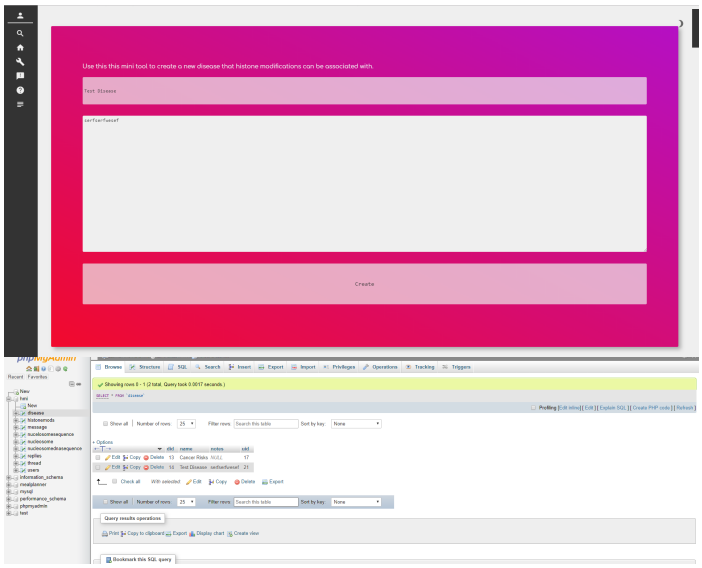
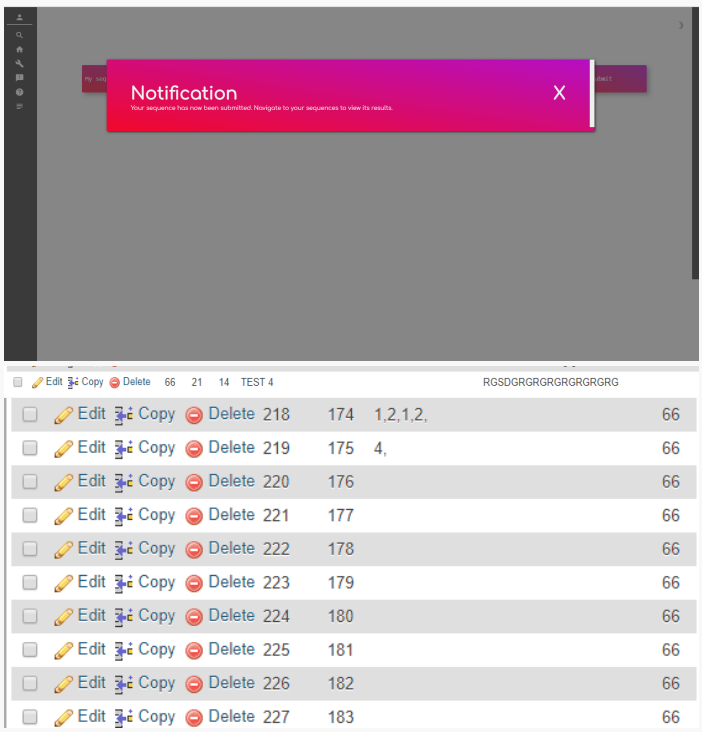
Reference Number	Evidence	Explanation
0		The thread has been **successfully posted**. This is proved as it is displayed on the forum page, therefore it has been submitted to the *thread* table.
1		The message has been **successfully posted**. This is proved as it is displayed on the thread page, nested "within" the original post, therefore it has been submitted to the *messages* table.
2		The reply has been **successfully posted**. This is proved as it is displayed on the thread page, nested "within" the message, therefore it has been submitted to the *replies* table.

