Bilkent University

Department of Computer Engineering

Senior Design Project

LIBRA: Genetic Filtering and Diagnosis Matching System

High Level Design Report

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

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

High Level Design Report
Dec 31, 2019

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.

1.1 Purpose of the System

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.

1.2 Design Goals

1.2.1 Scalability

The Gemini tool uses SQLite database. In order to increase the scalability of our system we plan to upgrade the infrastructure by switching to PostgreSQL database, later extended to the distributed version.

1.2.2 Backup and Recovery

The hospitals should be able to do their backup on their local database. Also the server needs to do regular backups for the accounts of doctors and their patients in the genetic matching platform.

1.2.3 Availability

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

1.2.4 Accessibility

The application should provide language support for Turkish language since the application's focus group is medical doctors in Turkey.

1.2.5 Reliability

The system will ensure that the comparison of the patients' information to find matches for diseases must give reliable results and handle unexpected failures. That is, if the system fails while doing comparison (server/client side errors), the error will be reported and the comparison procedure will handle the process appropriately, making sure only reliable matches/mismatches will be displayed.

1.2.6 Portability

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

1.2.7 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

1.3 Definitions, Acronyms, and Abbreviations

- **CRUD:** Short for Create Read Update Delete.
- **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.
- **Variant Call Format(VCF):** The Variant Call Format specifies the format of a text file used in bioinformatics for storing gene sequence variations.

1.4 Overview

LIBRA is a web-based application that focused on being used by medical institutions as a tool that safely collect and store patient information and their genomic data to compare with others and detect/diagnose rare diseases. The users will be able to create profiles for their patients which consist of personal information, diseases, genomic data and related pictures of patients. Those fields and patient profile altogether can be private for other users; however, users will be able to use the application for their patients.

The application has two main systems which are Query Builder and Matchmaker. Both of the systems provide a web-interface which is supported by back-end servers to process and store data. To use systems of LIBRA, users must belong to medical institution which is accepted by LIBRA. After acceptance of a institution, accounts for their respective personnel will be signed in LIBRA. By those accounts, users can log in to LIBRA from its web-interface. After that, users can navigate through Matchmaker and Query Builder system. In Matchmaker, users can create, delete or edit patient profiles. They can use matcher for their patient profiles after they decide which institutions they want to include to their search for a match. Furthermore, they will be notified by an email when there is a match for the patient profiles they shared with other institutions to be able to contact them. In Query Builder, users can create customized queries for their use by selecting pre-defined query options and run existing queries on uploaded VCF databases with their own VCF data. Also, user will be able to upload single or multiple VCF files to intended genomic databases that connected to LIBRA. While uploading VCF's or querying on databases for a VCF file, users can log out or close the application to avoid waiting time because of annotation process of multiple VCF files while uplodaing them or querying a large genomic database with a complicated query for their VCF.

The process of the above functionality will be ran on cloud servers and sent back to users from those servers since all data will be stored in cloud servers to avoid downloading large genomic data on devices of users. After the process done in cloud servers, results will be sent to client side of

LIBRA for users.

2 Current Software Architecture

Currently, there exists two systems that are like the major components of our project. The first one is the GEMINI framework and the second one is the Matchmaker Exchange project.

2.1 Gemini Framework

Most of the functionality of GEMINI is available through a command-line based tool. This causes issues in the installation and deployment of the system on the machines of users. There are initiatives to develop a web browser UI for the tool, however, currently, the web browser is also opened through the command-line. Also, the browser UI is running on the local machine of the users.

