Technological Feasibility Analysis

GNomes

Members
Jacob Christiansen
Allen Clarke
Yuanyuan Fu
John Jackson

Mentor
Mahsa Keshavarz

Clients
Dr. Toby Hocking
Christopher Coffey

November 5, 2019

Contents

1 Introduction **2**

2 Technological Challenges **3**

3 Technological Challenges **3**

4 Technology Analysis **4**

4.1 Selecting a genome browser

4.1.1 Metrics 5

4.1.2 Alternatives

4.1.3 Comparison and chosen approach 8

4.1.4 Proving feasibility

4.2 Selecting a database

4.2.1 Metrics 9

4.2.2 Alternatives

4.2.3 Comparison and chosen approach 11

4.2.4 Proving feasibility12

4.3 Optimal UI framework 12

4.3.1 Metrics 12

4.3.2 Alternatives 13

4.3.3 Comparison and chosen approach 15

4.3.4 Proving feasibility 15

5 Technology Integration **16**

6 Conclusion **17**

1 Introduction

Team GNomes is working with Dr Toby Hocking to create PeakLearner, a machine learning web app to process genomic data. The problem is that we do not understand genetic diseases. To really learn about these, a massive amount of data would have to be analyzed by biologist, charting what genes are active in patients with diseases by hand as there is no automatic tool to do this for them as of right now.

Our solution is to implement a machine learning algorithm that will automatically go through this information to predict what genes are being expressed. To make this as easy as possible to learn for biologists, we will be building our solution on top of an existing genome browser, a common piece of software for biologists researching genomic data.

Biologists' usual approach to this problem is to hand-draw labels on histone concentration charts (named for the protein responsible for gene expression) to mark areas of high activity—"peaks"—in a genome. This method is too slow and inefficient as there is an enormous amount of data to sift through for even a single sample.

The solution Dr Hocking envisions is a quick, easy-to-use web application for adding, examining, and editing peak-predicting models. To create this, we are breaking the problem into three distinct subproblems which we will cover at length in this document.

First, we will select a genome browser, is supplying an easy interface to biologists, to solve this we are building our modeling in to a known tool called a genome browser. We must be able to add a new plugin to one of these interfaces to show our own information.

Our second task is to select a database for storing peak-models. This model will need to update whenever a user updates an existing label as well as add a new label. To create a system that can update as quick as possible, we will be calculating new models on the NAU Monsoon High Performance Computing (HPC) system. Not only that but the goal is to also have a database to hold several models close to what the user is currently seeing and access those before calculating new ones when the user makes an edit.

The final section, selecting a framework, is concerned with choosing a Javascript front-end system that can communicate with Monsoon and the database, integrate with our chosen genome browser, and quickly render peak predictions to the screen.

2 Technological Challenges

Our project is specifically about bridging the gap between scientists and statistics through machine learning. We are trying to show scientists' DNA data in a way that is easy for them to mark significant areas in the graph so that the system can learn regions of interest and generate models to predict the data. Currently, genome browsers can view but not modify the data, so it is viewed in a browser and edited by hand. Starting with the major challenges, we need to decide on what type of genome browser, database, and framework we will be using.

- *Genome browser.* This display that users (scientists, researchers, doctors) will be looking at needs to be incorporated into one of the known genome browsers that users are already comfortable with, so they do not need to learn another tool.
- *Database.* Another challenge will be storing a database full of these models as we should not have to rebuild a model from scratch if we have previously calculated a model that would fit the data provided. We should have a handful of models ready to use and calculate new ones for regions of interest if the existing models do not fit.
- *Framework.* Additionally, they will need a framework that allows us to be able to use a mouse to add labels to their data to mark regions of their sequencing data that contain a PeakStart, PeakEnd, Peaks, or NoPeaks. This will generate a model for the data to learn where peaks are in a given genome, which will be difficult to display.

3 Technological Challenges

Our project is specifically about bridging the gap between scientists and statistics through machine learning. We are trying to show scientists' DNA data in a way that is easy for them to mark significant areas in the graph so that the system can learn regions of interest and generate models to predict the data. Currently, genome browsers can view but not modify the data, so it is viewed in a browser and edited by hand. Starting with the major challenges, we need to decide on what type of genome browser, database, and framework we will be using.