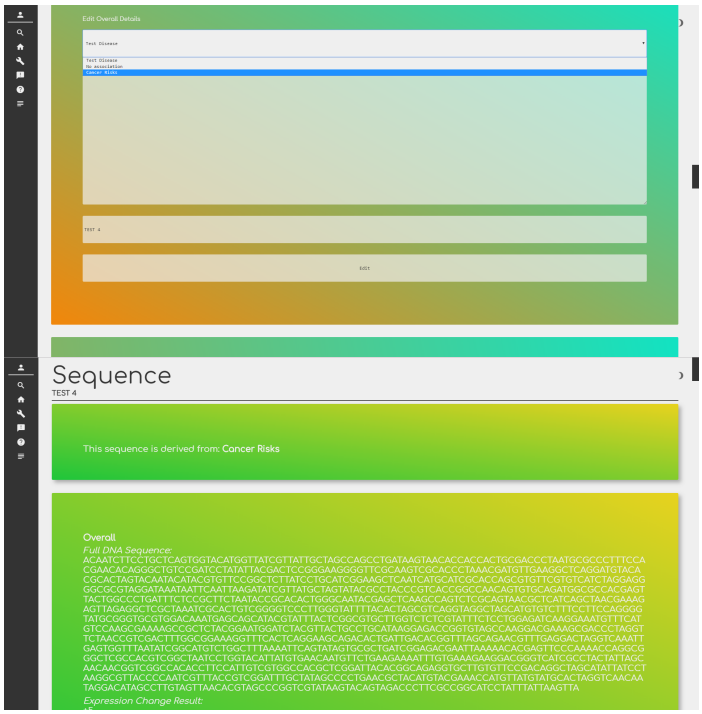
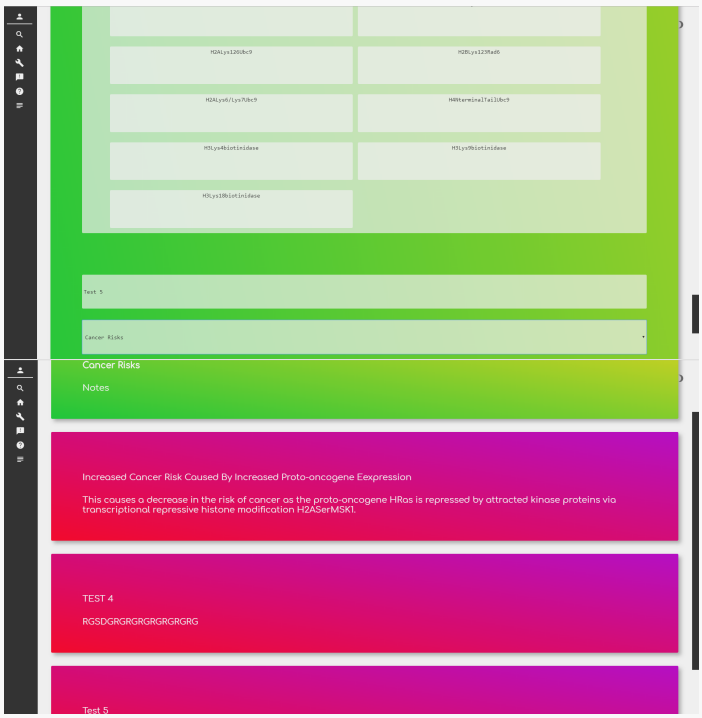
Reference Number	Evidence	Explanation
3	 	We are viewing this thread as the user *Shinya* and the original post is displayed therefore this test is **successful** - users can viewing original posts created by other users in a thread.
4	 	We are viewing this thread as the user *Alexander* and this user has posted a message in response to the original post, therefore a row has been added to the *messages* table. Therefore this test was **successful** as users who did not start the thread can post responses to it.

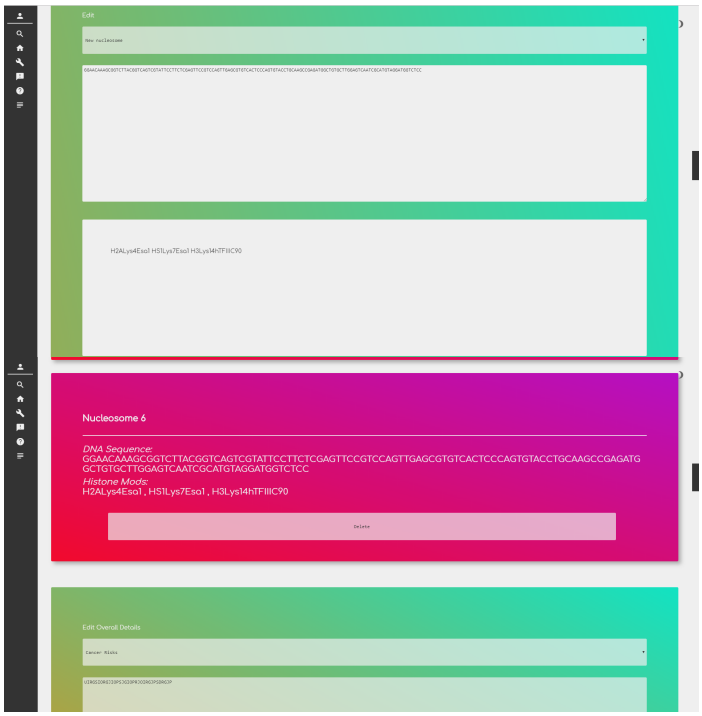
Reference Number	Evidence	Explanation
5		The *John* user attempted to post a message that contains the HTML tags that contains CSS code that would turn containing text black. They were **successfully** filtered out of the message as the message text is not black. This means malicious users can not use a XSS attack.
6		The reply is no longer displayed. This means that it must have been deleted from the *replies* table. This test was therefore **successful** - the admin can delete replies successfully.

