Bilkent University

Department of Computer Engineering

Senior Design Project

LIBRA: Genetic Filtering and Diagnosis Matching System

Final Report

Mahmud Sami Aydın, Berke Egeli, Naisila Puka, Halil Şahiner, Abdullah Talayhan

Supervisor: Can Alkan
Jury Members: Abdullah Ercüment Çiçek and Hamdi Dibeklioğlu

Final Report
May 27, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1 Introduction

Many hospitals, laboratories and medical research centers want to collect and store genomic data, as well as explore and interpret that data based on specific needs. There are open-source programs developed and utilized for such purposes that have been accepted by the authorities [1]. Yet, they are not suitable for direct use and the installation requires specific expertise.

The collective data produced by these institutes possess valuable information related to genetic profiles. These genetic profiles can be useful for comparing and diagnosing rare diseases. For example, a child in San Francisco Bay Area had a rare disease caused by not being able to produce tears. The doctors were suspicious about a gene called NGLY1 after the results of genetic profiling. Yet, they were not sure about the cause because of the lack of genetic profiles of other patients for comparison. The issue was resolved after finding a patient with similar phenotypes studied by Duke University and NGLY1 was indeed the gene causing the disease [2]. Examples like this have created a high demand for genomic discovery through the comparison of genotypic/phenotypic profiles.

The aim of LIBRA is to provide a user-friendly genetic filtering and annotation system equipped with a genetic profile matching platform, that can be quickly integrated and easily used by medical institutions in order to explore genetic variation, detect and diagnose rare diseases, as well as safely collect and store their data.

2 Overview

LIBRA is a platform independent, web based genetic annotation and matchmaking tool. Its main objective is to combine and improve upon features and limitations of other genetic annotation or matchmaking tools like Gemini, Varapp and MatchMaker. It consists of two main modules: The Genetic annotation and filtering module, and the matchmaking module.

When a user opens up LIBRA in their browser they will be directed to the landing page where they will have the option sign in or sign up to our application. After signing in, the users can create new projects and upload VCF files to the system. Users can select if they want to upload a VCF file for a patient in the system. They can also upload multiple VCF files to the same project. If no patient is selected during this process the user will have the option to create a new patient and it will be associated with the VCF file being uploaded. After the upload is complete, the VCF files will be annotated in the backend asynchronously. Once the annotation process is complete

the users will be able to filter the annotated VCF file based on frequency, impact and scenario of variants.

Users will be able create and manage patients in the My Patients page. Users can edit the name, diagnosis and phenotype of the patients if they wish to. If a VCF file associated with one of the patients is uploaded to the system then after the completion of the annotation, genotype of the patient will be automatically filled by the system.

Users can edit their profile to change their personal or contact information and the phenotype/genotype similarity thresholds for above which they want to be notified through an email after matching occurs in the matchmaking module.

The matchmaker module matches different patients based on the cosine similarity of their phenotype. After matchmaking is complete, the users will receive a report mail automatically generated by the system that shows the phenotype and genotype similarity of the matched patients. After the matching occurs, the system gives the option to the user to contact with the doctor responsible for the other patient.

3 Requirements Details

3.1 Functional Requirements

3.1.1 User Accounts

- Everybody will be able to sign up to LIBRA.
- Users will be able to edit their contact and personal information, and set thresholds for phenotype/genotype similarity for above which they want to receive an automated report email after matchmaking process if there is a match.

3.1.2 Genetic Variant Upload and Automatic Annotation

- Users will be able to load genetic variants of their patients of interest to LIBRA using a specific format (VCF).
- After uploading, the annotation will take place in the backend asynchronously.
- Users will be able to upload multiple VCF files at the same time.

- Users will be able to select the patient who is associated with the VCF file through a list of patients.
- If no user is selected, a unique default user will be created and associated with the file.
- The genetic variants will then be automatically annotated by using the SnpEff annotation tool.

3.1.3 Annotation Based Filtering

- Users will be able to filter annotated VCF results based on a set filters. These filters select tuples based on scenario, impact and pathogenicity of variants.

3.1.4 Patient Profiles

- Users will be able to create accounts for their patients in LIBRA with the name of the patient.
- Users will be able to enter the phenotypes of the patient to the system.
- Users will be able to enter the diagnosis they put on the patient into the system.
- After the annotation process, the genotype of the patient will be automatically updated by the system.

3.1.5 Patient Matching

- Users will be able to search for similar patients based on the phenotype of the patients.
- Users will be notified through an email if a match occurs in the system and phenotype/genotype similarity is above the threshold set by the user.

3.2 Non-functional Requirements

3.2.1 Availability

The application should be accessible to users at all times, with only the possible exceptions of server maintenance.

3.2.2 Portability

The system should be compatible with different browsers. This means that LIBRA will be developed as a platform independent web application.

3.2.3 Sustainability

In order to increase the capacity of our system to endure through time, we are mainly focused in the underlying database. The more scalable the database is, the more sustainable it will be when compared with the exponential data growth (in particular genomic data) within the years. Also, updates according newly released versions will contribute in sustainability of the system.

4 Final Architecture and Design Details

The LIBRA system is designed as a client-server architecture. Client side is a web application accessed through a web browser which allows users to create projects, upload VCF files, create patients, select specific patients for the projects and uploaded VCF files and initiate matchmaking. Server side handles most of the computation like annotating the uploaded VCF files, querying the database for the selected filter options, calculating the similarity results for the matchmaking process, handling authentication and data management. This architecture is merged with the classic Model-View-Controller/Logic software design pattern, as will be explained in the following subsections.

4.1 Full Diagram

In this section the full diagram of the system is shown, along with its packages according to the server-client model, and detailed classes with their attributes and operations.

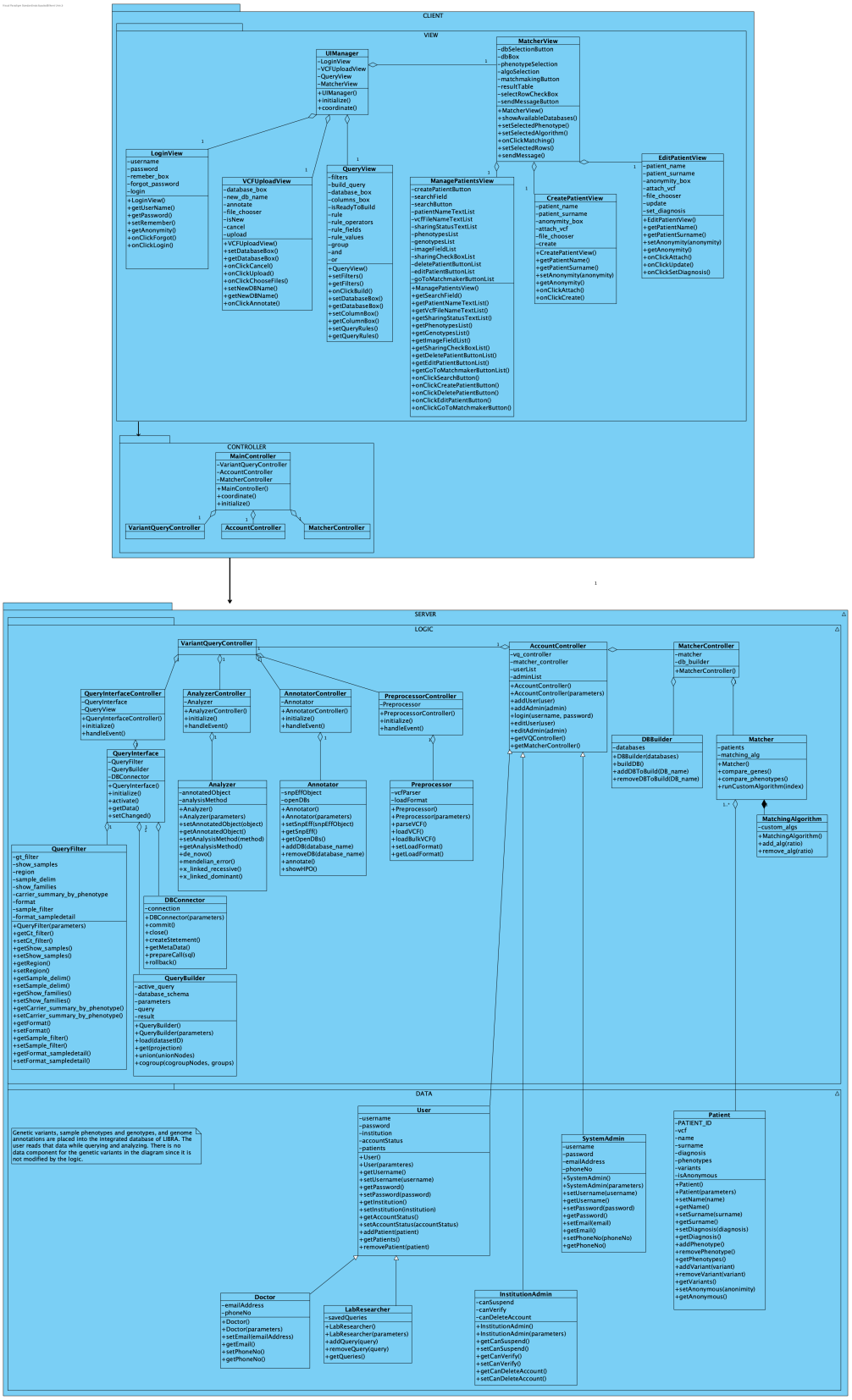


Figure 1: Full Diagram of the System of LIBRA

4.2 Client

The client of LIBRA is the user interface layer of the general application, which can be accessed through the web, as the project is compatible with many browsers. This subsystem handles the interaction between the view and controller components of the whole system. The controller accepts input and converts it to commands for the view. As usual, the view component contains user-friendly UI elements for the user to interact with LIBRA. The controller component handles the communication of user's requests in the UI with the functionalities of the system. This is achieved by interaction with the server subsystem. The controller component also transfers responses to the view accordingly.

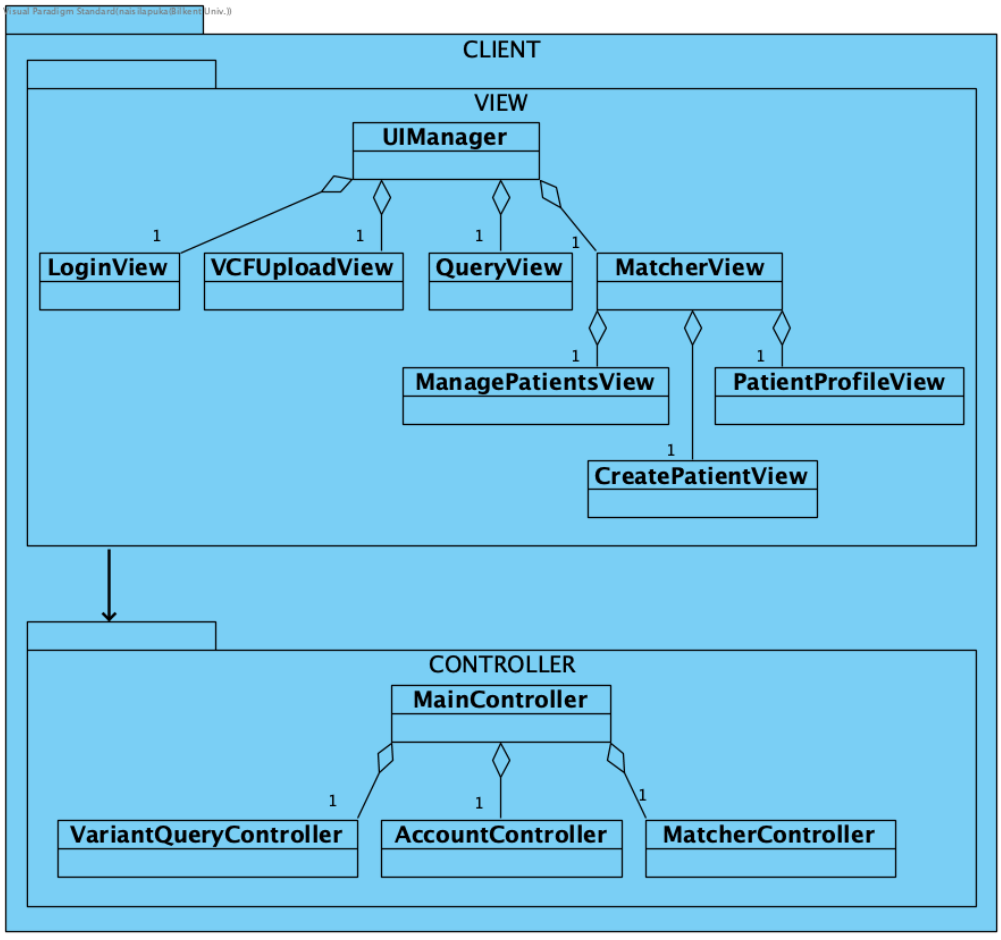


Figure 2: Hierarchy of Classes inside the Client Subsystem

4.2.1 View

The view component contains all the views of the application and their event firing mechanisms.

