Bilkent University

Department of Computer Engineering

Senior Design Project

LIBRA: Genetic Filtering and Diagnosis Matching System

Low 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

Low Level Design Report
Feb 17, 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.

To meet this demand, LIBRA aim 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.

1.1 Object Design Trade-offs

1.1.1 Functionality vs Usability

LIBRA will mostly operate on hospitals where clinicians don't have time to teach themselves how a software works in its way. Therefore, the functionality that GEMINI allows querying on genome data in a more advanced and complex way must be limited to functions that a non-tech person can use it without thinking over how those genome queries work to increase usability.

1.1.2 Security vs Cost

Since patient related data cannot be shared by institutions with other institutions without any privacy, we need to give the ability to share or hide some information fields of a patient or patient's profile altogether in our system. However, we need to save this patient information to show it to the creator of patient profiles for further use. Therefore, for every patient profile, we need to encrypt

the VCF data and other patient information in our system to prevent its violation of privacy. To secure the system without giving up speed and responsiveness of our system, we need to spend more on cloud services.

1.1.3 Rapid Development vs Functionality

The system aims to provide as many functionalities as possible, considering that the project from which we were inspired, GEMINI, has many of them. Functionality already has usability as an obstacle. However, our very first focus is rapid development, since we need to deliver a well-working program within three months. This rapid development may lead to less functionalities supported, based on the amount of time we will have at hand. Nevertheless, the main functionalities of LIBRA will be reliably and rapidly delivered.

1.2 Interface documentation guidelines

Class	*Name of the class*
Description of the class	
Attributes	
Name of an attribute	*Description of that attribute*
Methods	
Signature of a method **method(args)**	*Description of that method*

1.3 Engineering Standards

Throughout this report, as well as in our previous ones, we have used the Unified Modeling Language Standard (UML) [3] for the diagrams depicting the details of the system. Also, IEEE (Institute of Electrical and Electronics Engineers) [4] citation style has been used in our references.

1.4 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.

2 Packages

The large-scale architecture of the application will follow the client-server model, and the packages are designed following that fashion. 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.