- *Genome browser*. This display that users (scientists, researchers, doctors) will be looking at needs to be incorporated into one of the known genome browsers that users are already comfortable with, so they do not need to learn another tool.

- *Database*. Another challenge will be storing a database full of these models as we should not have to rebuild a model from scratch if we have previously calculated a model that would fit the data provided. We should have a handful of models ready to use and calculate new ones for regions of interest if the existing models do not fit.

- *Framework*. Additionally, they will need a framework that allows us to be able to use a mouse to add labels to their data to mark regions of their sequencing data that contain a PeakStart, PeakEnd, Peaks, or NoPeaks. This will generate a model for the data to learn where peaks are in a given genome, which will be difficult to display.

4 Technology Analysis

In this section we will focus on three major design decisions that face us in the beginning stages of our project. These are: choosing a genome browser to incorporate into our web app, choosing a database to store models, and choosing a UI framework for our site. In each case, we will analyze the tools under consideration, evaluating them according to specific metrics before committing to a decision. (All metrics are on a scale 1–5.)

4.1 Selecting a genome browser

Gene recognition, which uses biological experiments or computers to identify biologically-characterized fragments on DNA sequences, is the basis of genomic research. Genomic browser technology is of great significance in gene sequence analysis, genetic code interpretation, and complex disease research. We have two prospective genome browsers; one is JBrowse, and the other one is UCSC Genome Browser (hereafter UCSC).

4.1.1 Metrics

We will discuss these two genome browsers, evaluate them on the basis of **visual content**, **visualization**, and **software system architecture**, and select one to use.

- *Visual content.* To make sure the browser is clear and easy for users to work on.
- *Visualization.* To make sure it can display all the useful information.
- *Software system architecture.* To determine if it is compatible and code is available.

4.1.2 Alternatives

UCSC and JBrowse are two of the most widely-used genome browsers, and each is distinct in its feature profile and its visual data types and visualization methods.

The reason for choosing UCSC as one of the options is because UCSC Browser is a graphical viewer optimized to support fast interactive performance and is an open-source, web-based tool suite built on top of a MySQL database for rapid visualization, examination, and querying of the data at many levels [1].

And the reason for choosing JBrowse as one of the options is because it is a genome browser with a fully dynamic AJAX interface, being developed as the eventual successor to GBrowse. It is very fast and scales well to large datasets. JBrowse is javascript-based and does almost all of its work directly in the user's web browser, with minimal requirements for the server [2]

Below, we introduce each browser, describe its advantages and disadvantages, and assign a grade to it for each of our metrics.

1. UCSC Genome Browser. UCSC is a powerful genome browser developed by the University of California, Santa Cruz, which is used to browse the genome and view genomic annotation information. It does not make any conclusions by itself, but only provides users with reference information. UCSC is currently used extensively throughout the world, and many other genomic database projects, such as Ensembl, use its genomic sequence data [3].

    - *Pros*: UCSC Genome Browser is a very comprehensive genome browser with more species and more complete visualizations. It can generate genomic annotation images on the server and then download the PNG image to the web browser.

- *Cons*: The development of UCSC Genome Browser is still in the era of web1.0, and the new generation of web technology and partial refresh technology are not fully applied. Therefore, some program-driven scripts in UCSC have certain restrictions on user access frequency.

Table 1: An evaluation of the UCSC Genome Browser

	Visual Content	Visualization Methods	Software System Architecture
UCSC	4	3	2

- **Visual content**. After entering the system homepage, you can search for a gene based on gene name, keyword, etc., or you can directly query based on the chromosome or nucleotide base range. By scaling, the user can view the gene density of each region of the entire genome from a macroscopic view, as well as microscopically view the genetic information in a sequence region. Researchers can add their own annotations from a scientific or educational perspective. There are four main useful functions of visual content, so we give 4 points in this part [1].
- **Visualization**. UCSC presents information in the form of a track. The track is expressed as a horizontal or vertical strip. Different areas on the strip represent different biological meanings by different colors, lines, squares, etc. The main interface of the system can be divided into three parts from top to bottom: search query, visualization and track management. These are three useful functions, but it operates with obvious stagnation, so we give 3 points for this metric [4].
- **Software system architecture**. The main development languages of UCSC are Java, Python, and C. The background database depends on mysql, this is conflicted with our choices of the database which is a serious drawback. The pros include that UCSC can be well compatible with mainstream web browsers such as IE, Chrome, and Firefox. UCSC is open source and supports multiple development languages, which is good.But its storage database only supports mysql, which conflicts with our selection in the database. So we just give a score of 2 points for the first two advantages [5].