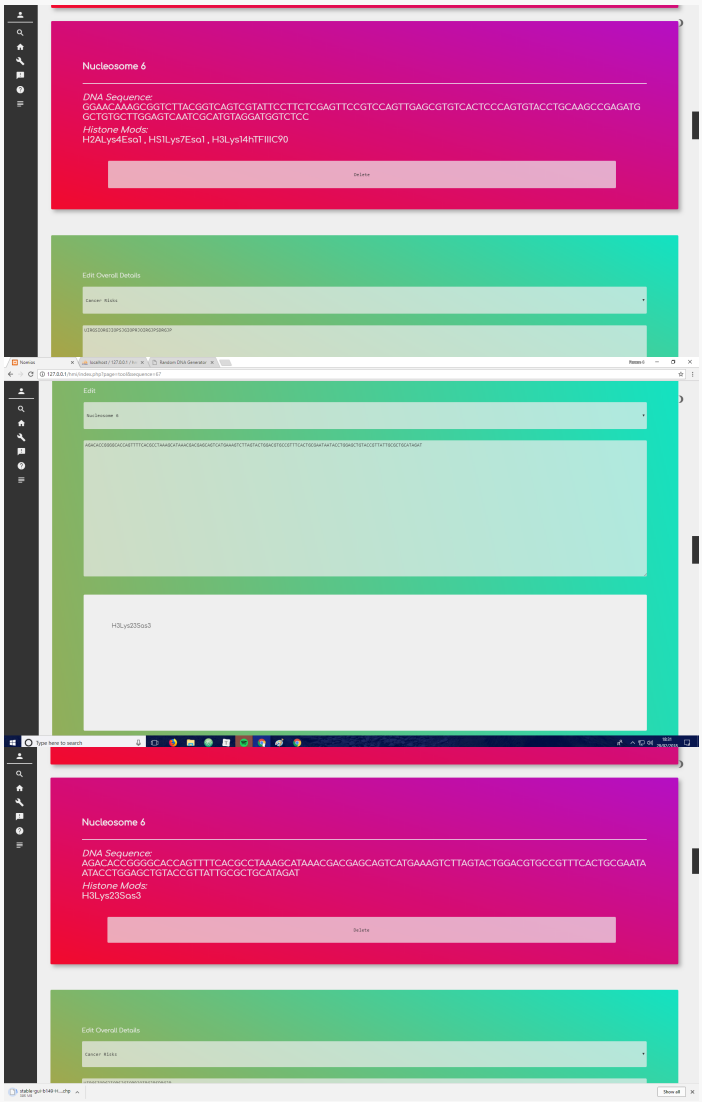
Reference Number	Evidence	Explanation
7		The *Shinya* account is a standard account (i.e. has no admin privileges). As we can see there is no form for deleting a thread. Therefore, the account cannot delete threads, messages or replies. Therefore the test was a success.
8		The *Shinya* account has posted a thread that contains artificial mark up. Its result is displayed first with the text that is stored in the database, second. This shows that the artificial mark up is translated to HTML mark up. Therefore the test was a success.
9		The HTML and CSS is displayed as intended. The help menu is clearly visible. Therefore the test was a success as the HTML and CSS code was correctly interpreted.

Reference Number	Evidence	Explanation
A		The first screenshot shows an account being created. The second shows the same account being activated. The third shows that the user has logged into this account. This progression shows an account creation. Therefore, the test was successful.
B		The first screenshot shows that the user cannot enter the forum (shown by the cursor image when hovered over the forum icon). The second shows that no HTML is displayed in the forum and tool pages. Therefore the user cannot submit any HTML forms. The test was successful.