2.1 Packages' 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.

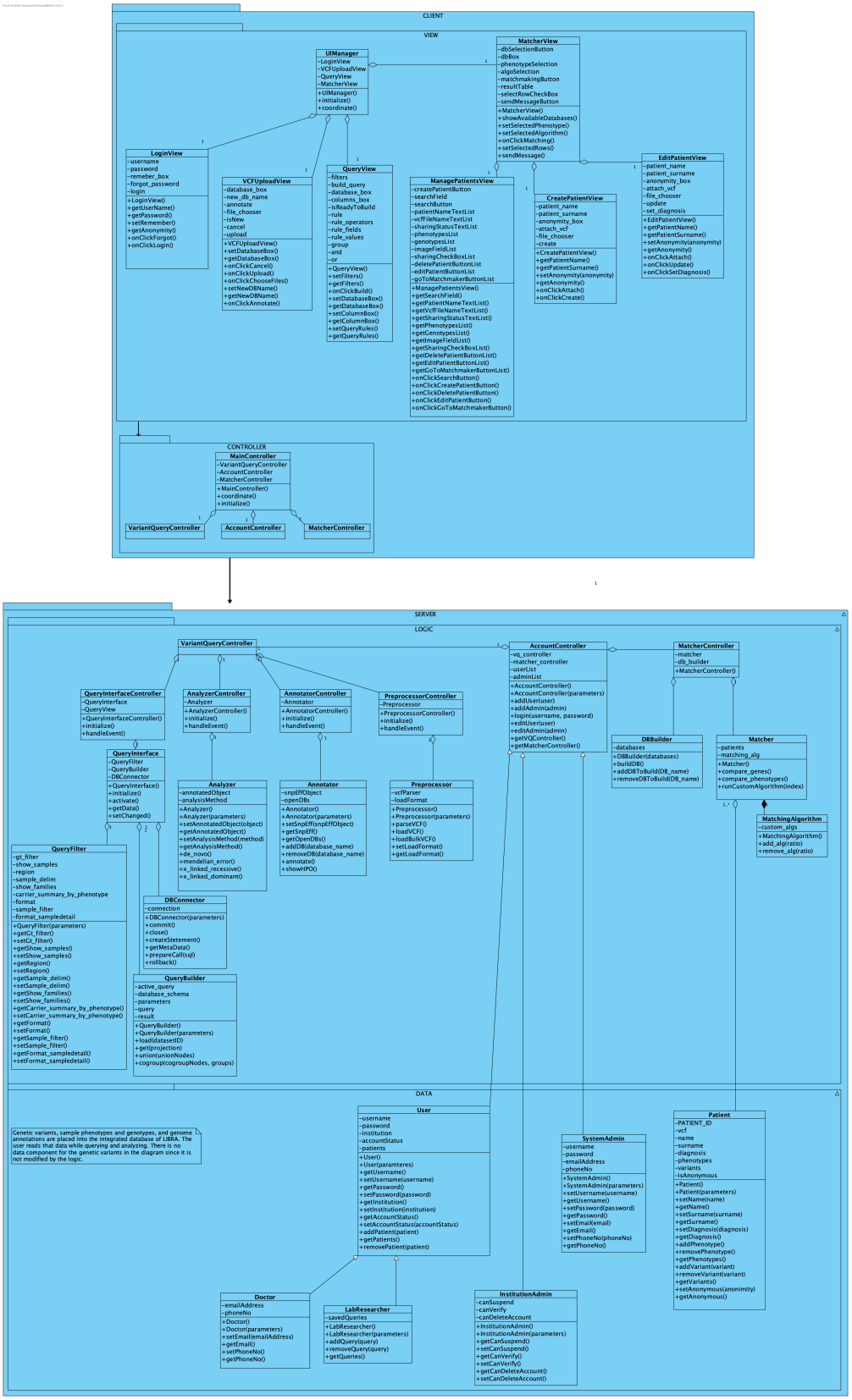

Figure 1: Full Diagram of the System of LIBRA

2.2 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.

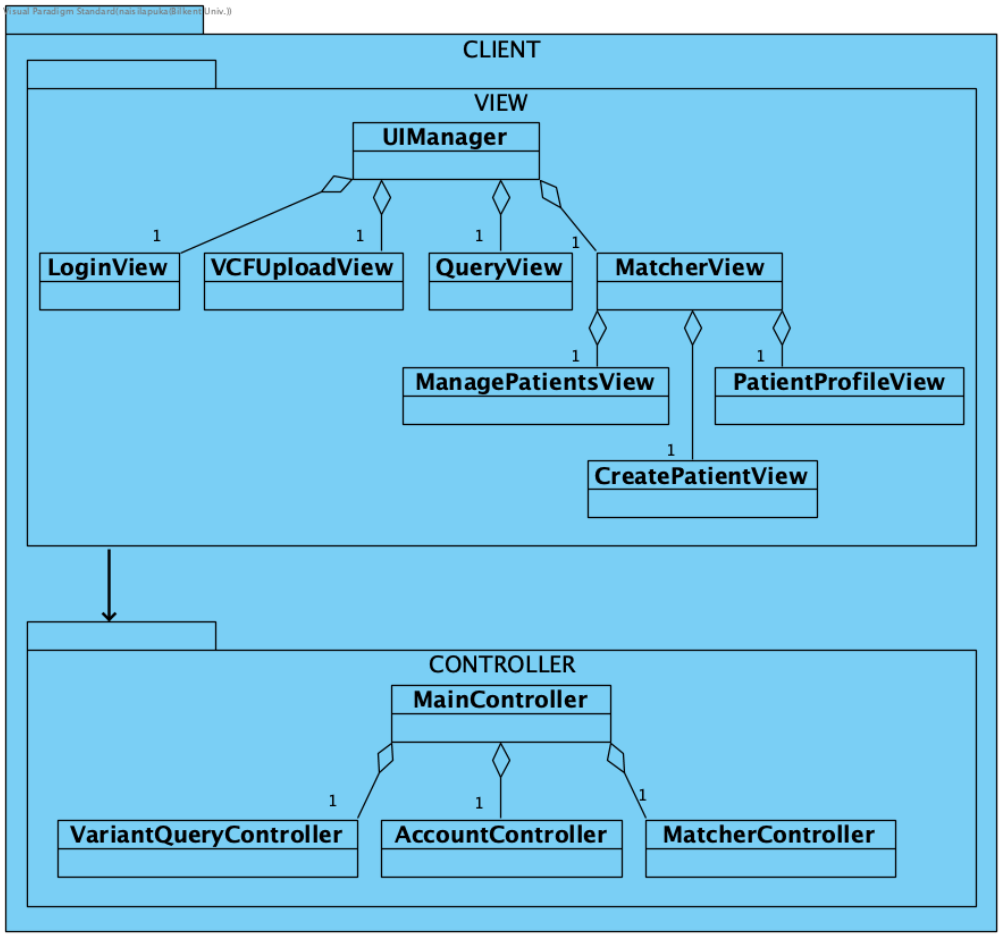

Figure 2: Hierarchy of Classes inside the Client Subsystem

2.2.1 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.