2. JBrowse. JBrowse is an open-source genomic browser that applies next-generation web technologies. Its main purpose is to apply advanced local refresh technology in tools such as Google Maps to the genome browser for a smooth visualization [2].

    - *Pros*: JBrowse belongs to the next generation of genome browsers and is based on the latest web technologies. In JBrowse, the server-side load is greatly released, and the back-end server only needs to send static files to the browser client. From the complicated calculation work, a lot of calculation work is assigned to the front end. At the same time, HTTP cookies technology has also been well applied to effectively record user preferences.
    - *Cons*: Although JBrowse puts the visualization work on the browser side, its visualization method is just some ordinary HTML markup. The front end needs to run a lot of script code when drawing the image, and we have to consider the browser to the new tag in HTML5. Compatibility issue.

Table 2: An evaluation of JBrowse.

	Visual Content	Visualization Methods	Software System Architecture
JBrowse	3	4	4

- **Visual content**. JBrowse can display the overall view of the genome, as well as display gene span, tRNA, transposon, oligonucleotide, protein binding site, enhancer, gene regulatory region, non-coding RNA, point mutation, sequence variation information, etc. Users can upload content that needs to be visualized and support files in various formats such as GFF, GFF3, WIG, BED, FASTA, Wiggle, BigWig, and BAM. It can satisfy our basic requirements and it supports files in various formats. But it only provides the genome information of humans, so we cut 2 points off and give 3 points in this part [2].
- **Visualization**. JBrowse also visualizes track(Figure 2), providing smooth dynamic movement and zooming, as well as navigation and channel selection. JBrowse can display a variety of track views; in addition to the basic view, users can display untranslated areas, exons, intron structures, and so on. There are four main useful functions, especially that it operates smoothly. So we give 4 points in this part [2].

- **Software system architecture**. JBrowse is open source. The main development languages are Javascript and HTML5. The Javascript API is provided externally, and the data is formatted by Perl. The front-end browser requirements are to support new tags in HTML5 (such as canvas). The latest release version of the browser requirements are Firefox10 + , Safari5 + , Chrome17+ and IE9+; the background server is mainly stored with json files. Since json itself is a subset of Javascript-based, it can be seen as objects and arrays in javascript, so faster front-end communication can be achieved. It is relatively fast which is important for us because we focus on improving the speed. But it needs to consider browser compatibility. So it was reduced by only one point [6].

4.1.3 Comparison and chosen approach

Table 3: An evaluation of Berkeley DB

	Visual Content	Visualization Methods	Software System Architecture
UCSC	4	3	2
JBrowse	3	4	4

As seen in Table 3, we've determined that **JBrowse** offers the best performance according to our analysis. First of all, it is compatible and user friendly. Although it requires the latest HTML and web browser, it is in common use. Also, its interface looks clean and operates smoothly.

4.1.4 Proving feasibility

We will prove that JBrowser is the best genome browser, that it is intuitive for our labeling to add to, and that it will provide better user experience and support files in various formats for users to submit in.

4.2 Selecting a database

We have large amounts of data that needs to be stored, including various parameters for potential data models, how well a model fits a set, the labels applied to a model, and the data in which we are modeling. Storing this data in a database would be ideal as long as we can get it back quickly enough to not affect the user from updating their data with labels. These models that would be stored need to be updated as quickly as possible should a user add any labels to the data.

4.2.1 Metrics

We are grading on **speed** to make sure our model is fast enough. We are scoring on **ease of use** and **ability to hold data** because the chosen database needs to be able to contain our model and be something that we can learn and maintain.

- *Speed.* We are grading on speed to make sure our model is fast enough to update as soon as a new label is added. The speed was tested by recording the time it took to upload the data, and then by reading speed comparisons online of the two systems [7].
- *Ease of use.* If the system is not easy to use and maintain, then we will not be able to update it easily as new models are generated by the Machine Learning Algorithm.
- *Data Handling.* If the database is to be a valid long term solution, it should be able to hold the data necessary to form a model.