2.2 Matchmaker Exchange API

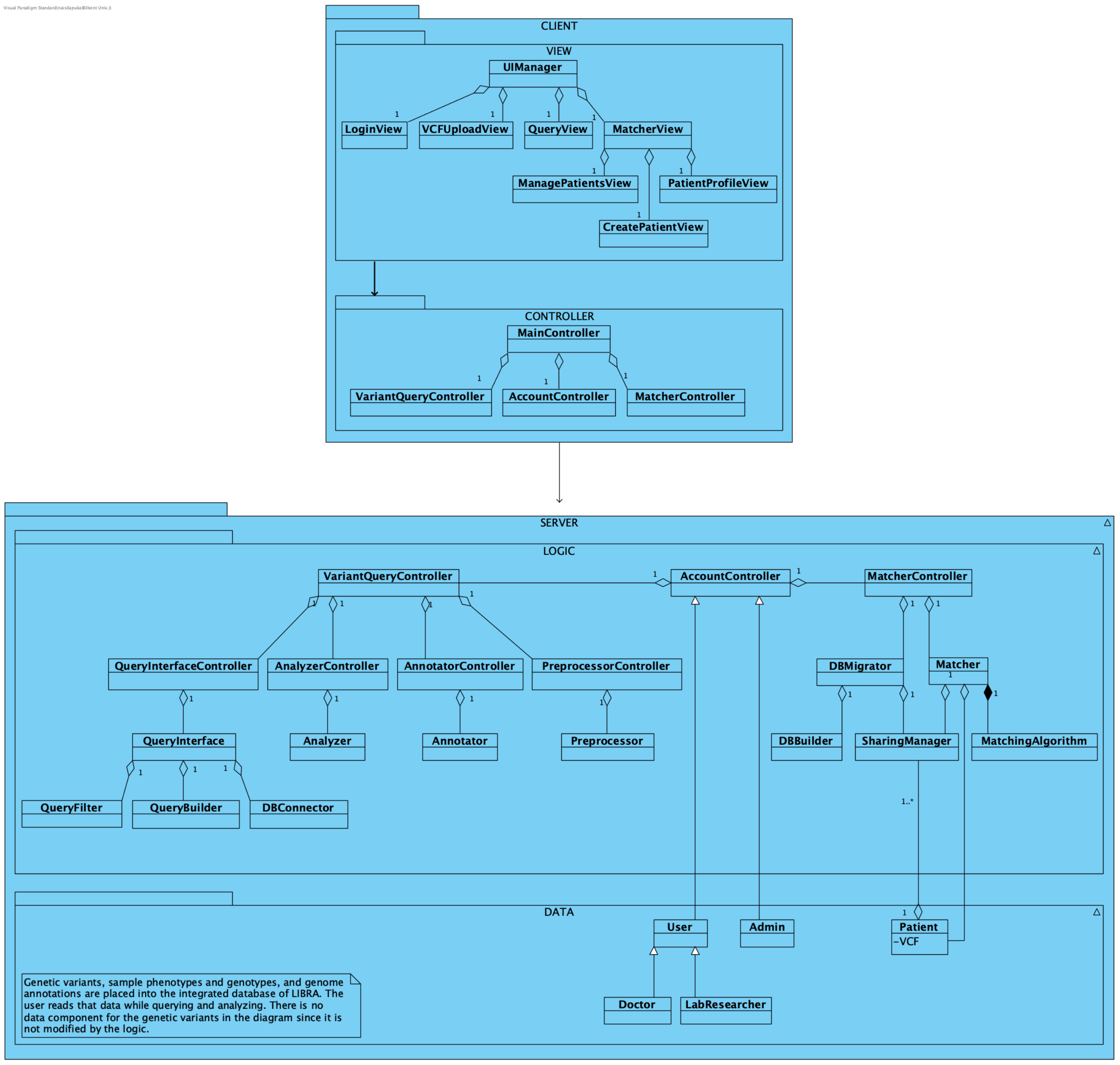
Matchmaker Exchange is a collaboration of multiple organizations. These organizations provide matchmaker services which can later be accessed through the API provided. Data sets can be deposited to any of these services to make matching available for these data sets. There are three types of users, namely, clinicians, researchers and patients. Depending on the service used access to functionalities of matchmaker might be restricted to patients. Also, depending on the service used, matching protocols, scoring protocols, how users are notified when internally and externally initiated queries result in a match differ.

3 Proposed Software Architecture

3.1 Overview

LIBRA is going to be a web application composed of two main modules: *Genetic Variation Query Interface* and *Patient Matching Platform*. The first module provides an interface for storing and annotating genomic data in order to query variants and explore the data, whereas the second one acts as a patient social network for doctors who seek similar genetic profiles related to a specific disease in order to understand and diagnose the disease further. These modules will be integrated into the same user interface. The potential users of this project are medical doctors in medium-sized hospitals, laboratories and research centers in Turkey.

3.2 Subsystem decomposition



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.

Figure 1: General Client-Server Architecture of Libra

The architecture of LIBRA will follow the client-server model, as shown in Figure 1. Execution of operations like VCF uploading and annotation, variant querying and analyzing and matching patients algorithms will be achieved through communication of client with the server. Each event will be triggered in the client, and executed in the server, and then the response will be sent back to the client. Each operation is explained in detail in Section 4.