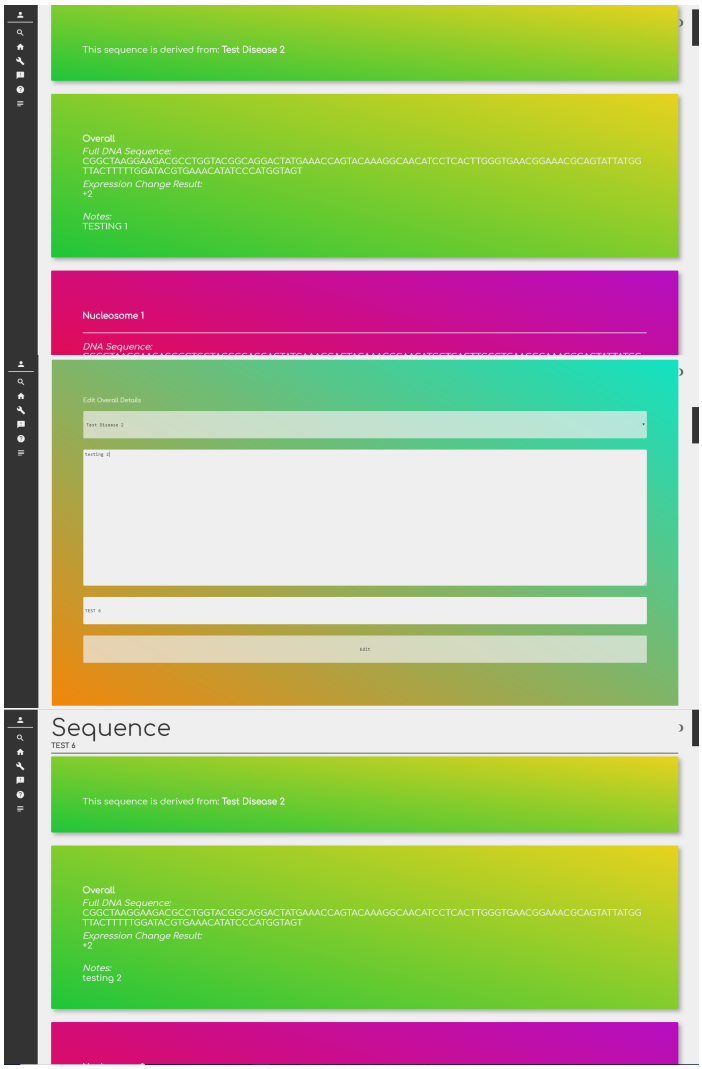
Reference Number	Evidence	Explanation
C		The first screenshot shows the HTML form the user has filled. The second shows the record that has been created in the *diseases* table after the form was submitted. Therefore, the test was successful.
D		The first screenshot shows us that the nucleosome sequence has been submitted. The second shows that a new record in the *nucleosomesequences* table has been made. The final screenshot shows that new records in the *nucleosomes* table have been made. Some *nucleosomednasequences* records may have been created or reused. Thus the test was successful.

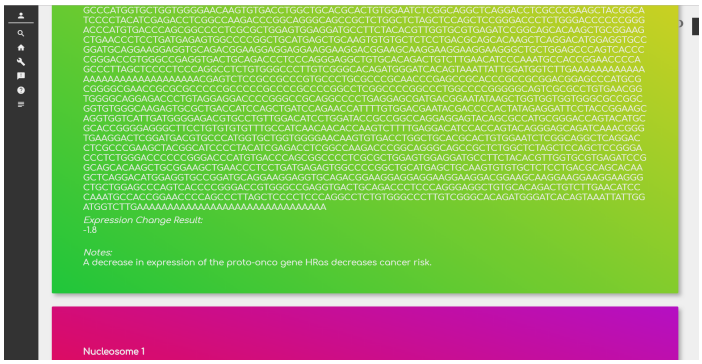
Reference Number	Evidence	Explanation
E		The first screenshot shows the edit form being used to change the disease that the sequence is currently associated with to a different one. The second screenshot shows that the edit was successful as the sequence now displays the name of this other disease.
F		For this test I logged into another account (*John*)The first screenshot shows a new sequence being created and associated with the "Cancer Risks" disease - the same as the previous test. The second screenshot shows both within the same disease page. Both sequences can be seen within the page as HTML blocks. Therefore the test was a success.

Reference Number	Evidence	Explanation
11		The first screenshot shows the edit HTML form that creates a new nucleosome. It contains a DNA sequence and some histone modifications. The second screenshot shows the result after submission. The nucleosome has been created using the inputted data. Therefore a new record was added to the *nucleosome* table. The test was successful.

Reference Number	Evidence	Explanation
12		The first screenshot shows the original DNA sequence and histone modifications for Nucleosome 6. The second shows the new data before submitting the edit form. The third shows Nucleosome 6 having the new data instead of the old. Thus this *nucleosome* record has been updated. The test was successful.