4.2.2 Alternatives

We have found a few database systems that have the potential to be used in the system. The two databases we will consider are PostgreSQL (PSQL) and Berkeley DB (BDB). We chose these because PSQL is capable of being less structured than a standard database and BDB is designed for optimizing speed.

1. PostgreSQL. PostgreSQL was our initial thought because it is similar to any other relational database but can hold JSON data types to contain blobs of data. PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance [8].
    - *Pros*: The pros include PSQL's ability to use a traditional relational database system and use standard SQL queries. PSQL can also handle JSON objects capable of holding any data type. It is also scalable (able to handle very large amounts of data) which will be necessary given the potential size of a bigWig file.
    - *Cons*: JSON objects that can hold the abstract data is a relatively slow form of structured data when querying. As the number of stored models grows and a label on a dataset is updated, every

single model will need to be updated to see which one is the optimal fit for the data. These updates are the largest area of concern when trying to optimize the speed of the database.

Table 4: An evaluation of PostgreSQL.

	Speed	Ease of use	Data handling
PostgreSQL	3	4	5

- **Speed**. PostgreSQL scored a 3 of 5 on speed because as the tests grew larger and more complex, involving multiple layers of JSON objects, the query times became longer. The speed was deemed to be average in comparison to other Databases based on online results, and thus scored a 3.
- **Ease of use**. This database scored a 4 in Ease of Use because of its relational schema and SQL language, earning it 2 of 2 points. The documentation can be sparse and confusing at times, which is why it lost one point here scoring 2/3 in documentation and combining for a score of 4.
- **Ability to handle data**. PSQL can handle JSON object capable of holding any datatype. It is also scalable to handle very large amounts of data which could be a possibility given the potential size of a bigWig file. The Data handling scored a 5 as this was an all or nothing category and PostgreSQL is capable of holding an array, object or value we may need in the future.

2. Berkeley DB. Berkeley DB was suggested by Dr. Hocking as it is what he used in his previous research project to make sure data was received fast enough. With that in mind, we decided to look into it to see how it compared to a more known database system. Berkeley DB is a family of embedded key-value database libraries providing scalable high-performance data management services to applications. The Berkeley DB products use simple function-call APIs for data access and management. It enables the development of custom data management solutions, without the overhead traditionally associated with such custom projects [9].

    - *Pros*: Speed was the biggest advantage of this database system. The data is stored as key-value pairs and very easy to take out as a value and turn into the necessary object. When running it against the PostgreSQL database, both an insert and a select

statement were completed faster. The entire database is embedded and capable of storing any Python objects, so it can store anything we can possibly put in it. This is largely due to the existing bsddb3 Python library and modifications to the SegAnnDB code written by Dr. Hocking for a previous project, which helps store only the necessary information instead of the entire file [10].

- *Cons*: Berkeley DB, as a NoSQL database, lacks the type of relational system that many of us are familiar with using.

Table 5: An evaluation of Berkeley DB

	Speed	Ease of use	Data handling
Berkeley DB	5	3	5

- **Speed**. Speed was the biggest advantage of this database system. The data is stored as key-value pairs and very easy to take out as a value and turn into the necessary object. When running it against the PostgreSQL database, both an insert and a select statement were completed faster. After running a few tests on this database, it was clear that the speed was significantly faster than the previous database, as shown below. Because the speed was faster, BDB scored a 5 in speed
- **Ease of use**. The ease of use was not as apparent since the documentation was lacking for BerkeleyDB in a similar fashion to PostgreSQL, scoring 2/3 for documentation. Since the database is not a traditional relational database, and lost one point in that category, combining for a score of 3 in the ease of use.
- **Ability to handle data**. The system could handle all python objects as data types, which is exactly what we need to store, scoring a full 5 pts in the Data Handling category, which is reflected in the table below.

4.2.3 Comparison and chosen approach

Table 6: A comparison of two database systems.