3.3 Hardware/Software Mapping

The system will be composed of two main devices that act within the Client-Server Model.

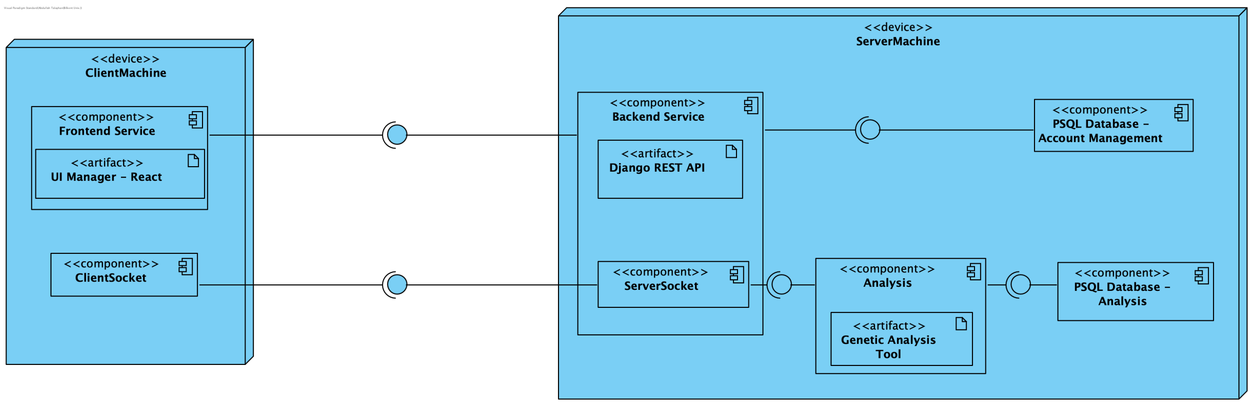


Figure 2: Deployment Diagram for Client and Server

The client machine runs a front-end service which will be implemented in React JS, this service will fetch data from the back-end service using Django REST API to be displayed to the client machine.

Although the Django REST API will handle the requests for many CRUD tasks, it will not be capable of fetching Genetic Analysis data during a lifecycle of an HTTP request. Hence, the client machine will be connected to another backend service via a Server Socket which is a dedicated service for handling genetic analysis and matchmkaing queries. This dedicated service will also have a seperate database in order to serve the same purpose.

3.4 Persistent data management

Persistent data contains uploaded VCF file/s to the intended database. During the upload of a VCF file to a database, VCF file needs to be annotated by our system and then it can be actually uploaded to the intended database. Therefore, the VCF files need to be kept on servers to annotate and to be uploaded to the intended database. Then VCF files can be erased from servers. Although we deal with VCF files through GEMINI, we use PostgreSQL in LIBRA instead of SQLite, which is used in the current GEMINI system.

The server also keeps information of users -medical doctors- and patient profiles that are connected to the users. Patient information and their connection to the doctors is kept on servers because of the patient matching system.

3.5 Access Control and Security

To create a user account, users must prove that they work in a hospital that LIBRA accepts (a list of hospitals will be prepared). After the hospital is accepted by LIBRA, they will provide personal and work information. A verification email or SMS will be sent to the hospital the user claims to be working at, based on the hospital's contact data found in LIBRA. By that verification, users can access the databases that are only accessible by that hospital since hospitals may have privately shared databases between each other. A relation between a user and hospital will provide this permission control.

To secure the database system and control the access control on them, we changed the GEMINI database system from SQLite to PostgreSQL since PostgreSQL has native support for using SSL connections to encrypt client/server communications for increased security.

3.6 Global Software Control

LIBRA has a event-driven software control system. When users send a request to our servers as a query on GEMINI, the processed result of the request will be sent to users. Also for the custom query, after giving the information about the query, there will be an event that saves that query on our server for later use. For the side of patient matching, the creation and editing of a patient profile needs to be requested to the server as an event, after users decide they are done with filling or changing the information about patient profile.

3.7 Boundary Conditions

3.7.1 Initialization

LIBRA is a platform independent web application; therefore, it is available to all prospective users through any web browser. In order to use the application, the users must be authenticated in the login page. Users that do not have an account have to create a user account in the sign-up page. Only doctors working at verified hospitals will be able to create an account. Verified hospitals will be added to the system by the administrative staff of LIBRA. Doctors must use their real names and credentials to sign up. Sign-up procedure will automatically fail. After signing up, doctors will be accepted or rejected, and they will be notified through email. Then, they will be able to access their account and LIBRA functionalities through the login page. Logged in users will be given an authentication token that will keep them logged in and prevent from authenticating again for a period. Users need internet connection for initialization.