- **UIManager:** This class acts as a global manager for all the UI layers of LIBRA.
- **LoginView:** This class is responsible for the UI of Login screen. It communicates with AccountManager to make login requests.
- **VCFUploadView:** This class is responsible for the UI of VCF upload and annotation screen. It communicates with VariantQueryController to make upload requests.
- **QueryView:** This class is responsible for the UI of genetic variant querying and analyzing screen. It also communicates with VariantQueryController to make requests related to querying.
- **MatcherView:** In this view the user will be able to configure a matching algorithm. Communication is done with the MatcherController in order to transfer the request to the server.
- **ManagePatientsView:** This view is responsible for the UI of the management of a patient. Data for a patient is read and modified through various UI components in this class, and the requests are transferred by communication with the MatcherController.
- **CreatePatientView:** This view is responsible for the UI of the creation of a new patient. It communicates with the MatcherController as well to make new patient requests.
- **PatientProfileView:** In this view the user can view the information of a patient in the system. This view communicates with the MatcherController as well in order to make patient view requests. In this procedure, data is only read and not modified. This eases the communication with the server.

4.2.2 Controller

The logic of the client is responsible for triggering the execution of operations like VCF uploading and annotation, variant querying and analyzing and matching patients algorithms. These are achieved through communication with the server, depending on the events the user created in the views of the program. Here we give a brief description of the controllers. Server subsystem explanation includes more detail on how the controllers mediate the logic of the program.

- **MainController:** This class acts as a global manager for all the controller components of LIBRA.

- **VariantQueryController:** This controller handles VCF uploading and annotation, based on user's request in the views of LIBRA.
- **AccountController:** This controller handles account validation requests in order to log in, as well as new sign-ups to the system.
- **MatcherController:** This controller is responsible for executing operations of matching patients algorithms based on the events triggered through the views.

Note: MainController is not a real part of the core modules of the system. It is part of the diagram only for explanatory purposes. Each view of the system will directly communicate with its corresponding controller and will not use the MainController as a transfer median. However, considering that we will use the React JS library for building the user-interfaces, each view of the system will redirect its request to the main UIManager, as it is really convenient to handle multiple components inside a big component in React.

4.3 Server

All the main operations will be executed in the server and then communicated to the client. Regarding VCF upload and annotation, the request arrives at the server and automatic annotation is performed. The annotated variants are added to the user's database(s). The client only receives a response on whether the operation was finished successfully or not. For querying and analyzing variants, the user triggers the event in the client views by forming the queries and choosing various analyses to be conducted. These requests are communicated to the server, where querying and analysis actually occurs. The final output is then sent back to the client. Patient matching operations follow a similar fashion. The Logic component of the server subsystem is the main module of the program. It handles all the core functionalities. As per usual, the data component is where persistent data is kept. This data can be added, read and modified by the logic component.

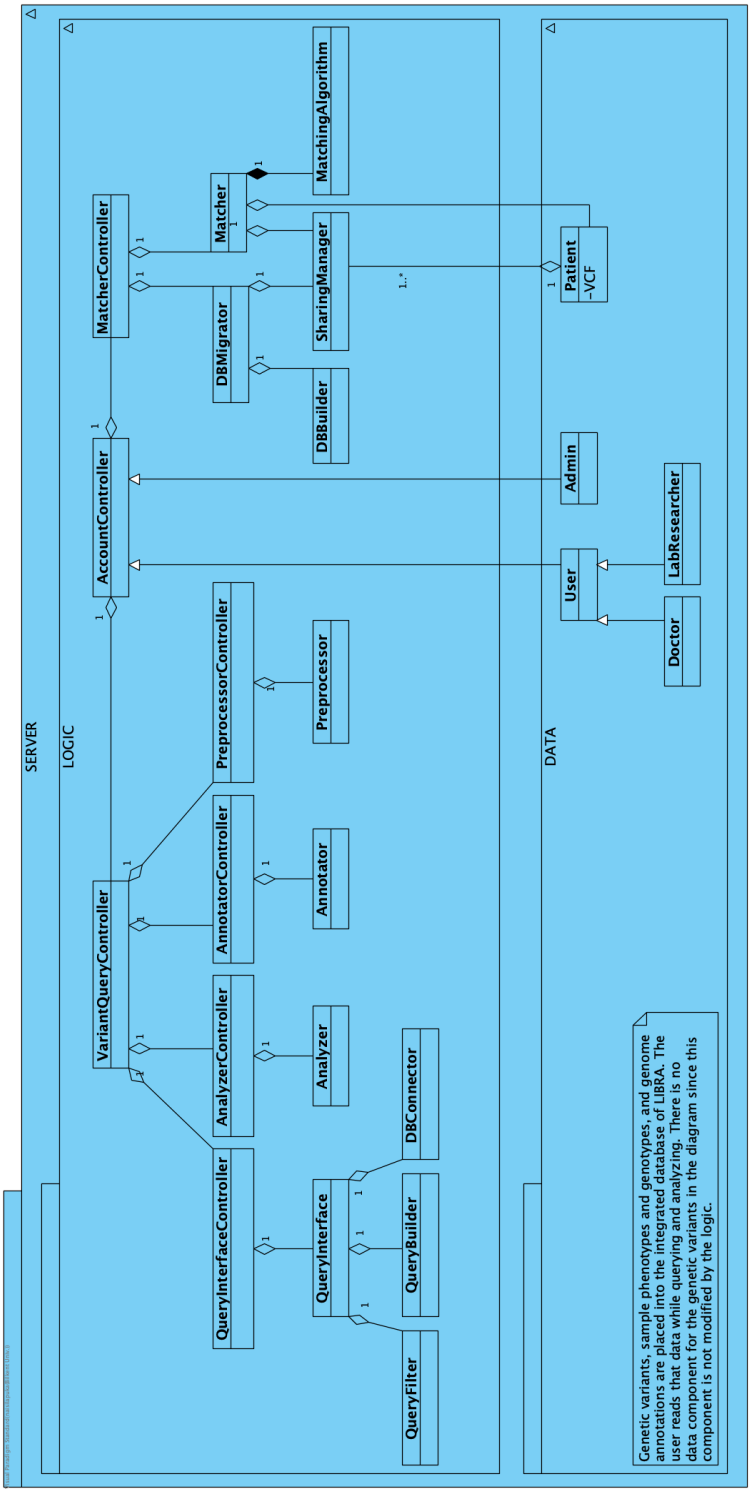


Figure 3: Hierarchy of Classes inside the Server Subsystem

4.3.1 Logic

This component is responsible for handling all the core functionalities of the system. **Genetic Variation Query Interface**

- **VariantQueryController:** This class acts as a global manager for all the controller layers of LIBRA's genetic variation querying interface.
- **Preprocessor/Controller:** This class is responsible for parsing of the VCF files before annotation.
- **Annotator/Controller:** This class is responsible for automatic annotation of uploaded VCF files.
- **Analyzer/Controller:** This class is responsible for serving as an API for several analysis method such as mendelian error and de novo mutations.
- **QueryInterface/Controller:** This class is responsible for building the queries, sending queries to the databases and fetching results.
- **QueryBuilder:** This class creates a database query based on the user input.
- **QueryFilter:** This class sets the filters for a specific query.
- **DBConnector:** Auxilary class for sending queries to databases and fetching results.

Matchmaking System

- **MatcherController:** This class acts as a global manager for all the controller layers of LIBRA's patient matching.
- **Matcher:** This class contains instances of many objects related to patient matching and orchestrate them.
- **MatchingAlgorithm:** This class is used for designating a customized matching algorithm.
- **SharingManager:** This class is used for managing the shared information related to patients.
- **DBMigrator:** This class is used for enrolling new databases to the matching process.
- **DBBuilder:** This class is used for setting the database up and running on LIBRA servers.

4.3.2 Data

This component is responsible for keeping the persistent data related to the program. It is merged with the Model in the MVC pattern.

- **User:** This class is the parent user class responsible for the general user tasks such as storing user information.
- **Admin:** This class has the ability to manage user accounts such as suspension or deletion.
- **Doctor:** This class is a user account that has the ability to set diagnosis.
- **Lab Researcher:** This class is a user account similar to Doctor but does not have the ability to set diagnosis.
- **Patient:** Class for storing patient information.

Note: Genetic variants, sample phenotypes and genotypes, and genome annotations are placed into the integrated database of LIBRA. The user reads that data while querying and analyzing. There is no data component for the genetic variants in the diagram since it is not modified by the logic.

5 Development/Implementation Details

5.1 Client/Server Implementation

The client and the front-end UI is written in React JS. We made this choice because of it's efficiency of rendering many components without refreshing the entire DOM. The server back-end is written in Python Flask. We built a REST API based on Flask and the REST API makes use of several annotation frameworks that are already installed in the remote server such as SnpEff. The server also containts the necessary databases for doing the annotation.

5.2 Similarity Algorithms

For the Similarity, modified cosine similarity is used. There are ontology's which are represented with directed acyclic graphs. For given two set of term, firstly sets are modified. The modification is adding parent of each element in the set into the modified set until it converges. After that, each element of graph represent a binary number in a vector if it present in the modified set it is 1 otherwise 0. For two set, cosine similarity calculated by these two vectors. In general, there

is matrix for all patient, when it multiplied with a patient vector it gives all similarities for that patient which we sort like an array.

5.3 Annotation Frameworks

The main framework we use is SnpEff. SnpEff is an open source tool that annotates variants and predicts their effects on genes by using an interval forest approach. SnpEff is used for annotating DBSNP and 1000 Genomes annotations. SnpEff comes with a different falvour called SnpSift which is used for obtaining the PolyPhen pathogenicity information. We run the SnpEff tool in multi-threaded mode with 16 threads in order to speed-up the annotation process.

5.4 Reliable Storage

LIBRA stores the genetic information in a PostgreSQL database. There are three main reasons behind this choice: reliability, performance and scalability.

PostgreSQL does everything possible to guarantee reliable and secure operation. This is crucial in storing really significant and delicate information like genetic build of variants. One aspect of reliable operation is that all data recorded by a committed transaction should be stored in a nonvolatile area that is safe from power loss, operating system failure, and hardware failure.

Moreover, query performance is really high in Postgres, and we make use of this both in genetic information upload and statistical querying. Also, LIBRA is planned to be extended to a distributed database, named Citus, which will substantially increase query performance.

LIBRA runs the matchmaking similarity algorithms (explained in the above subsection) in the background, which are constantly communicating with the database. Reliability and performance are very important here. In the following subsection, which talks about genetic filtering by queries, database choice is really important as well.

5.4.1 Filtering

The frontend of the filters use Material UI library to maintain a consistent appearance with the rest of the application. The UI is similar to filters of Varapp. There are four filters designed in total: Impact, scenario, frequency and pathogenicity. Filters on the UI are presented through collection of filters. Filters are modularized to enable addition or removal of filters to the Filter List. Each filter has its own state that keeps information related to the selected options for that filter. The state of the filter list holds the aggregation of states of all the other states. When the

user presses on the apply filter button on the filter list the aggregate state is forwarded to the backend for further processing and querying of relevant data from the annotated VCF file. The queries are optimized, but still simple to read as we work on a relational database. As mentioned in the previous subsection, database choice was crucial in the system. An example filtering query is as in Figure 4.

```
SELECT
  chrom,
  pos,
  variant_id,
  ref,
  alt,
  qual,
  annotation_type,
  annotation_impact,
  genename,
  feature_type
FROM
  vcf
  JOIN samples ON vcf.id = samples.vcf_id
WHERE
  variant_id IS NOT NULL
  AND annotation_impact = "HIGH"
  AND (
    annotation_type = "exon_loss"
    OR annotation_type = "frameshift"
  )
```

Figure 4: Example Variant Filtering Query

6 Testing Details

6.1 VCF Upload, Annotation and Querying

To test VCF Upload, first a database instance is created. This tests Postgres setup as well. Two small sample VCF files are uploaded to the system in both options: single patient and batch. The sample VCF files contain all the necessary information to trigger every part of the upload functionality: they mock a large 8 GB normal VCF file.

After uploading the sample VCF, annotation is done on it by a SNPEff command. To test the

correctness of the annotation, each field of the expected annotation output is accessed in order to store the information of the annotated VCF samples in the Postgres database.

As we can understand, testing in the system is done sequentially, arriving at the final step of querying. The filtering queries are tested with SQL queries as in Figure 4, comparing with an expected output. All testing uses *pytest* in Flask.

6.2 Matchmaker

To test the Matchmaker system, we separated the test into two major part as Patient Management, matching patient by HPO and Gene Ontology similarity algorithms. Patient Management tests were mainly done by hand to see the changes in the frontend on necessary database tables as well. This part was crucial since the information of the patient and the relationship between the patient and user are determining the outcome of the matchmaking process. The HPO and Gene Ontology similiarity tests were done by research mostly. We investigated the algorithms on the ontology similarity subject to get the intended results from the similarity algorithm we built. Then, we run this algorithm with different sizes of patient batches in our system to make sure it is scalable to use in our project.