	Speed	Ease of use	Data handling
PostgreSQL	3	4	5
BekeleyDB	5	3	5

PSQL's speed was suboptimal as we began to update many rows, especially when JSON objects were involved. The SQL language was easy to use, but the documentation for setting up a database was sparse and hard to follow at times. The data handling however, was perfect PostgreSQL uses a relational database system that we are familiar with and we can always store objects inside of a JSON blob should they not fit into a conventional column.

After running a few tests on this database, it was clear that BDB's speed was significantly faster than the previous database, as shown in Table 6. This score of 5 in speed by BDB is the reason we are choosing to use it over PostgreSQL. The ease of use was not as apparent since the documentation was lacking for BDB and the entire database is more complicated to use. Lastly, the system could handle all python objects as data types using the previously mentioned library and by adapting Dr. Hocking's existing code, which is also reflected in the table.

4.2.4 Proving feasibility

We are going to show that the BerkeleyDB database is perfect for storing the data models in a similar manner to Dr. Hocking's current SegAnnDB, which is a BDB system that stores data models to detect breaks instead of peaks. We will prove the speed is adequate by showing that a new model is applied to the data as soon as new labels are applied. We are currently determining exactly how fast is appropriate for the model to update. We will demonstrate this by showing a model on a dataset, applying labels to it and observing the model update to fit the new labels. in a reasonable amount of time.

4.3 Optimal UI framework

PeakLearner is in essence a web application, so we will need a Javascript frontend that is conducive to fulfilling the requirements of our project. In order to create a dynamic, maintainable, and easily modifiable website, one needs to use a Javascript framework.

4.3.1 Metrics

Since we want model updates to render to the page as quickly as possible, our primary concern is a framework that is **fast**. Additionally, we want a framework that is **easily-usable** (programmer-friendly) and adequately **functional**. We made judgments by reading articles about how well the frameworks fared in comparison with one another.

- *Speed.* Speed is how fast a framework renders changes to the DOM. We want to minimize latency so that users get fast results [11].
- *Ease of use.* None of us are expert front-end developers, so this metric is an estimate of the simplicity of learning the framework in question [12].
- *Functionality.* Frameworks and libraries differ in how many options and components they provide, as opposed to how many have to be imported from third parties. This metric is a measure of the amount of options the framework gives a developer [13].

4.3.2 Alternatives

The three most popular frameworks for professional projects are Angular, React, and Vue.js. While React is more properly called a "library"—that is, a collection of reusable code for solving UI problems, but which doesn't structure the UI itself—for simplicity, we will refer to it as a "framework." Although we believe each would be capable of satisfying the needs of our project, a substantive examination of each framework is in order before we commit to a final decision.

1. Angular. Angular is part of the MEAN stack of JS-supporting systems. It was developed by Google, and is used by many large companies such as YouTube and Apple [14].

    - *Pros*: Angular uses the MVVM architectural pattern, which allows allows two programmers to work on different portions of the code at the same time, using the same data.
    - *Cons*: Angular's large size makes it unsuitable for projects that demand quick performance. It also has a steep learning curve. It updates with extreme frequency, making it a challenge to keep one's knowledge current.

Table 7: An evaluation of Angular.

	Speed	Ease of use	Functionality
Angular	3	2	5

- **Speed**. Angular's status as a fully-equipped framework is sure to be advantageous for some projects, but its large size causes a slower performance, making it unsuitable for our project.
- **Ease of use**. Angular has a steep learning curve, and there are many concepts to learn before one can begin building a UI.

- **Functionality**. Angular has a built-in heavy library of components, so it doesn't require as much customization as React.

2. REACT. Though more properly described as a "library," React is thought of as a rival to Angular and holds an even larger market share. It's maintained by Facebook and is used by many successful startups including Airbnb and Uber [15].

   - *Pros*: React has a huge community of users and a wealth of packages, tools, and tutorials.React is easy to learn. It uses JSX, which is similar to HTML. It's highly modular and flexible, socode can be reused in other parts of a project.
   - *Cons*: Since React is a library, it offers little in the way of predefined structures. Programmers will have to make more decisions about how to structure the UI and manage the application they would with a fully-featured framework like Angular.

Table 8: An evaluation of React.