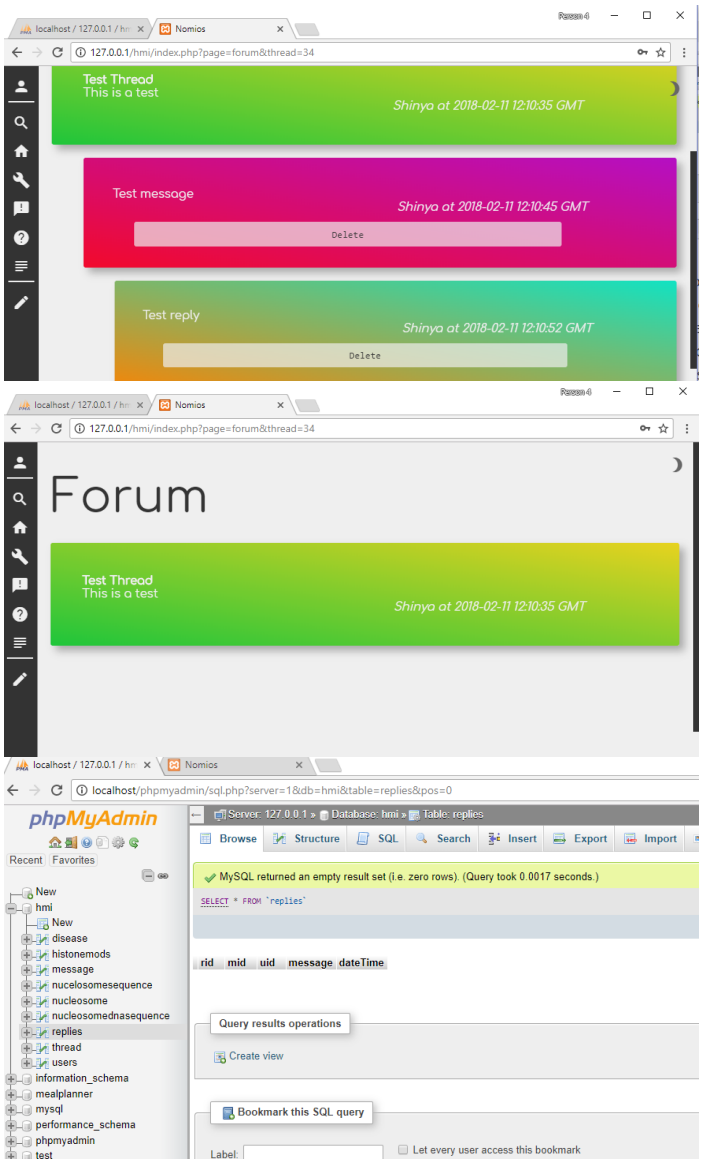
Reference Number	Evidence	Explanation
13		The first screenshot shows a nucleosome before the delete from is submitted (i.e. the delete button). The second shows no sign of this nucleosome after the form was submitted. Therefore the record was deleted from the *nucleosome* table. The third screenshot shows a nucleosome sequence before the delete form is submitted. The final screenshot shows what happens after submission. The *nucleosomesequence* record was removed. Thus this was a successful test.

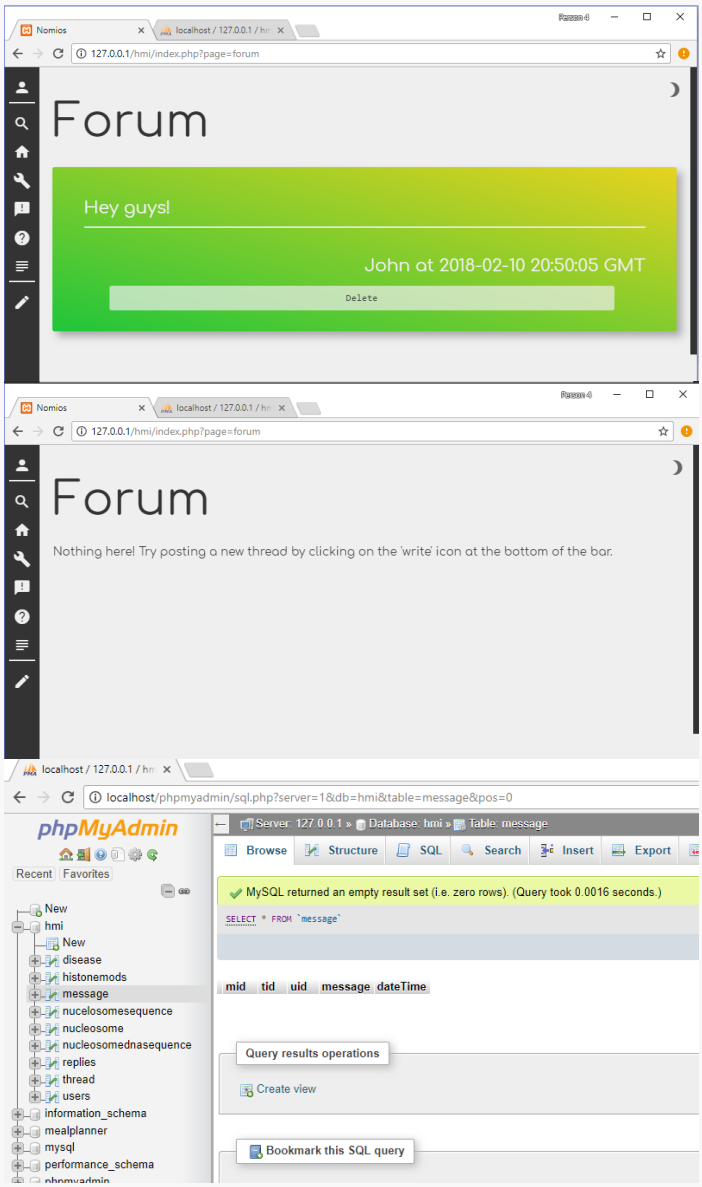
Reference Number	Evidence	Explanation
14		This nucleosome sequence was created by another user. No edit form is displayed. therefore the user cannot perform the same previous three tests on the sequence. the test is therefore successful.
15		The first screenshot shows the current *notes* of a sequence owned by the user. The second shows the edit HTML form with the new data. The third shows the change of the *notes* for this sequence. The fourth screenshot shows the current *notes* for a disease at the top of the screenshot, with the new data in the edit form at the bottom. The last screenshot shows that the disease's *notes* has changed to contain the new data. Therefore the test was successful.