7 Maintenance Plan and Details

The system would require two maintenance endpoints. One of them is related to "genome builds". Basically, genome builds are different reference that are used during the annotation process. Currently, we are using GRCh38. The genome build can be updated regularly in order to support more recent VCF files. The second maintenance plan would be to update annotation databases regularly such as DBSNP, PolyPhen, 1000 Genomes, etc.

8 Other Project Elements

8.1 Consideration of Various Factors

In this section, various factors that affected the analysis and design of the project will be discussed.

8.1.1 Public Health

The system is designed in a way that it handles any unexpected hardware/network failure while annotating genetic variants. No patient is diagnosed with the wrong disease since that may impose a health risk.

8.1.2 Public Safety

The system makes sure that no wrong/incomplete results are displayed to the doctors, since that may impose a safety risk to the patients of those doctors.

8.1.3 Global Factors

The nationalities of patients should be considered because it is important for interpretation of genetic mutations. Also, it would be beneficial to make patient queries between different nations for increasing the chance of matching. Yet, it is hard to design such a system because of the strict regulations between different nations.

8.1.4 Social Factors

The privacy of the patients should be protected. The system ensures that the sensitive data of the patients cannot be shared without the patients' consent. In the case where the patient data is shared, the system ensures that the data cannot be traced back to the patient. The system is designed in a way that protects patients against discrimination in their professional or social life.

Factors	Effect Level	Effect
Public Health	10	The system handles any unexpected hardware/network failure while annotating genetic variants. No patient is diagnosed with the wrong disease.
Public Safety	9	The system makes sure that no wrong/incomplete results are displayed to the doctors.
Public Welfare	0	
Global Factors	7	The nationalities of patients should be considered because it is important for interpretation of genetic mutations. Also, it is beneficial to make patient queries between different nations for increasing the chance of matching.
Cultural Factors	0	
Social Factors	8	The privacy of the patients is protected. The system ensures that the sensitive data of the patients cannot be shared without the patients' consent. In the case where the patient data is shared, the system ensures that the data cannot be traced back to the patient.

Table 1: Summary of Consideration of Various Factors

8.2 Ethics and Professional Responsibilities

There are not many commercial products like this project. Similar products exist to meet research demands. The existing products are either provided as additional services or are sold at high prices. Our project aims to provide a ubiquitous and uniform service to hospitals around Turkey to allow diagnosis, research and treatment of diseases, especially rare ones. Privacy of the patients is important. Certain genetic variations and diseases can lead to the discrimination against the patients in their social and professional lives. Therefore, it is imperative that the privacy of the patients is protected. Data will not be utilized other than the intended purposes of LIBRA. The doctors will filter what they want to share regarding their patients. The information shared between doctors in the platform will be restricted to ensure only the permitted data will be displayed.

8.3 Judgements and Impacts to Various Contexts

The various factors discussed in Section 8.1 were considered while designing and implementing LIBRA. Informed judgments that we made in our project consider the impact of out solution in global, economic, environmental, and societal contexts. The impact was observed both on the users and the hospitals. We summarized these in the following table:

Judgement Description:	The privacy of the patients comes before giving the information about patients to the other users. We only share patient id's and diagnosis between users.	
	Impact Level	Impact Descripton
Impact in Global Context	7	Because of the privacy restrictions, matching patients won't be informative to act on it without getting permissions from the other user's patients. This had both positive and negative impact on the users, since not sharing information has its advantages and disadvantages.
Impact in Environmental Context	0	This has no impact in the environment.
Impact in Economic Context	10	The impact is quite positive on the users since LIBRA is a free software that eases the job of doctors on analyzing genetic variants.
Impact in Societal Context	10	Privacy of the patients in the health industry should be protected since tools such that saves patients lives shouldn't concern the patient with their privacy also. The impact is positive, since our security and privacy restrictions create faith in the users of the system.

Table 2: Summary of Judgements and Impacts

8.4 Teamwork and Peer Contribution

We all contributed in delivering the reports and the user manual. We were divided into two teams for the implementation:

- VCF Upload, Annotation and Filtering: Abdullah, Berke, Naisila
- Matchmaking: Halil, Sami

8.4.1 Abdullah Talayhan

I was responsible for implementing the core features and maintaining the general structure of the codebase. These include the following:

- Writing the user authentication and registration system along with the project structure related with users. (both back-end and front-end).
- Writing the detailed project information page where the upload, notes, and VCF table components are displayed (both back-end and front-end).
- Designing the e-mail template for the similarity report that are sent to users via the matchmaking system.
- Preparing the landing page along with other UI components.

8.4.2 Berke Egeli

I worked on the initial design of matchmaker component back in January. I implemented the initial schema for the VCF file in the backend. I decomposed the VCF table into VCF and samples table to remove redundancy in the database. I implemented the frontend of the filter system which included scenario, frequency, impact and pathogenicity filters, the filter panel and filter list for the presentation of the filters in the UI.

8.4.3 Naisila Puka

I was responsible for the main functionalities of the back-end in genetic information uploading and filtering. I worked in collaboration with the front-end implementation of Abdullah and Berke to complete the following:

- VCF parsing using PyVCF and upload to the database after annotation.
- Database schema for relations between variants and patients (in accordance with the filters).
- VCF filtering in impact, DBSnp frequency, 1KGenome frequency, scenario and pathogenicity.

8.4.4 Halil Şahiner

I worked on the implementation of the Matchmaker future of our system with Sami. I built the frontend and backend of the patient management such as editing, creating and connecting patients to the matchmaker algorithm and users. For the backend of Matchmaker algorithms, I linked them to the necessary patients and users to send notifications by email for the matched patients.

8.4.5 Mahmud Sami Aydın

I have done ontology and similarity related parts on matchmaker. I firstly worked on querying human phenotype ontology terms in order to create search bar. Also made HPO traverser which gives details for HPO terms. After that, I worked on similarity algorithm for ontologies. I also redesign matchmaker with Halil. I linked genes and gene ontology terms.

8.5 Project Plan Observed and Objectives Met

As explained in the teamwork section, for the project we split into two groups. Abdullah, Naisila and Berke worked on the projects page and its related features while the Halil and Sami worked on the patients page, match making system and its related features.

In the projects page we first focused on the creation of separate projects. Then, we worked on the upload of VCF files to these projects. After successfully implementing the upload functionality, we connected the annotation tool to the backend. We optimized the loading time of the VCF table by only uploading a small portion of the VCF table initially. Later, we increased upload options of the VCF files: Users can only upload a VCF file, they can batch upload multiple VCF files, users can associate the VCF files with existing patients or new patients can be created per sample in the VCF file. After enhanced upload options were completed we implemented filters for scenario, impact and frequency of variants in the VCF files.

The patients page features were developed in parallel with projects page. We first implemented patient creation. Then we implemented the matchmaker functionality through HPO metrics. After matchmaker, we implemented the auto-creation of patients and editing patients features. Finally, we implemented the mailing feature where users will be notified through an email if there is a match after matchmaking process.

Name	Duration	Start	Finish
Genetic Filtering and Analysis Back-End	46 days	11/11/2019	01/13/2020
Genetic Filtering and Analysis Front End	28 days	12/11/2019	01/17/2020
Query Builder	42 days	12/25/2019	02/20/2020
Matchmaker Front End	44 days	03/03/2020	05/01/2020
Database Sharing	48 days	02/10/2020	04/15/2020
Creation and Management of Profiles	20 days	02/17/2020	03/13/2020
Matching Patients	41 days	03/06/2020	05/01/2020

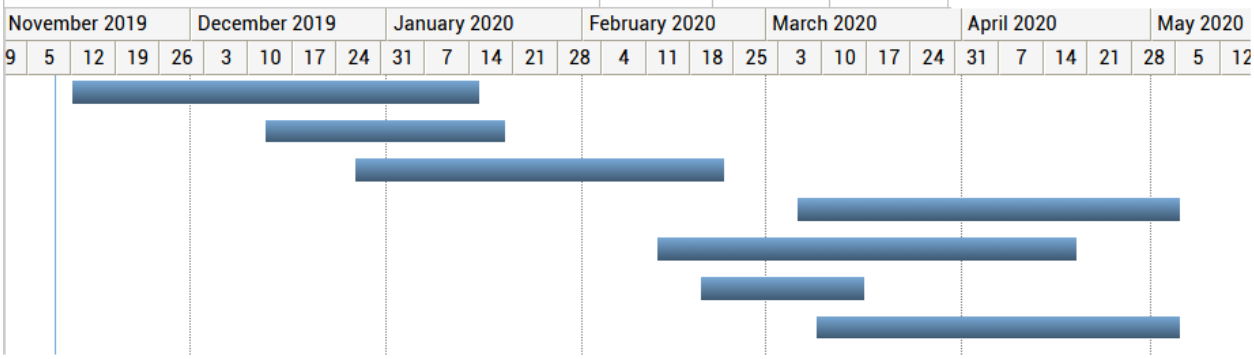


Figure 5: Gantt Chart (Date format: M/D/Y)

Above, you can see the initial project plan we devised. The only deviance we had from this project plan was related to the implementation phase of the projects page which included the genetic filtering and analysis front-end/back-end, and the query builder. We pushed the project page implementation to the beginning of the March. The time frame we allocated for the tasks in reality and in the Gantt chart were roughly equal.

We had weekly meetings with our supervisor. Some of our objectives changed as a result of our weekly meetings. We abandoned the database sharing feature. We also abandoned the idea of having system and hospital administrators, and instead gave more functionality to the users. We successfully implemented the VCF upload and annotation features. After meetings with our supervisor and bioinformaticians from the industry we decided to alter our query builder feature. Instead we implemented user friendly filters that automatically queried the database for the appropriate tuples of the VCF file. We successfully implemented features for the patient creation and matchmaking.

8.6 New Knowledge Acquired and Learning Strategies Used

Most of the group was inexperienced with web development and bioinformatics tools. So, we had to get acquainted with React and Redux libraries, flask and django frameworks, and tools like SnpEff for VCF annotation. To learn React and Redux, we postponed implementation and spent one week to learn about the basics of libraries through online tutorials. We used a similar approach to learn about the frameworks as well. We took one week off from the implementation of the project to learn the basics of the frameworks through online tutorials. We read the documentation of SnpEff to integrate it into our back-end.

9 Conclusion and Future Work

9.1 Conclusion

In conclusion, the system we want to build reached to the end product with some design changes such as changing backend framework or adding Gene Ontology similarity algorithm for matchmaking which are not in the initial design of the system. However, we implemented and integrated the features that we planned to create to Project Libra while keeping it in maintainable and open to further development state.

9.2 Future Work

The filtering system could be improved with more options to query the annotated VCF file . Currently we have filters for scenario, frequency and impact of variants. Later on, these could be extended to the pathogenicity of variants as well. Currently, the filtering system we have works through a set of options selected through a static interface. It is possible to extend this feature by adding a text editor which would allow more advanced users to enter custom queries to filter out the VCF file results.

In the future the underlying architecture of the system could be changed to make the system more scalable and make better use of cloud resources. The architecture could be changed from a simple 2-tier architecture into microservices architecture where different components of the application like authentication, projects and patient management can be containerized and orchestrated to decrease computation load like annotation or similarity calculation on individual servers.

10 Glossary

- **Variant Call Format(VCF):** The Variant Call Format specifies the format of a text file used in bioinformatics for storing gene sequence variations.
- **Genetic Annotation:** Genetic Annotation is the process of identifying the locations of genes so that it serves as an explanation about the functionality of genes.
- **Genotype:** The genetic constitution of an individual organism.
- **Phenotype:** The set of observable characteristics of an individual resulting from the interaction of its genotype with the environment.

References

[1] J. E. Stajich and H. Lapp, "Open source tools and toolkits for bioinformatics: significance, and where are we?" *Briefings in Bioinformatics*, vol. 7, no. 3, pp. 287–296, Sep. 2006. [Online]. Available: https://doi.org/10.1093/bib/bbl026

[2] "Crying without tears unlocks the mystery of a new genetic disease - scope," Mar. 2014. [Online]. Available: https://scopeblog.stanford.edu/2014/03/20/crying-without-tears/

User Manual for LIBRA

1) Build and Run

This section explains the procedure of building and deploying LIBRA from the github source files.

There will be many code snippets that explain the building process. Each snippet has the following format:

- First the github repository has to be obtained.

```
$ git clone https://github.com/projectlibra/projectlibra.git
```

- Once you clone the repository, you will encounter a file structure like the following:

```
/projectlibra
├── README.md
├── libra-backend
└── libra-frontend
```

Building the Client

The client is written in React JS and the build procedure is fairly simple.

Install React JS

If React JS is not currently installed on your system, we consult you to the official React JS documentation for the installation process.