	Speed	Ease of use	Functionality
React	5	5	3

   - **Speed**. React, as a lightweight library, is tied for the quickest performance among the three frameworks, so it scores a 5.
   - **Ease of use**. React uses JSX, a scripting language which is similar enough to HTML that it shouldn't present problems in programming. In addition to learning JSX, one must learn to write components, manage internal state, and use props for configuration.
   - **Functionality**. React, as a JS library, offers only the essentials. It must be combined with other libraries to build a complete UI.

3. VUE.JS. Vue.js has a much smaller market share than the other two frameworks and is maintained by a smaller team. Its developer, who worked on Angular, wished to retain the parts of Angular he liked and discard the rest [16].

   - *Pros*: Vue.js is newer framework that is quickly growing in popularity and reputation. Like React,Vue uses a scripting language that's similar to HTML. It's well-suited to large-scale applications.
   - *Cons*: As a smaller project with a lower market share, Vue.js has fewer resources than the other two frameworks under consideration.

Table 9: An evaluation of Vue.js.

	Speed	Ease of use	Functionality
Vue.js	5	4	4

- **Speed**. Vue.js, a lightweight framework, is at least as fast as React.
- **Ease of use**. Vue.js's learning curve is roughly equal to React's, and much lower than that of Angular. And, what it lacks in community-shared knowledge it makes up for with ample documentation.
- **Functionality**. Like React, Vue.js is lightweight and focuses on the core part of the library. Programmers will need to customize the UI to a greater degree than they would with Angular.

4.3.3 Comparison and chosen approach

Table 10: A comparison of three JS frameworks.

	Speed	Ease of use	Functionality
Angular	3	2	5
React	5	5	3
Vue.js	5	4	4

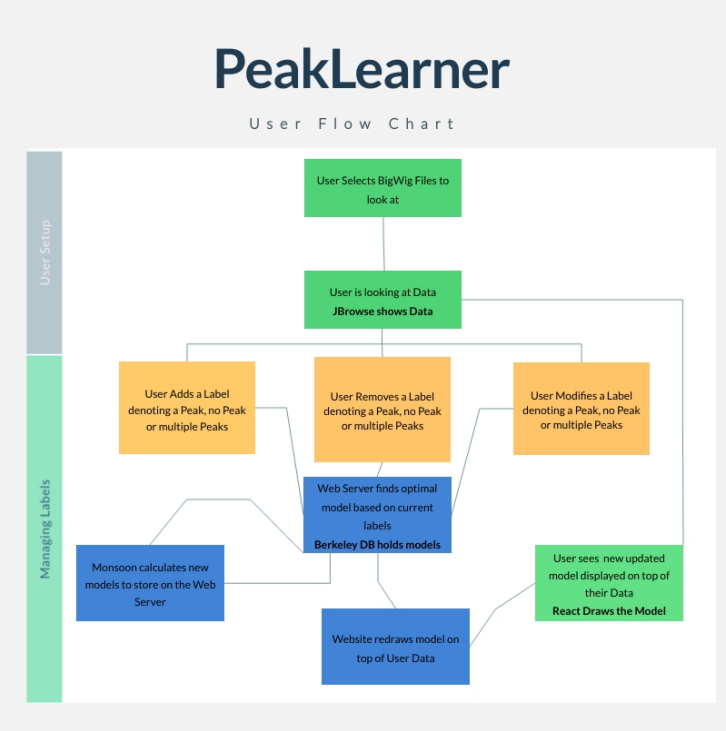
As shown in Table 3, we have elected to use React for our web app frontend. Since Angular has the slowest rendering speed among the three, we could eliminate Angular from consideration. React and Vue.js perform about equally by our metrics, so the deciding factor was that our team has more collective experience with the former. Therefore, we chose **React**. Even though the functionality is slightly improved by Vue.js, we felt as though the improved ease of use of React was the reason we chose to use it.

4.3.4 Proving feasibility

This is only a tentative conclusion. More research, prototyping, and discussion (both among ourselves and with our client) is necessary before committing to React. We will prove feasibility by using React to generate the data held in the database for the tech demo. Ideally we will show that react is capable of displaying data and creating a simple, intuitive user interface for scientists to use. There is no better way for us to show React is usable than by using it to create a demo.