3.7.2 Termination

The session will expire either when a user logs off or when their authentication token expires due to inactivity. If the user closes the tab but their authentication token has not expired yet, they will be able to log in without authenticating again.

3.7.3 Failure

Failures of the system can be related to either client side or server-side issues. VCF uploading, annotating or matchmaking functionalities can stop arbitrarily if the server or the database crashes or the internet connection goes out. In case of such failures none of the ongoing operations will be completed and the system will be rolled back to the last state that was saved before the crash.

4 Subsystem Services

In LIBRA, two main subsystems will interact with each other to query variants and match patients in an interactive manner: client and server. This architecture is merged with the classic Model-View-Controller/Logic software design pattern, as will be explained in the following subsections.

4.1 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.

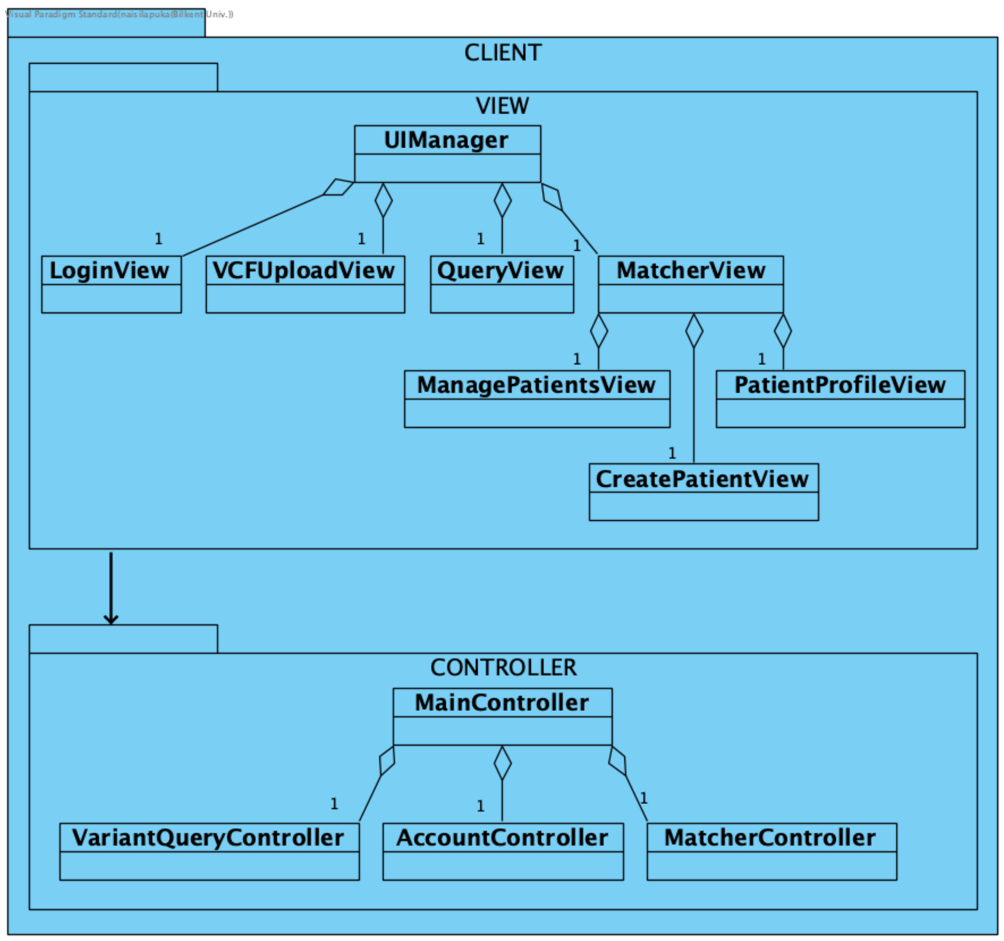


Figure 3: Hierarchy of Classes inside the Client Subsystem

4.1.1 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.