Install dependencies

You can install all dependencies required for the client using the following snippet:

```
$ cd libra-frontend
$ npm install
```

Setting up the HOST IP

You must change the host IP embedded in the file **host.js** for the backend server that you are running:

```
var host = 'Backend Server IP:PORT'
```

Running the client

You can run the client via npm and the landing page will be displayed in the default browser:

```
$ npm start
```

Building the Server

Since the server has many dependencies in Python, we suggest using a virtual environment:

```
$ pip install virtualenv
$ virtualenv libra
$ source libra/bin/activate
# For deactivating virtualenv
$ deactivate
```

Installing dependencies

```
$ cd libra-backend
$ pip install -r requirements.txt
```

Setting up a PostgreSQL Server

We consult you to PostgreSQL server setup tutorial from digitalocean.

After the setup, the configuration line under **app.py** has to be changed with the following information:

```
postgres://user_name:password@db_server_ip:port/db_name
```

Setting up SnpEff

We consult you to SnpEff installation manual

In addition to the installation procedure, we need GRCh38.86 database.

```
$ java -jar snpEff.jar download -v GRCh38.86
```

Running the Server

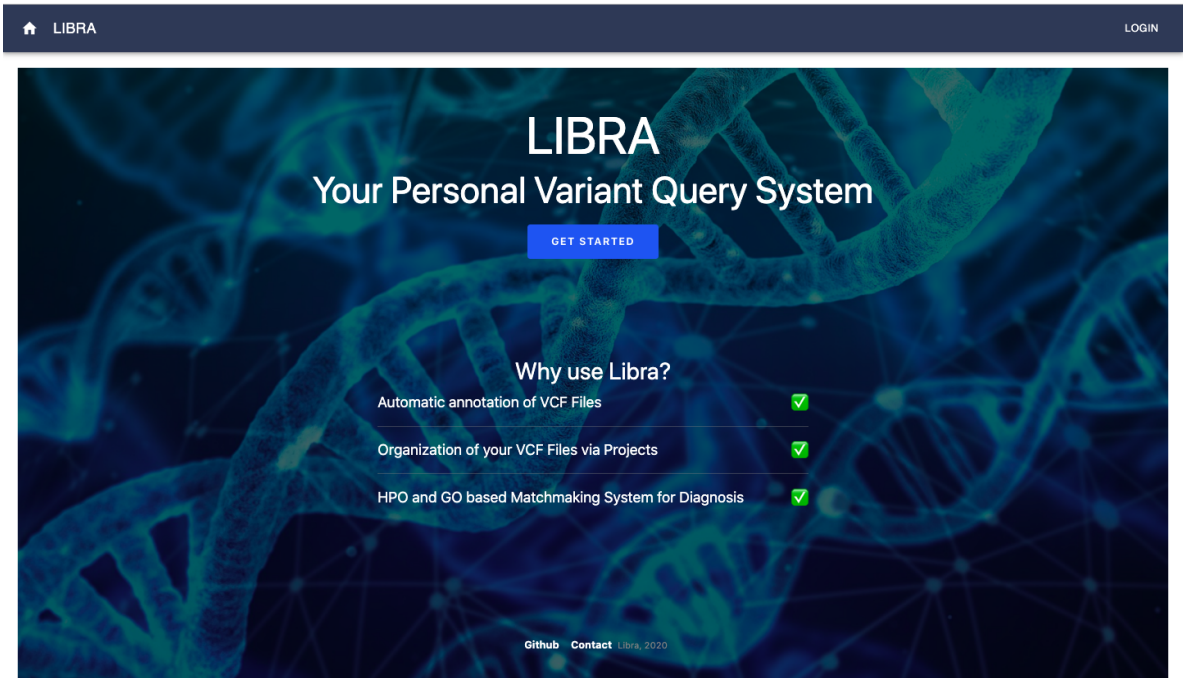
The following will run the back-end server with the specified **PORT** on the server's IP address accepting all connections.

```
$ cd libra-backend
$ flask run --host 0.0.0.0 --port PORT
```

2) Usage of LIBRA: Registration and Profile

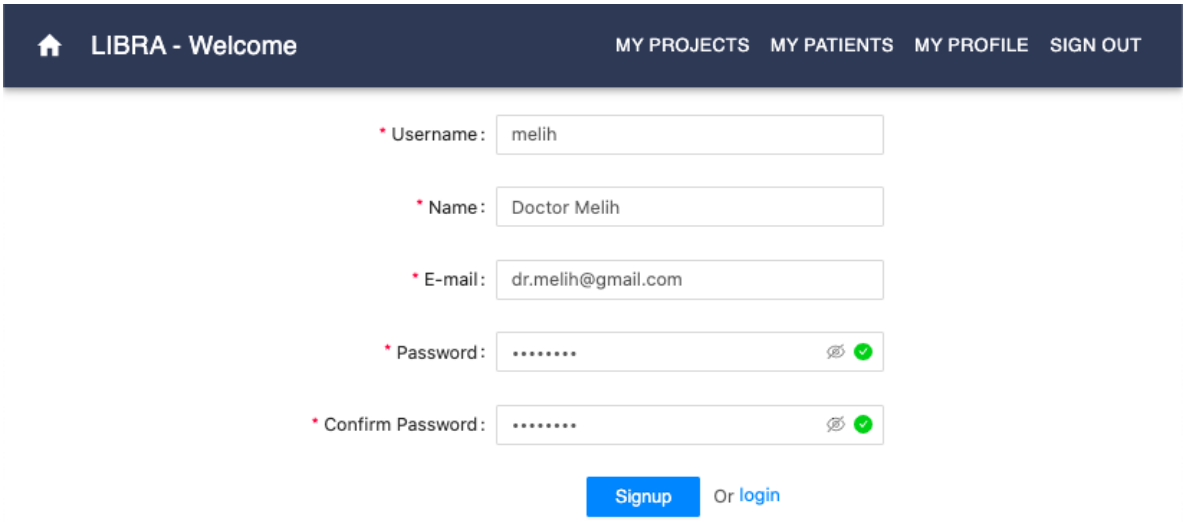
Homepage

This is the landing page potential users will be greeted with when they click on our website on a web browser. Visitors can proceed to sign up by pressing the Get Started button in the center.



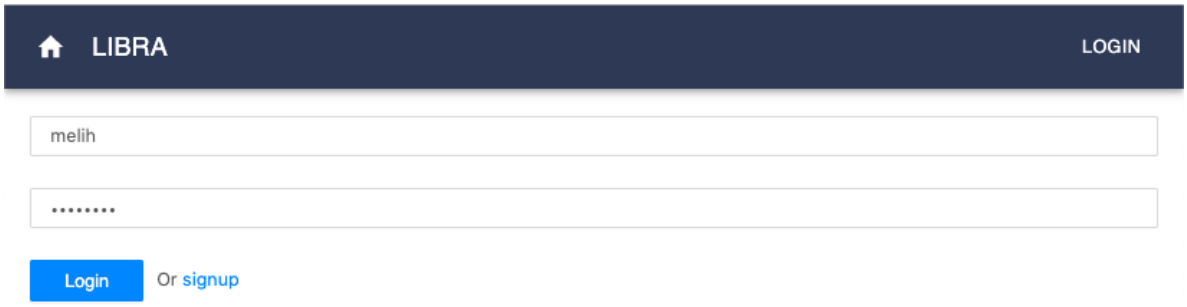
Sign Up

In this page, to create a new users visitors will have to fill in the required input fields. These fields are username, name, email address, password and password confirmation. After clicking sign up, users will be redirected to the index page to login their accounts.



Login

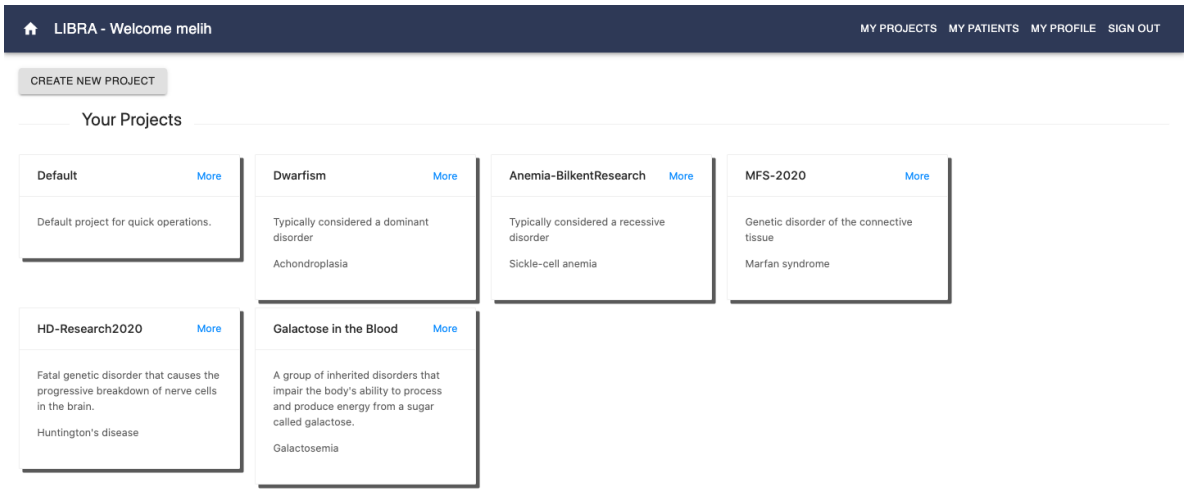
In this page visitors will be presented with two options: If the visitor has an account they can proceed to log in by entering their username and password or if they do not have an account they can choose to create one by clicking on the sign up button.



3) Usage of LIBRA: VCF Upload, Annotation and Filtering

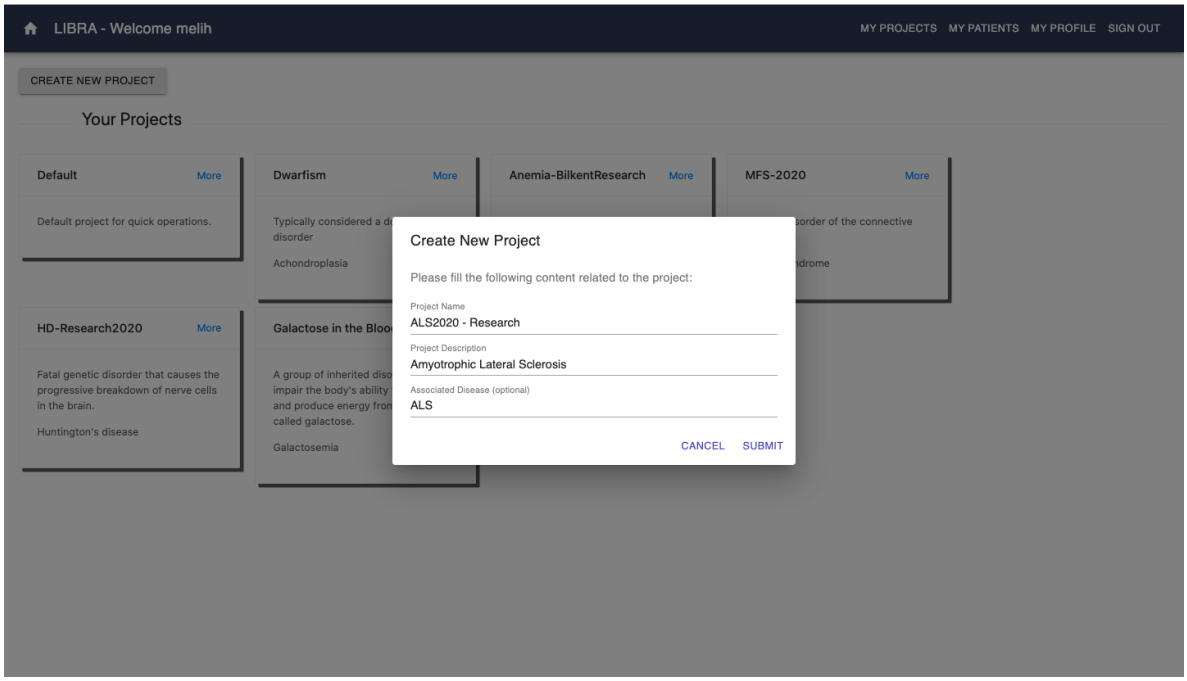
My Projects

After logging in, users will be directed to the projects page. In this page they will have one project by default for quick start on a random project. Additionally, users can create their custom projects.