2.2.2 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.

2.3 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.

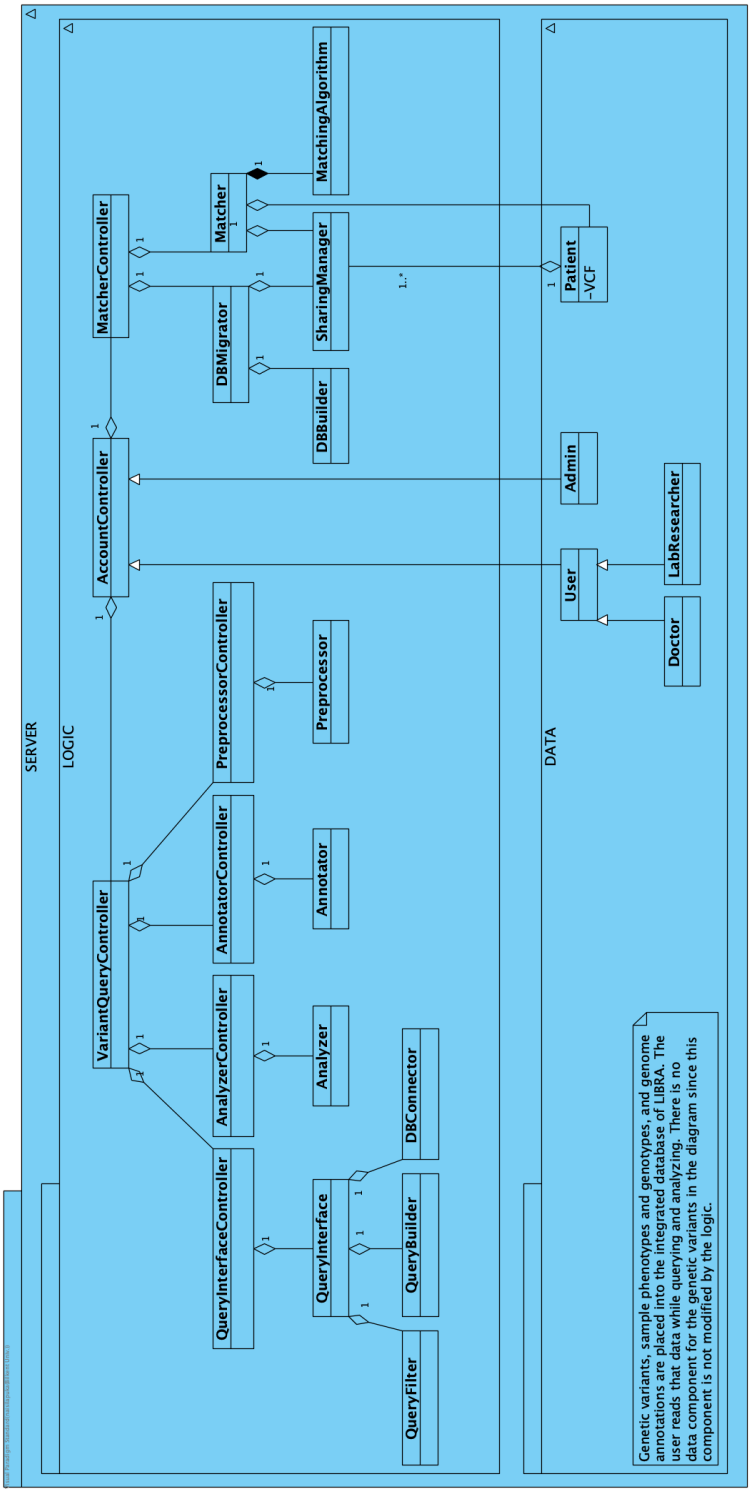

Figure 3: Hierarchy of Classes inside the Server Subsystem

2.3.1 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.