4.1.2 Controller

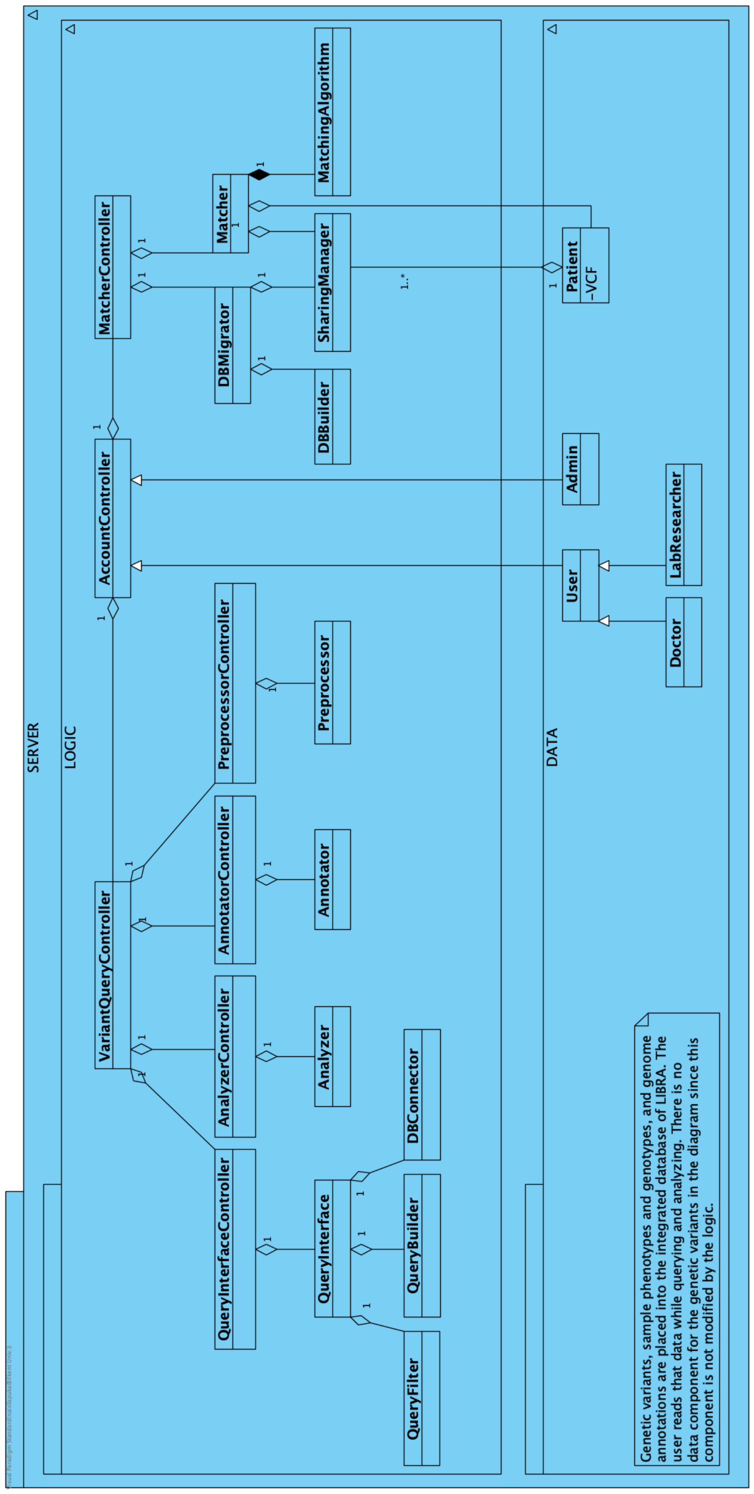
The logic of the client is responsible for triggering the execution of operations like VCF uploading and annotation, variant querying and analyzing and matching patients algorithms. These are achieved through communication with the server, depending on the events the user created in the views of the 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.

4.2 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.



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 this component is not modified by the logic.

Figure 4: Hierarchy of Classes inside the Server Subsystem

4.2.1 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.

4.2.2 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 New Knowledge Acquired and Learning Strategies Used

We started by locating the suitable technologies for our project. We knew that there will be separate tasks for individual members and these tasks will be merged one day. We needed a language that is capable of easy prototyping and decoupling so that separate tasks can be assigned to individual members. In turn, we have found React JS which is a good candidate to serve these purposes because of the separate Component structures and State based logic.

For learning, we have collected several resources related to the technologies that we are planning the use such as React JS, Django and general web development principles. These resources include video tutorials for the technologies and documentation related to them. We have agreed on a list of tutorials so that the teams knowledge base is similar to one another.

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/