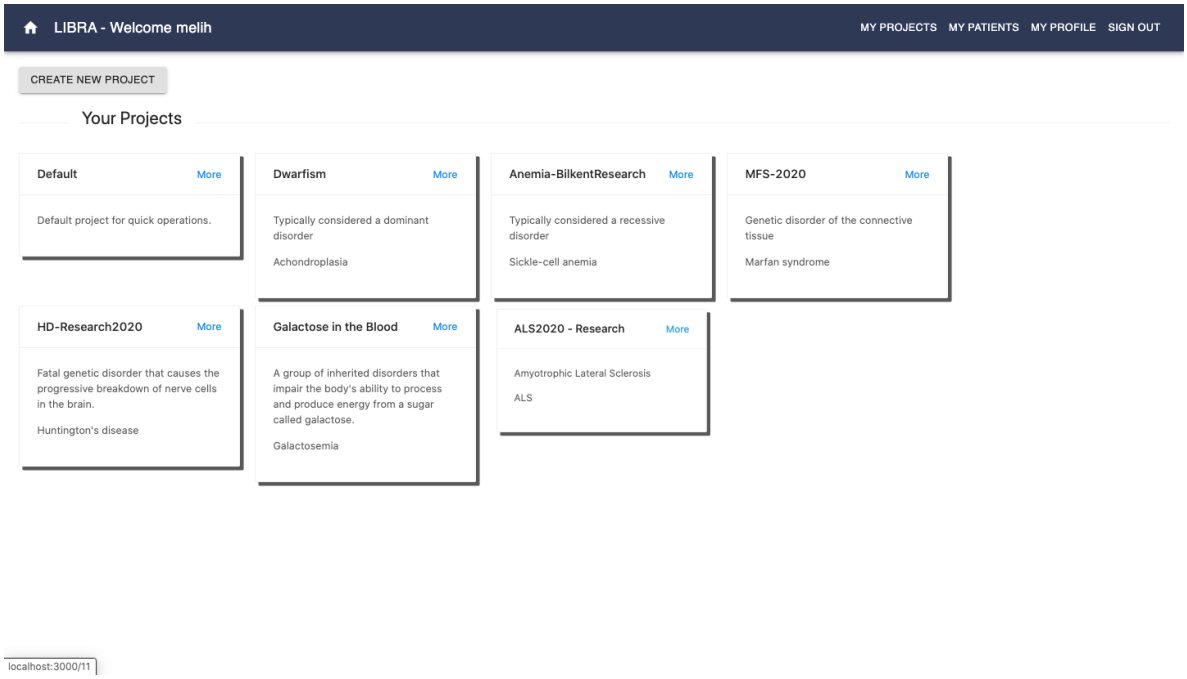
Create a New Project

When the users press the "CREATE NEW PROJECT" button, a dialog will pop up which will ask the user to fill in the Project Name, Project Description and optionally the disease associated with the project.



Project Details

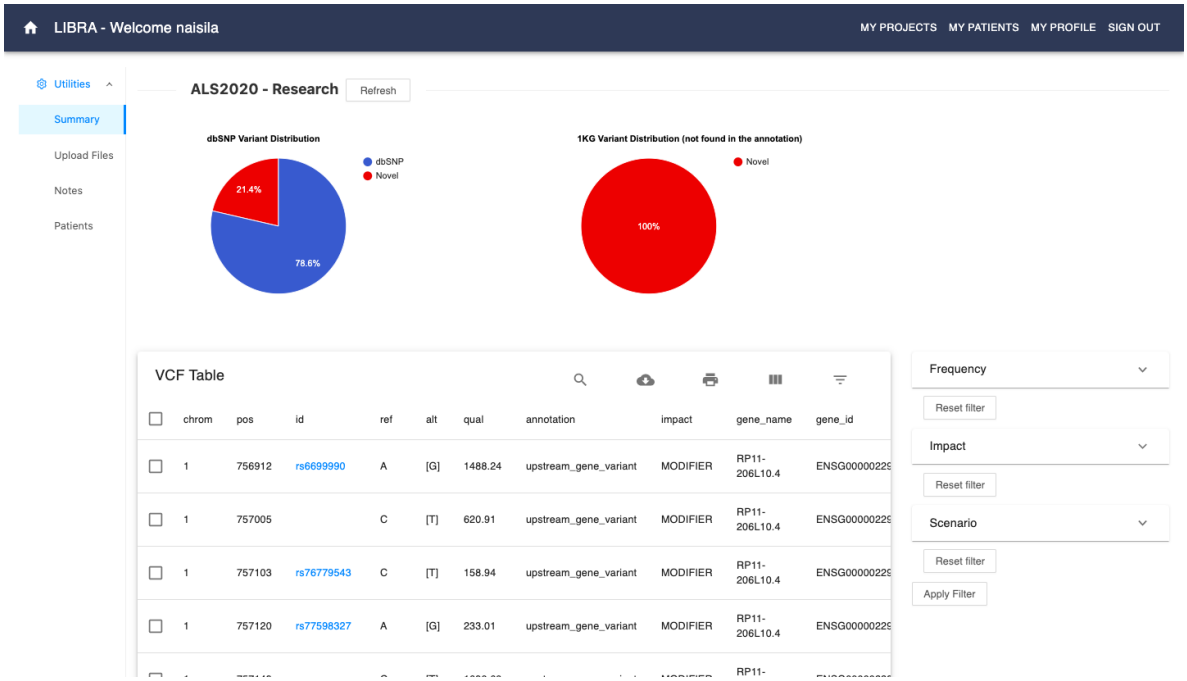
By clicking "More" at the top right corner of the project box, the user can redirect to project details page. On the right side of this page, there is a menu that contains summary, upload, notes and patients items.



localhost:3000/11

Project summary

User is redirected to this page after clicking on a specific project on the "Projects Page." User will first be presented the VCF table. If there are no VCF files uploaded to the project instead of seeing the VCF table the user will see "You haven't uploaded any files yet." If there is a VCF file uploaded to the project, however, they will see two pie charts indicating the presence of variants in the 1000 Genomes and dbsnp databases, the annotated VCF file presented in a table and a filter to select certain tuples of the VCF table based on frequency, scenario and impact of the variants, as shown in the following image:



Because of the immense size of VCF files, initially only 1000 elements of the table will be uploaded to the UI. The user can select to add another 1000 tuples to UI table by pressing on the "Load More" button. They can repeat this operation as many times as they want until the entire file is uploaded to the UI.

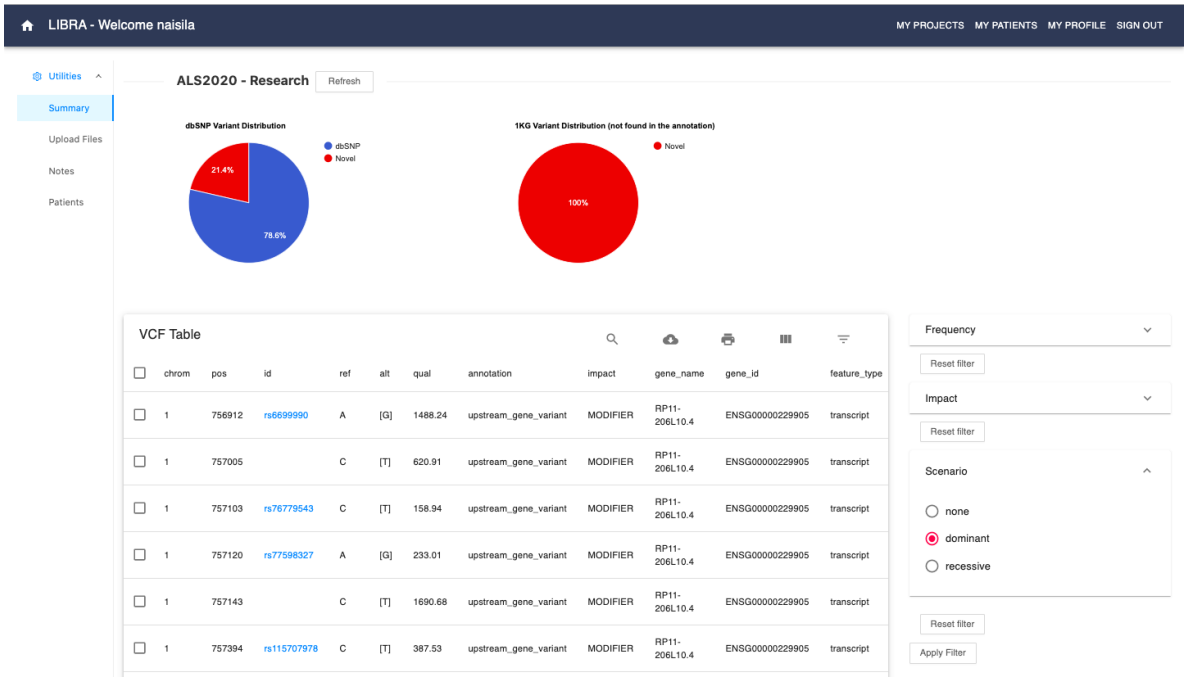
VCF Table

chrom	pos	id	ref	alt	qual	annotation	impact	gene_name	gene_id	feature_type
1	757143		C	[T]	1690.68	upstream_gene_variant	MODIFIER	RP11-206L10.4	ENSG00000229905	transcript
1	757394	rs115707978	C	[T]	387.53	upstream_gene_variant	MODIFIER	RP11-206L10.4	ENSG00000229905	transcript
1	757547		CCT	[C]	64.98	upstream_gene_variant	MODIFIER	RP11-206L10.4	ENSG00000229905	transcript
1	757640	rs3115853	G	[A]	38856.0	upstream_gene_variant	MODIFIER	RP11-206L10.4	ENSG00000229905	transcript
1	757734	rs4951929	C	[T]	44726.5	upstream_gene_variant	MODIFIER	RP11-206L10.4	ENSG00000229905	transcript
1	757936	rs4951862	C	[A]	37117.0	upstream_gene_variant	MODIFIER	RP11-206L10.4	ENSG00000229905	transcript

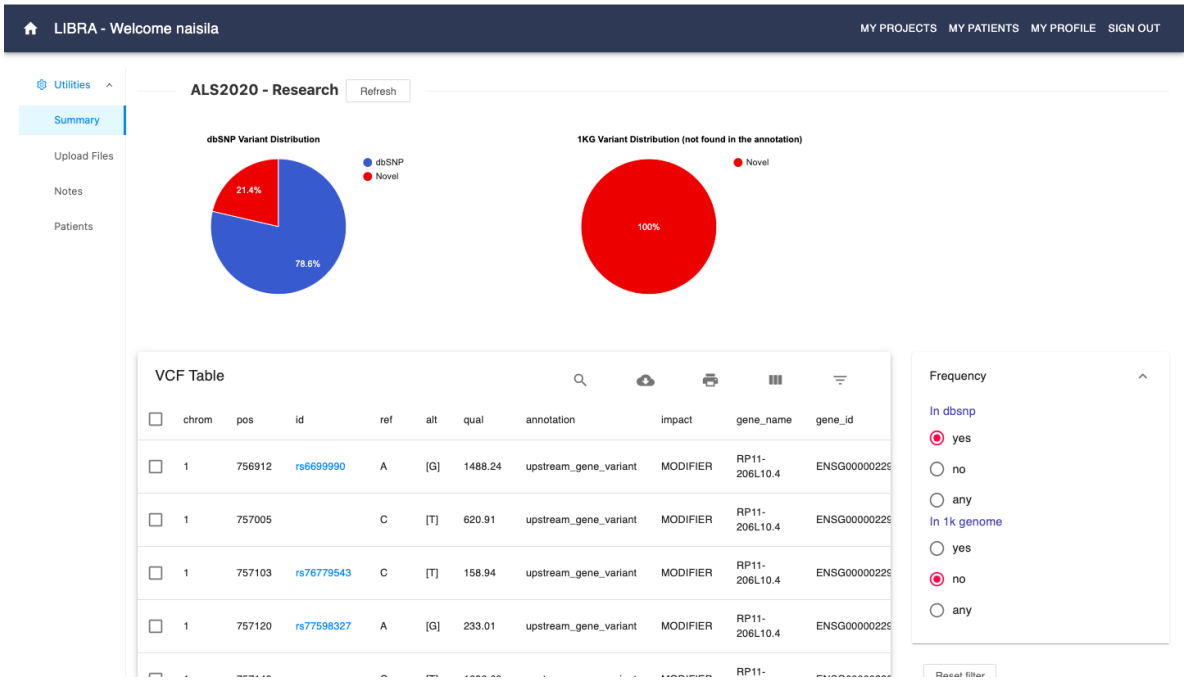
Displaying 1000/actual rows. Load More Rows per page: 10 1-10 of 1000

To the left of the VCF table there are three filters. These filters are scenario, frequency and impact filters. Users can select the options presented in the filters and when they are done selecting the options they can press on the "Apply Filters" button to view tuples on the VCF table that fit into the selected options.