Reference Number	Evidence	Explanation
		
16		This screenshot shows a description of the effect of the histone modification sequence in this nucleosome sequence. A positive result is had. Therefore the test was successful. This is evidence for the histone modification interpreter being accurate.
17		This screenshot shows a result to be negative. This is the expected result. Therefore the test is successful. This is evidence for the histone modification interpreter being accurate.

Reference Number	Evidence	Explanation
18		This screenshot shows the change in expression being positive. This shows an increase in expression. Therefore, the test was successful. This is evidence for the histone modification interpreter being accurate.
19		Notice that these screenshots show the search type as "My * " where * is "sequences" and "diseases" respectively. This means only records associated with the user are being searched through. The first screenshot shows the user searching for a particular sequence. It is found. The second shows the user finding a particular disease - it is a;so found. Both are displayed. the test was a success. The search result is accurate, particularly as the entire *name* of the sequence/disease was not fully inputted. Meaning that specific search results are returned with a smaller character count and are therefore, more efficient.

Reference Number	Evidence	Explanation
1A	 	Notice that these screenshots show the search type as "All * " where * is "sequences" and "diseases" respectively. This means all records are being searched through. The firsts screenshot shows that the specified sequence is returned and displayed. The second shows the same but with a disease *name*. Thus, the test is successful. The search result is accurate, particularly as the entire *name* of the sequence/disease was not fully inputted. Meaning that specific search results are returned with a smaller character count and are therefore, more efficient.

Reference Number	Evidence	Explanation
1B		The "Test Message" message no longer appears after the "Delete" button is clicked. This must mean that it was removed from the *messages* table. There are no replies too, which (as this was the only thread in the database) means that the associated rows in the *replies* table were removed too. This test was therefore **successful** - the admin can delete messages and their associated replies.

Reference Number	Evidence	Explanation
1C		The "Hey guys!" thread is no longer displayed after the "Delete" button is clicked. It must have been removed from the *thread* table. The associated messages and replies have been removed too. This test was therefore **successful** - the user can delete threads and associated messages and replies.

I used three accounts for this testing. This was to perform some of the tests that required it, but also to make my tests more reliable, as the wider range of accounts I use for testing will confirm that the tests work on multiple accounts - particularly as I repeated each of those tests on each account and the result of each test could be seen by each account, proving that different users can communicate and collaborate using *Nomios*. In case the user wished to use *Nomios* using these test accounts I have provided the necessary details to sign in using those accounts:

Email Address	Password
sy@example.com	shinya33
jg@example.com	john62
af@example.com	alex81
ma@admin.com	admin99

Note that ma@admin.com is an admin account.

Nomios - Evaluation

Note that the tests referenced are discussed in my Test Results and Test Plan

Has The Solution Met The Objectives?