2.3.2 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.

3 Class Interfaces

3.1 Client

3.1.1 View

Class	UIManager
This class acts as a global manager for all the UI layers of LIBRA.	
Attributes	
LoginView	The LoginView instance the module will control
QueryView	The QueryView instance the module will control
MatcherView	The MatcherView instance the module will control
VCFUploadView	The VCFUploadView instance the module will control
Methods	
UIManager()	Default constructor for the UIManager class.
initialize()	Initilalizer method for the module. Creates and connects tall the attribute instances
coordinate()	Coordinator method for the class.Switches between tabs and communicates necessary information to controllers

Class	LoginView
This is the view class for the Login page	
Attributes	
username	Text box for the username
password	Text box for the password
remember_box	Check box for remember me functionality
forgot_password	Hyperlink for forgot_password page
login	Button for login
Methods	
LoginView ()	Default constructor for the LoginView class.
getUserName()	Accessor method for the username
getPassword()	Accessor method for the password
setRemember(remember)	Mutator method for the remember_box
getRemember()	Accessor method for the remember_box
onClickForgot()	Method that will redirect to the forgot password page
onClickLogin()	Method that will initiate the login process

Class	VCFUploadView
This is the view responsible for VCF Upload and Annotation	
Attributes	
database_box	Dropdown box to choose the database in which the user wants to save the new uploaded data from the VCF
new_db_name	Text box to put the name of the new database to be created from a singleton VCF
annotate	Button that responds to annotating VCF files
file_chooser	FileDialog that is responsible for choosing the files to upload
isNew	Boolean flag that shows whether the user has chosen the "New Singleton VCF Database" option
cancel	Button that responds to cancelling the procedure
upload	Button for uploading VCF files
Methods	
VCFUploadView(parameters)	Default constructor for the VCFUploadView.
setDatabaseBox()	Mutator for the database dropdown box default value
getDatabaseBox()	Accessor for the database dropdown box default value
onClickCancel()	Method to perform the necessary operation when Cancel button is active
onClickUpload()	Method to perform the necessary operation when Upload button is active
onClickChooseFiles()	Method to perform the necessary operation when files are chosen
setNewDBName()	Mutator for the text value of the new database name in the text box
getNewDBName()	Accessor for the text value of the new database name in the text box
onClickAnnotate()	Method to perform the necessary operation when Annotate button is active

Class	QueryView
This is the view responsible for building a query	
Attributes	
filters []	Array of Check Boxes, each check box elements represents a filter
build_query	Button that responds to building a query. When clicked, it will show the query builder and transfer the selected filters to it.
database_box	Dropdown box to choose the database from which the user wants to query
columns_box	Dropdown box to choose the column(s) the user is interested in
isReadyToBuild	Boolean flag that shows whether the user has chosen the filters and has clicked on "Build Query" button
rule []	Array of Buttons each of which represents a rule in the query
rule_operators[]	A list of dropdown boxes, each of which representing the operator of the rule in this particular column and in this particular group of and/or conditions
rule_fields[]	A list of dropdown boxes, each of which representing the column in which there is a rule in this particular group of and/or conditions
rule_values[]	A list of text fields, each of which representing the threshold value of a particular rule in a particular column
group []	Array of Buttons each of which represents the build process of a new group of conditions on the query
and []	Array of Buttons representing the "and" conditions in the query
or []	Array of Buttons representing the "or" conditions in the query
Methods	
QueryView(parameters)	Default constructor for the QueryView.
setFilters()	Mutator for the filters of the query based on the checkbox values
getFilters()	Accessor for the filters of the query based on the checkbox values
onClickBuild()	Method to perform the necessary operation when Build button is clicked
setDatabaseBox()	Mutator for the database dropdown box default value
getDatabaseBox()	Accessor for the database dropdown box default value
setColumnBox()	Mutator for the columns dropdown box default value(s) (note that the user can pick multiple columns here)
getColumnBox()	Accessor for the columns dropdown box default value(s)
setQueryRules()	Mutator for the query rules
getQueryRules()	Accessor for the query rules