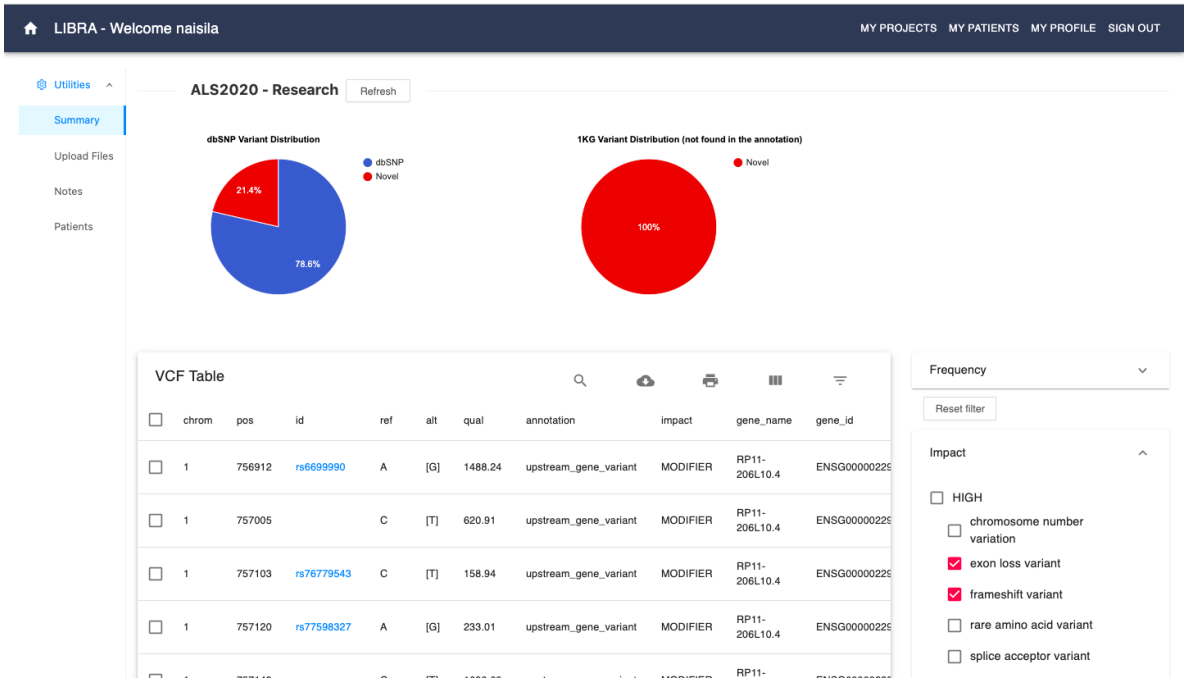
The scenario filters select dominant or recessive variants.



Frequency filter selects tuples in dbsnp or 1k Genome database.



Impact filters select tuples with high, moderate, low or modifier putative impacts.



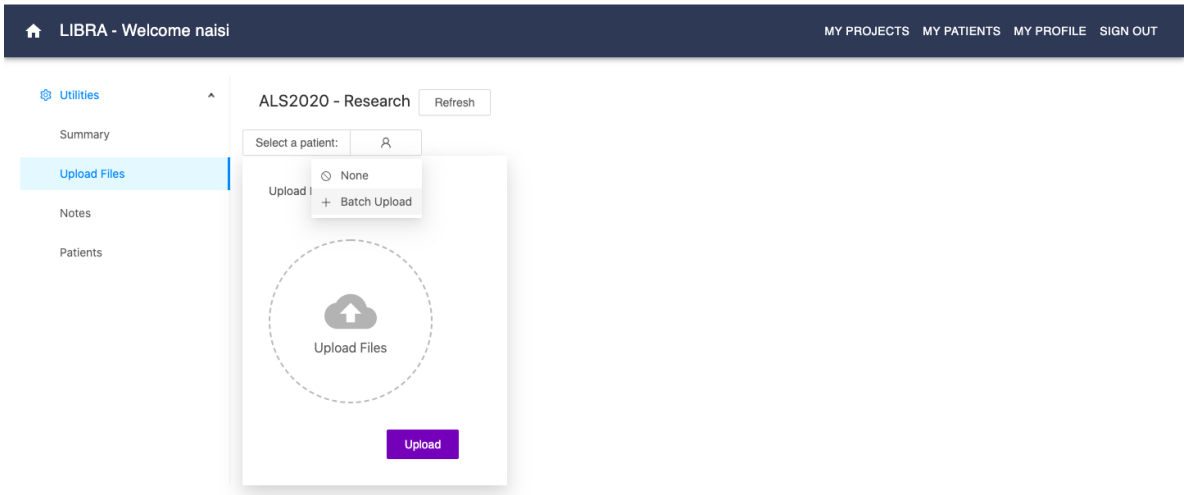
Scenario and frequency filters are radio button groups, so, the user will be able to only pick one of the options presented there for each filter. Impact filter is a checkbox group, so, the user can select a combination of the impacts.

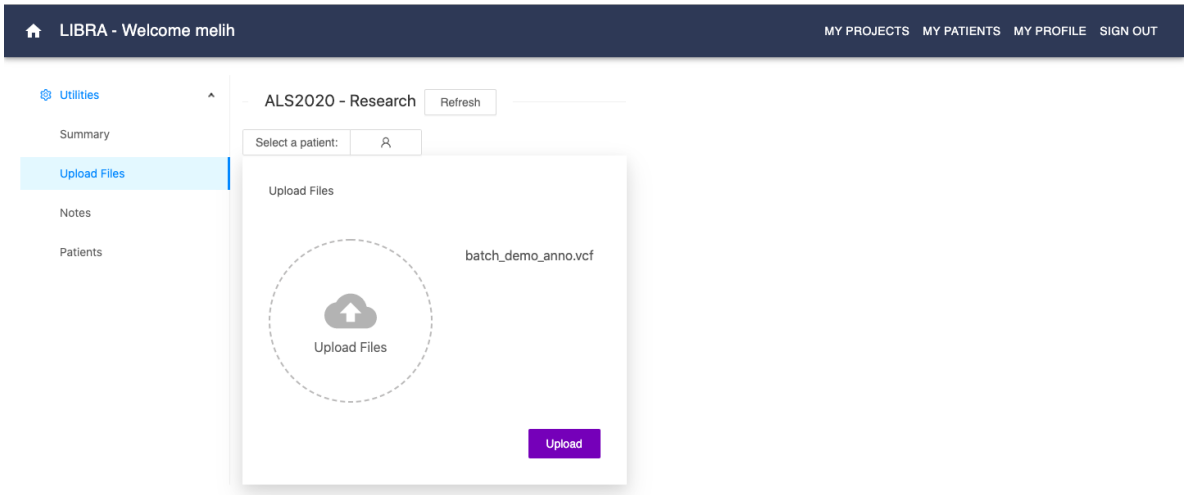
VCF upload

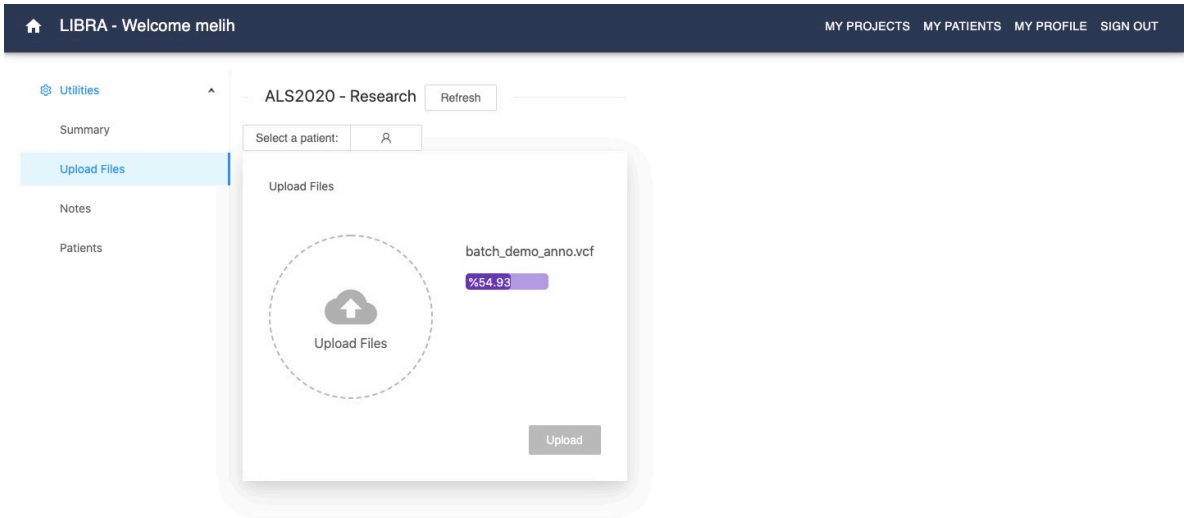
This page is where the user uploads the VCF files. On the top of the page there is a dropdown menu that has the options none, batch upload or specific patients. Below that is the dropzone for the file upload. The users can drag and drop one or more VCF files into the dropzone. Based on the option selected on the dropdown menu the upload procedure will work differently:

- If the user selected none option, then the uploaded VCF will not be related to any patient. It will be used for statistical purposes only.
- If the user selected the batch upload option, the uploaded VCF has two or more patients inside, and they all will have automatically created profiles in the system.
- If the user selected a specific patient, data would be imported for that patient from another project to the current one.

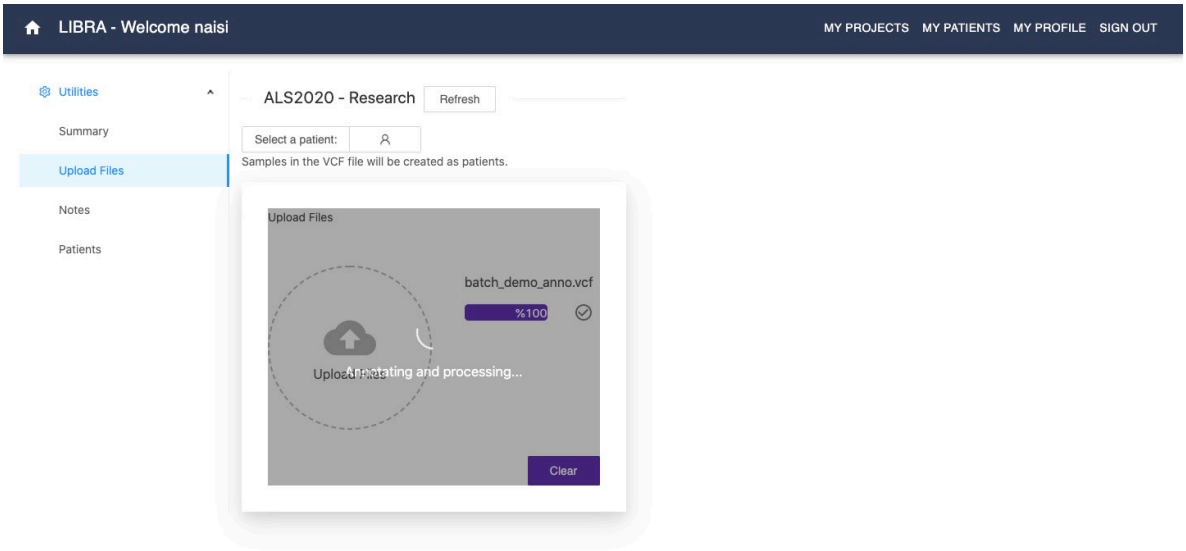
We show the procedure for batch upload:





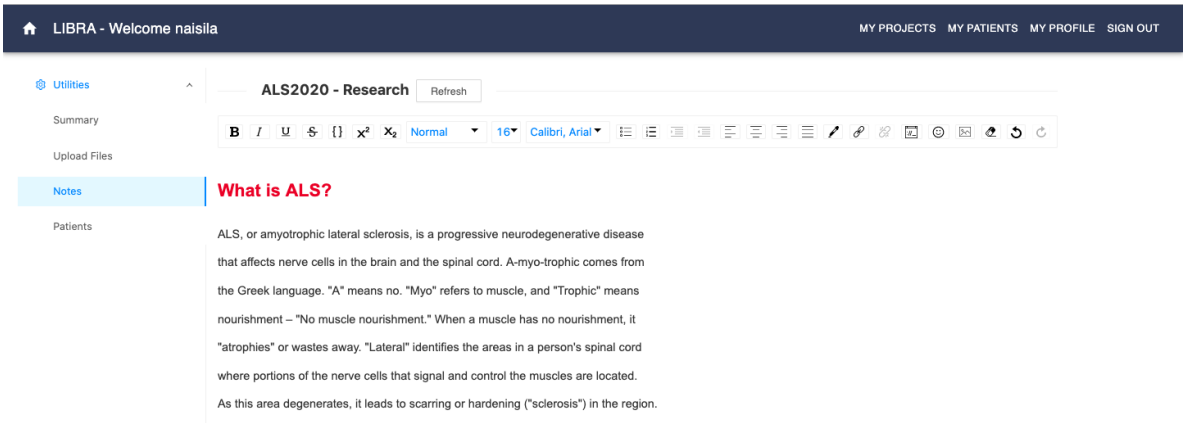


After the files are successfully uploaded the annotation process will take place in the backend and the users will be unable to interact with the dropzone.