Perhaps the best way to address this question is by first reading my objectives from my analysis:

> 1. The user should be able to start threads in the forum so that they can ask questions, further their knowledge and collaborate with other scientists.
> 2. The user should be able to respond to threads and messages within threads so that communication can take place between users.
> 3. The user should not be able to remove any messages that they have posted or threads that they have started. This is to make sure that other users can still access old questions so that they can learn from them too.
> 4. The user should be able to process a sequence of histone modifications on a particular gene **and** receive a result that lets them know what the change in expression will be. This is fundamental to the solution as it lets users predict how much a gene is expressed without conducting expensive lab work.
> 5. The user should be able to edit a DNA sequence so that they can (for example) amend any errors produced and change the current allele to another allele. They should be able to change the sequence of histones to change the resultant expression of the gene.
> 6. The user should be able to describe what a particular sequence does and be able to name a sequence so that they can differentiate sequences from each other.
> 7. The user should be able to view groups of sequences that attribute to a particular disease so that they can collaborate with other scientists on research for a particular disease.
> 8. The user should be able to search for other sequences, sequences within diseases and threads including their own so that they can benefit from other users' data and navigate to their own for their own research.
> 9. The user should have their own account so that the data that they have submitted can be attributed to them. Similarly, they should not be able to submit new data without being signed into an account to ensure that new data is attributed to their account.
> 10. There should be a "help" menu that the user can navigate to so that they can learn to use the website quickly and easily.
> 11. Users should be able to style their forum posts using some form of mark up. This will enhance communication between users as it structures texts to be more easily read and have a greater emphasis on particular words.
> 12. HTML should not be a valid form of text when submitting posts to forums or creating notes and names for particular sequences and diseases. This is to prevent malicious attacks on the website that may distort its intended display.
> 13. An admin should be available in the forum that can delete inappropriate threads, messages or replies. This is to ensure that the forum remains a friendly place to communicate so that users maintain communication and collaboration.
> 14. The result displayed once a histone sequence is interpreted (i.e. the change in expression of a gene) should be accurate to what is observed in real life.

> 15. The website must be viewable on a mobile and desktop device so that it can be accessed by a user no matter their device preference.

We will go through each of these objectives one at a time and evaluate them to find out whether or not we have met that particular objective.

> 1. The user should be able to start threads in the forum so that they can ask questions, further their knowledge and collaborate with other scientists.

This is possible as discovered by test 0. The user has a large set of inputs to ask a wide range of questions and to discuss certain topics. This objective has been met.

> The user should be able to respond to threads and messages within threads so that communication can take place between users.

This is possible as shown by tests 1, 2 and 3. Those tests were successful so this objective has been met. This means that users can converse with each other bout a particular topic or question which allows for more communication via the forum.

> The user should not be able to remove any messages that they have posted or threads that they have started. This is to make sure that other users can still access old questions so that they can learn from them too.

This is possible as shown by test 7. That test was successful so this objective has been met. However, it is difficult for users to navigate to extremely old posts without searching. This is because **pagination** is not implemented (this is where different pages containing different results - in this case threads - are generated).

> The user should be able to process a sequence of histone modifications on a particular gene **and** receive a result that lets them know what the change in expression will be. This is fundamental to the solution as it lets users predict how much a gene is expressed without conducting expensive lab work.

The tests 16, 17 and 18 test this and are all successful. Therefore this objective has been met. However, only naturally occurring histone modification sequences should be processed as these are the only histone modification sequences that are valid (i.e. the pattern of histone modifications adheres to the rules of histone modification sequences).

> The user should be able to edit a DNA sequence so that they can (for example) amend any errors produced and change the current allele to another allele. They should be able to change the sequence of histones to change the resultant expression of the gene.

Test 12 tests this and was successful. Therefore this objective has been met. The DNA sequence and histone modification sequence for a specific nucleosome (and therefore the entire nucleosome sequence) can be altered.

> The user should be able to describe what a particular sequence does and be able to name a sequence so that they can differentiate sequences from each other.

Test D tests this and was successful. Therefore the objective has been met. Using the *notes* section a user can describe what a particular sequence does. Users are prevented from naming two of their own sequences the same. This means that users can differentiate sequences from each other.

> The user should be able to view groups of sequences that attribute to a particular disease so that they can collaborate with other scientists on research for a particular disease.

Test F tests this and was successful. Therefore, this objective has been met. Nucleosome sequences that are associated with the same disease are grouped together on a signal disease page. Scientists can contribute to these sequences so that they can collaborate on the research for a particular disease.

> The user should be able to search for other sequences, sequences within diseases and threads including their own so that they can benefit from other users' data and navigate to their own for their own research.

Tests 19 and 1A test this. They were successful. Therefore, the objective has been met. Users can use the search page to find a specific sequence, thread or disease by name. The results can be further filtered to only return the own user's sequences, threads and diseases.