5 Technology Integration

The first hurdle that we will cover in this section is that of usability. This includes the issues of adding labels to the data to edit the generated model as well as how biologists will be able to quickly pick up and use this information. Both of these issues are solved by the choice of genome browser, JBrowse. JBrowse is already a widely used genome browser, it also has built in devtools for allowing custom click operations to happen inside of our code. This will allow our project to be able to be picked up and used by biologists as quickly as possible.



The second problem is that of space. Bigwig files can approach multiple gigabytes in size, and if we were to store them all, we would quickly run out of space. To handle this we will keep only the samples currently being viewed in the database and, when we need them to compute the model, access them very quickly with binary searches.

Lastly, we examine the issue of speed. For this we have to consider the database, server code, and access to Monsoon. We have determined that

NoSQL on the Berkeley DB database will be the quickest combination not to bottleneck from Monsoon's output. For the frontend, we've selected the React JS framework so that at no point is information being held back from the user.

6 Conclusion

Our project addresses a vacuum in the inventory of tools available to genomic researchers; there is too much genomic data and not a good-enough tool to process it to learn what genes influence genetic diseases. To create a web app for this purpose, we will implement a machine learning algorithm across NAUs Monsoon, and we will show the data in the form of a known genome browser biologists already use.

Table 11: A summary of our conclusions.

Challenge	**Proposed Solution**	**Confidence**
Genome Browser	JBrowse	4
Database	BerkeleyDB	5
JS Framework	React	3

This document has weighed the feasibility of such a web app. For a genome browser, we have selected JBrowse, because of its large array of documentation and its wide use in the real world. We selected Berkeley DB for our database as it was shown to be the fastest of any we could find. For our framework we are confident that React will give a quick and easy implementation of all other needed parts of this project. Our solutions to each problem are shown in Table 11. We determined the confidence level of each choice by sitting down as a group and reviewing our own experience with each subject as well as running short tests between each choice.

With the power that BerkeleyDB and Monsoon offer, we are positive we will be able to provide our intended features quickly and without hold-ups. Since JBrowse is widely used in industry, we believe many biologists can quickly move to using our product without too much inconvenience. Lastly, the React framework should supply us with a simple-to-implement architecture to join these parts together. These solutions will build a sturdy foundation for the implementation of our product. Our solution will give biologists the information they need in order to learn more about these diseases that affect and harm so many people throughout the world. This tool will eventually lead to cures and new treatments. Overall, we are certain PeakLearner will give biologists a new genomic tool that is valuable and easy to use.

References

[1] Wikipedia contributors. Ucsc genome browser — Wikipedia, the free encyclopedia, 2019. [Online; accessed 3-November-2019].

[2] GMOD contributors. Jbrowse, 2016. [Online; accessed 3-November-2019].

[3] Genome Bioinformatics Group, 2018. [Online; accessed 3-November-2019].

[4] D. Karolchik, A. S. Hinrichs, and W. J. Kent. The ucsc genome browser. *Current Protocols in Bioinformatics*, 28(1):1.4.1–1.4:26, December 2009.

[5] Jian Shu. Genome browser compare list, 2019. [Online; accessed 3-November-2019].

[6] M. Skinner, A. Uzilov, L. Stein, C. Mungall, and I. Holmes. Jbrowse: A next-generation genome browser. *Genome research*, 19:1630–8, 08 2009.

[7] Integrant. When to use sql vs nosql, 2018. [Online; accessed 3-November-2019].

[8] PostgreSQL. The world's most advanced open source relational database, 2019. [Online; accessed 3-November-2019].

[9] Oracle. Oracle berkeley db, 2019. [Online; accessed 3-November-2019].

[10] T. Hocking, A. Williams, and A. Shrivastava. Seganndb, January 2019. [Online; accessed 3-November-2019].

[11] Stefan Krause. Keyed results, 2018. [Online; accessed 4-November-2019].

[12] FusionCharts. Top javascript frontend frameworks comparison in 2018, 2018. [Online; accessed 4-November-2019].

[13] Daityari S. Angular vs react vs vue: Which framework to choose in 2019, 2019. [Online; accessed 4-November-2019].

[14] Angular. Features and benefits, 2019. [Online; accessed 4-November-2019].

[15] React. Getting started, 2019. [Online; accessed 4-November-2019].

[16] Vue.js. Documentation, 2019. [Online; accessed 4-November-2019].