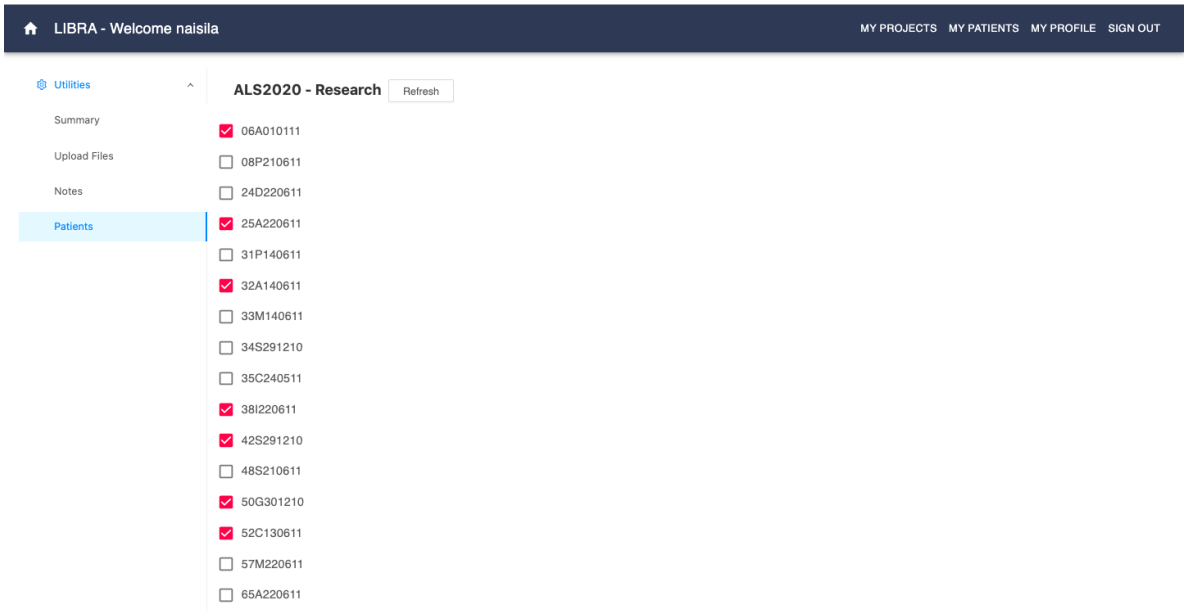
Project Notes

The user can add any notes related to the project in this tab. The notes are provided with cool formatting as well.



ProjectPatients

After uploading a batch vcf file, the patients will be automatically created, and the user can choose whether the patients have the associated disease of the project as well, by clicking on the checkboxes, as shown in the figure. In this example, the case is on ALS disease.

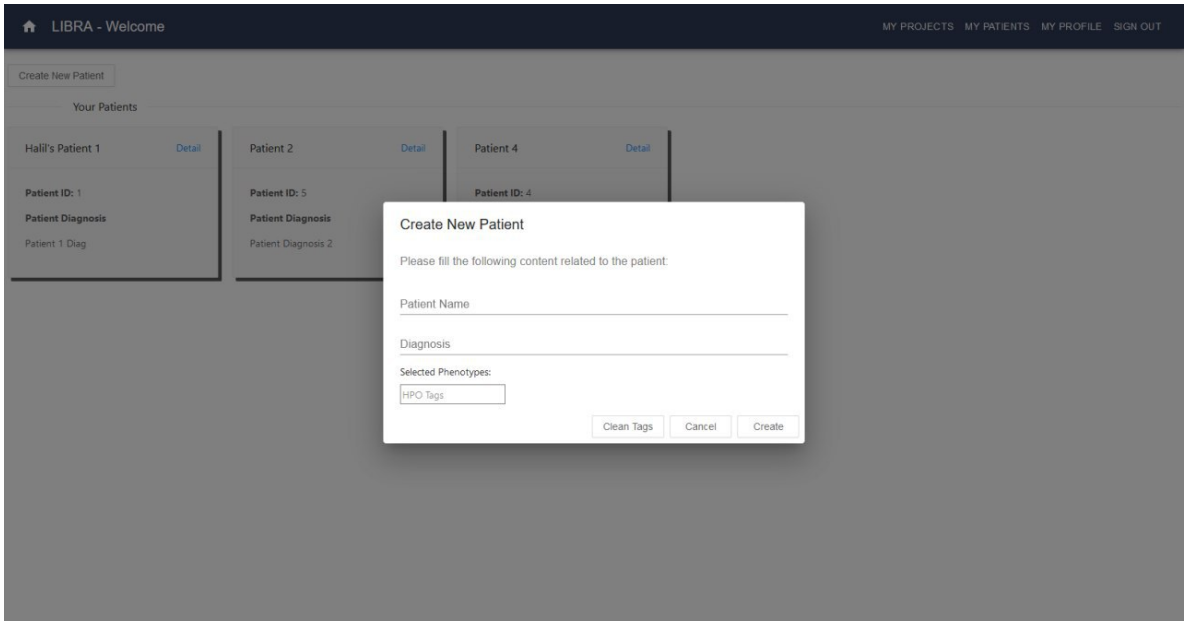


4) Usage of LIBRA: Match Maker

Match maker contains facilities which are: creating a patient with diagnosis and human phenotype ontology(HPO) terms, showing over all view for patients, showing detailed view for a patient, getting gene ontology similarity from VCF annotation, accessing HPO traverser, accessing automated similarity notification system, accessing manual similarity notification system, and changing similarity threshold configuration.

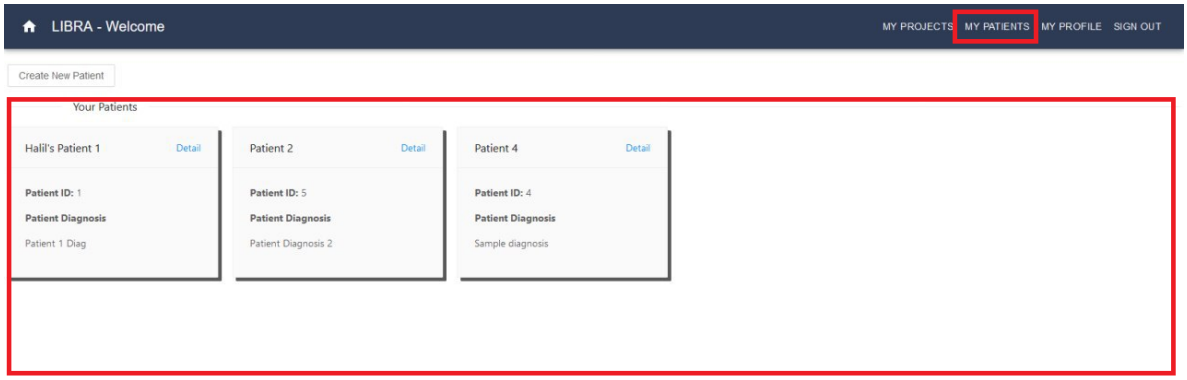
Create New Patient

To create a new patient click "My Project" button on the navigator. After that, press "Create New Patient" button. Fill "Patient Name" and "Diagnosis". Type HPO Tags via search bar and select from dropdown menu. You can delete tags by pressing "Clean Tags" button. After adding all phenotypes, click "Create" button.



See Overall View for Patients

Click "My Project" button on the navigator. Below "Your Patient", you see all patients. For each patient you see patient name, patient id and patient diagnosis.

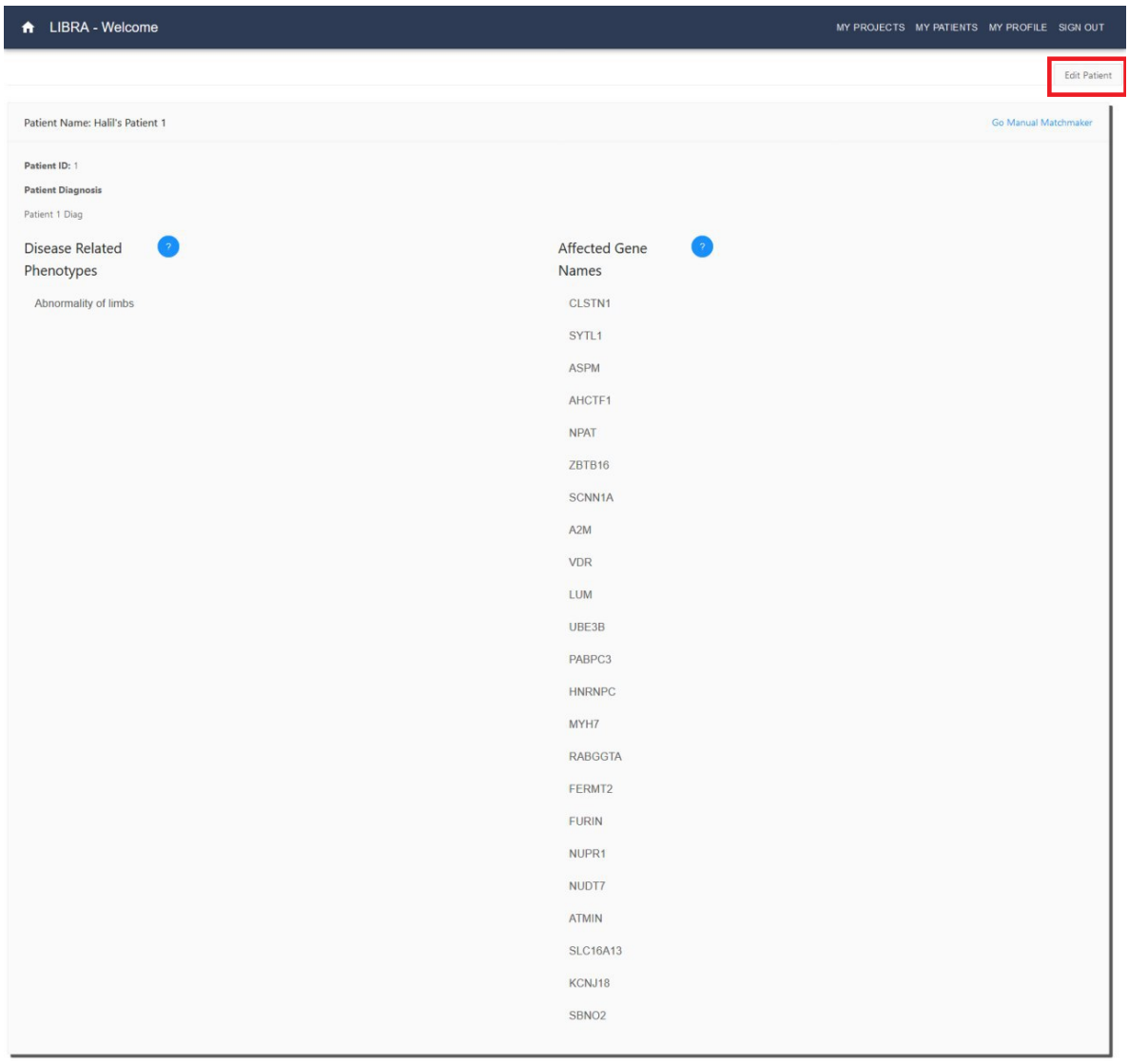


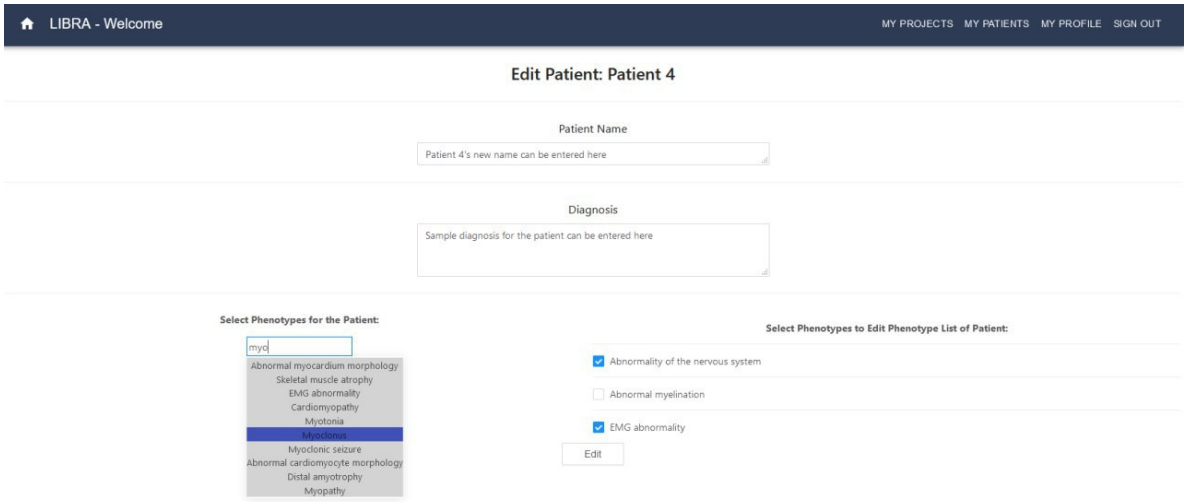
See Details for a Patient

On the overall view for patient, click detail button of correspondent patient. You will see the same information with preview. Also you will see disease related phenotypes and affected gene names. If you click a phenotype, you can see details of phenotype and traverse on phenotypes. If you click gene name, you are redirected to gene information.