> The user should have their own account so that the data that they have submitted can be attributed to them. Similarly, they should not be able to submit new data without being signed into an account to ensure that new data is attributed to their account.

Tests A and B test this. They were both successful. Therefore this objective has been met. A user can create an account (test A) and only access the forum and tool pages (and therefore can only submit data) if they are signed into that account (test B).

> There should be a "help" menu that the user can navigate to so that they can learn to use the website quickly and easily.

Test 9 tests this. It was successful. Therefore this objective has been met. The help menu is clearly displayed to the user and is always accessible from any page. Therefore, the user can learn to use the website very quickly.

> Users should be able to style their forum posts using some form of mark up. This will enhance communication between users as it structures texts to be more easily read and have a greater emphasis on particular words.

Test 8 test this and it was successful. Therefore, the objective has been met. The user can style posts using an artificial mark up, allowing them to structure their text and enhance communication.

> HTML should not be a valid form of text when submitting posts to forums or creating notes and names for particular sequences and diseases. This is to prevent malicious attacks on the website that may distort its intended display.

This is tested by test 5. This test was successful. The HTML tags were removed from the submitted text, preventing malicious code from being stored in the database. This prevents malicious client side code from being executed and prevents malicious attacks. This objective has therefore been met.

> An admin should be available in the forum that can delete inappropriate threads, messages or replies. This is to ensure that the forum remains a friendly place to communicate so that users maintain communication and collaboration.

This was tested by test 6 and was successful. Therefore, the objective has been met. Admins can removed inappropriate posts and threads, maintaining a friendly atmosphere within the forum. These posts and threads are deleted, censoring offensive material.

> The result displayed once a histone sequence is interpreted (i.e. the change in expression of a gene) should be accurate to what is observed in real life.

The tests 16, 17 and 18 test this and are all successful. Therefore this objective has been met. The result displayed from an interpreted histone modification sequence is accurate to the real life expectations. Therefore the histone modification interpreter is accurate.

> The website must be viewable on a mobile and desktop device so that it can be accessed by a user no matter their device preference.

CSS code has been programmed which changes the design of the user interface depending on the device used. This could only be simulated however as the website is not live hosted during testing and therefore cannot be accessed by mobile devices. This objective has thus been met.

In What Ways Can The Solution Improve?

One way the solution could be improved is by providing a feature that lets the users search for specific histone modifications and read about their discovery, their effects and any other important information attached to them. This would allow the epigeneticists to quickly find the effects of a single histone modification instead of only allowing them to find out information on entire histone modification sequences. This would also help a user epigeneticists understand another user's nucleosome sequence, therefore encouraging a more open scientific community.

Another way the solution could be improved is by creating a second tool. This tool could be used to predict the effect on expression of artificial histone modification sequences. This would be useful for biotechnology industries that may make use of an artificial histone modification sequence and therefore help researching in epigenetics by reduces time-consumption and resources. This would be significantly more complex to create as there is a complex set of rules that govern which histone modifications are allowed in a sequence with of modifications and which aren't.

A **layer** is a set of posts that respond to the same post. Level 1 for example are all posts that respond to the original message, level 2, to those messages, etc. The forum currently only supports up to level 2. If it expanded to include a much higher maximum level then users could create a much more complex thread which simultaneous conversations occurring at different levels in response to different posts. This would enhance communication greatly as more communication can take place at the same time. For example, a variety of solutions could be suggested for a particular problem that was posted. So when a later user visits the thread they have a much better chance at finding the actual solution for that problem.

What Does The Target Audience Think?

After using he forum Henry wrote:

> I love the use of the artificial mark up! I feel that it structures the text well so that a point can be put across well to another user. However, it is limited to only certain tags (for example, `[b][/b]` for emboldening text).

If I were to improve this solution I would add in extra mark up tags. This could be including different types of lists and hyperlinks. This could open up the website to external services and websites, such as *Wikipedia* pages for sources for answers within the forum. This would encourage more accurate answers to question within the forum.

After using the histone modification interpreter, Luke said:

> The help guide is incredibly useful as it can be accessed anywhere on the website straight from the menubar. The nucleosome sequences are broken down into a nucleosomes which are each displayed too. However, I find it difficult to copy and paste DNA sequences to other applications - I copy other texts on the web page. This is because no highlighting is shown to the user when selecting text to copy. This should be addressed.

If I were to improve this solution I would change the CSS code used so that when text is selected it is highlighted. This will show the user what text they're selecting. This prevents them from accidentally selecting other, unwanted text or even images. This lets users copy and paste DNA sequences to other applications, which was one of the demands outlined by my target audience in my *Analysis*.