Editing a Patient

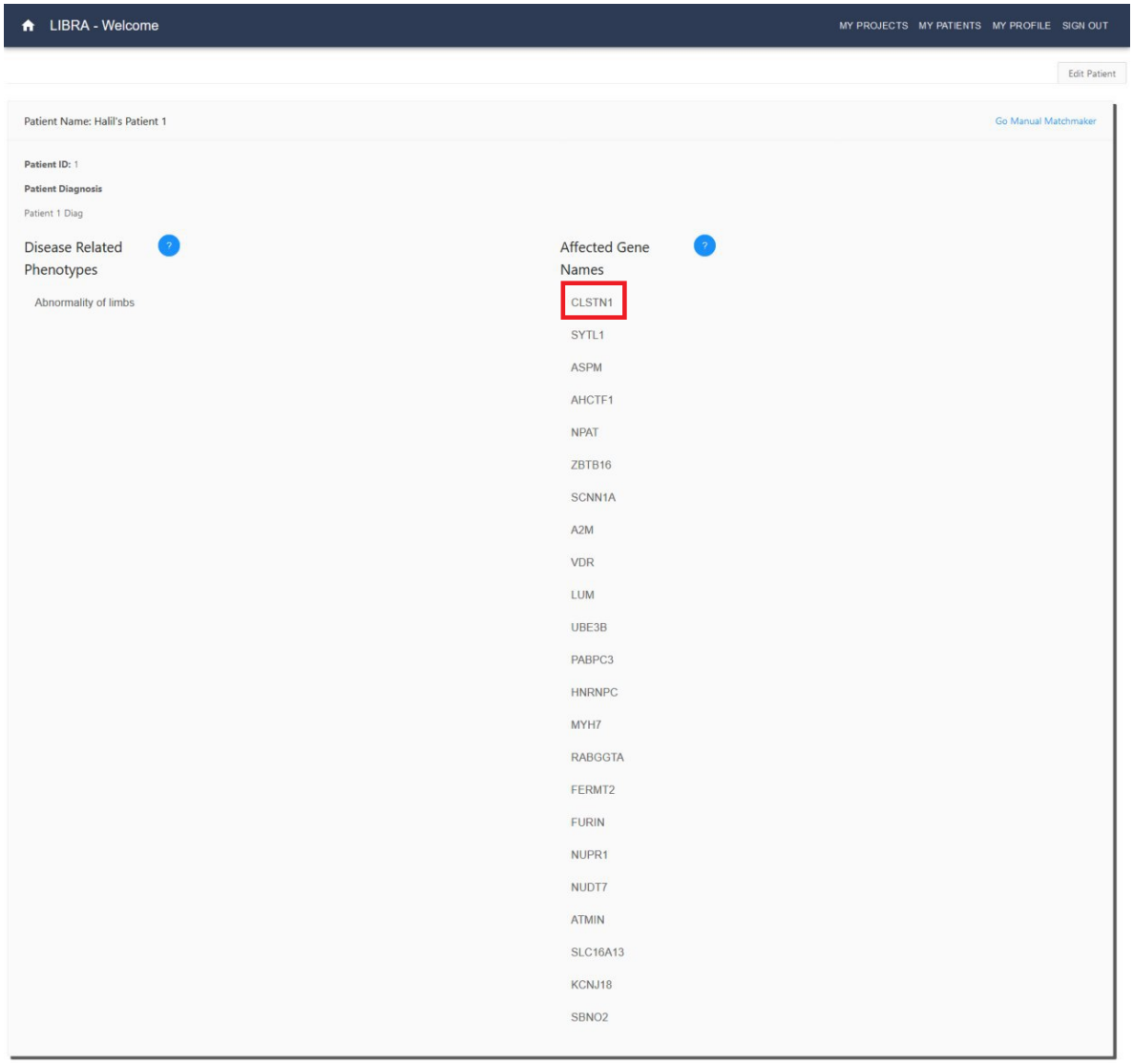
On the detailed Patient page, click "Edit Patient" button. You can give a new name to the patient. You can change diagnosis via text box. You can add new phenotype via search bar. Also you can remove phenotypes by unchecking the checkbox of corresponded phenotype.





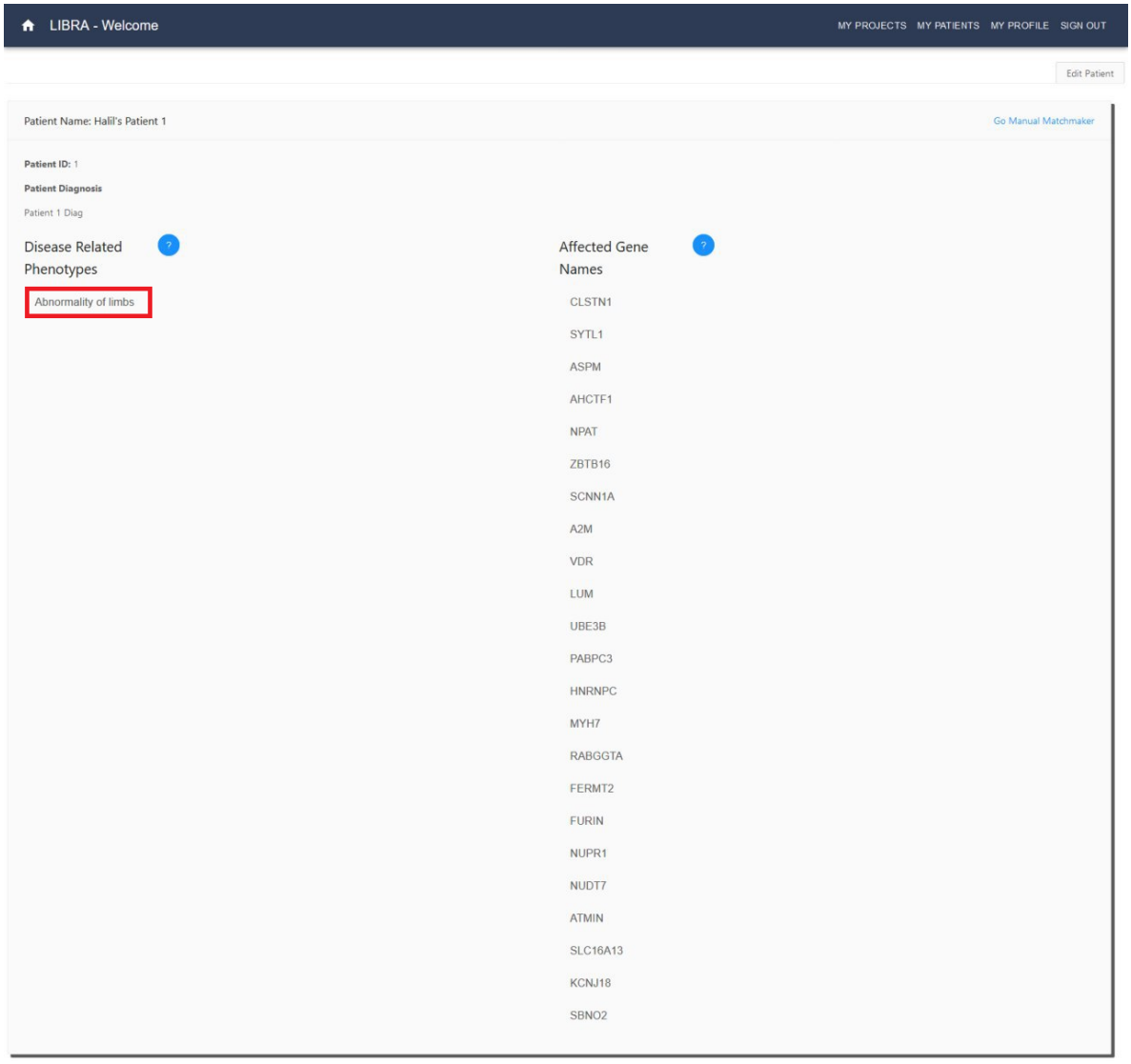
Get Highly Affected Genes For a Patient

After uploading the vcf file, click patient logo. Select name of the patient. Automatically highly affected genes will be added to the patient.



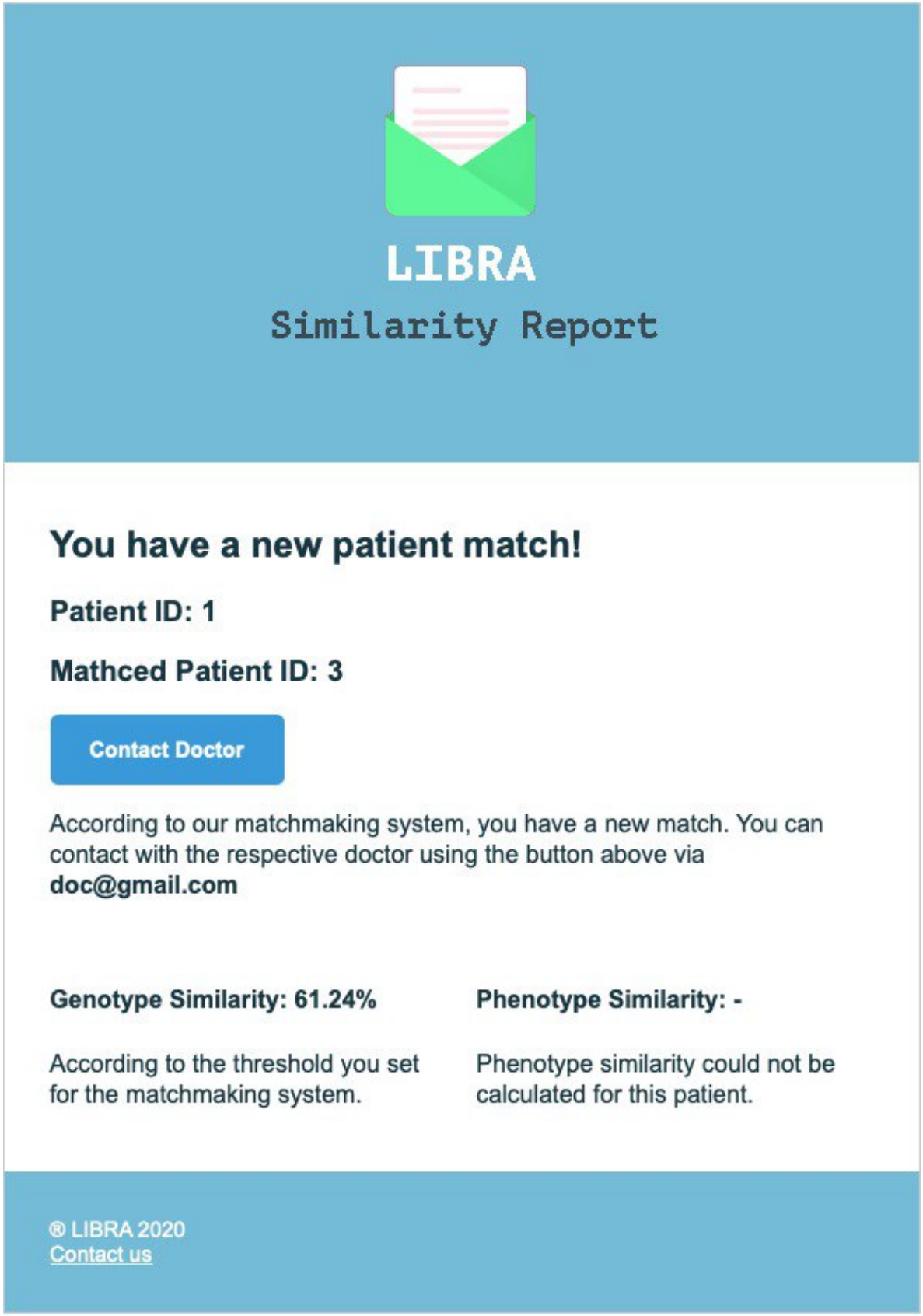
HPO traverser

When you click phenotype name on the detailed patient page, you will be directed to the HPO traverser. On the traverser, you will see name of the phenotype, definition of the phenotype and relative phenotypes. You can traverse parents and children by clicking name of that phenotype.



Automated Similarity Notification System

According to the threshold, if there is a similar patient to your patient, you will automatically get an e-mail which informs you about contact info of doctor who has the similar patient. Also e-mail indicates both the HPO similarity and GO similarity.



Manual Similarity System

On the detailed Patient page, click "Go Manual Matchmaker" link. There are three different systems to use. Firstly, you can choose some phenotypes and search patients with those phenotypes and click "Run Manual Matchmaker" button. It returns proper patients for the query. Secondly, you can list patients which are most similar to the patient according to phenotype by

clicking "Run HPO Matchmaker". Finally, you can list patients which are most similar to the patient according to affected genes by clicking "Run GO Matchmaker".

Similarity Threshold Configuration

Click "My Profile" button on the navigator. After that, enter value between 0 and 1 for both "Phenotype Similarity Threshold" and "Genotype Similarity Threshold". Click "Save Changes". New configuration affects Automated Similarity Notification System.