Class	MatcherView
The view class for the matchmaking functionality.	
Attribute	
dbSelectionButton	A button that opens up a checkbox for the available databases to be used in the matchmaking process.
dbBox	Check box for the available databases in the matchmaking process.
phenotypeSelection	Drop down list for the phenotypes that are present in a patient and will be searched for in the matchmaking process.
algoSelection	Radio buttons for the possible algorithms to choose from for the matchmaking process.
matchmakingButton	A button that starts the matchmaking process.
resultTable	A table object that contains the results for the matchmaking process with the phenotype similarities.
selectRowCheckBox	A check box list for the rows of the resultTable.
sendMessageButton	A button to send messages to the doctors of selected patients in the resultTable.
Methods	
MatcherView()	Default constructor for MatcherView class.
showAvailableDatabases()	Method that displays the check box list of available databases.
setSelectedPhenotype()	Mutator method for the phenotypeSelection.
setSelectedAlgorithm()	Mutator method for the algoSelection.
onClickMatching()	Method that will initiate the matchmaking process.
setSelectedRows()	Method that will set the rows selected in the resultTable.
sendMessage()	Method that will send messages to the doctors of the patients in the selected rows of resultTable.

Class	ManagePatientsView
This view is responsible for the UI of the management of patients of that account.	
Attributes	
createPatientButton	Button for switching from Manage Patient View to Create Patient View to create a patient.
searchField	The text field to enter search phrases over patient to filter them
searchButton	The button for calling search algortihm for what searchField has.
patientNameTextList	Text fields for patient name
vcfFileNameTextList	VCF file names that is related with patients
sharingStatusTextList	Text fields for indicating the status of sharing of patients to show it is private or not
phenotypesList	Phenotypes of the patients
genotypesList	Genotypes of the patients
imageFieldList	Image fields to put images that related to disease of the patient for diagnosis
sharingCheckBoxList	Check boxes to indicate and change sharing status
deletePatientButtonList	Buttons to delete patient
editPatientButtonList	Buttons for switching from Manage Patient View to Edit Patient View to edit a patient.
goToMatchmakerButtonList	Buttons for switching from Manage Patient View to Matchmaker View for the patient
Methods	
ManagePatientView()	Default constructor for the ManagePatientView class.
getSearchField()	Get the text in searchTextField
getPatientNameTextList	Get text fields for patient name
getVcfFileNameTextList	Get VCF file names that is related with patients
getSharingStatusTextList	Get Text fields for indicating the status of sharing of patients to show it is private or not
getPhenotypesList	Get phenotypes of the patients
getGenotypesList	Get genotypes of the patients
getImageFieldList	Get image fields to put images that related to disease of the patient for diagnosis
getSharingCheckBoxList	Get check boxes to indicate and change sharing status
getDeletePatientButtonList	Get buttons to delete patient
getEditPatientButtonList	Get buttons for switching from Manage Patient View to Edit Patient View to edit a patient.
getGoToMatchmakerButtonList	Get buttons for switching from Manage Patient View to Matchmaker View for the patient
OnClickSearchButton()	A method to call search algortihm on patients to filter them on view on click
OnClickCreatePatientButton()	A method to switch to Create Patient View from Manage Patients View on click
OnClickDeletePatientButton()	A method to delete patient on click
OnClickEditPatientButton()	A method to switch to Edit Patient View from Manage Patients View on click
OnClickGoToMatchmakerButton()	A method to switch to Matchmaker View from Manage Patients View on click

Class	CreatePatientView
This is the view class for creating new patients	
Attributes	
patient_name	Text box for the name of the patient
patient_surname	Text box for the surname of the patient
anonymity_box	Check box for determining the anonymity of the field
attach_vcf	Button object for attaching vcf file
file_chooser	FileDialog that is responsible for choosing the files to upload
create	Button object for creating the patient
Methods	
CreatePatientView()	Default constructor for the CreatePatientView class.
getPatientName()	Accessor method for the patient_name
getPatientSurname()	Accessor method for the patient_surname
setAnonymity()	Mutator method for the anonymity_box
getAnonymity(anonymity)	Accessor Method for the anonymity_box
onClickAttach ()	Method that will display the file_chooser and initiate upload process
onClickCreate ()	Method that will initiate the creation process

Class	EditPatientView
This is the view class for editing the existing patient information	
Attributes	
patient_name	Text box for the name of the patient
patient_surname	Text box for the surname of the patient
anonymity_box	Check box for determining the anonymity of the field
attach_vcf	Button object for attaching vcf file
file_chooser	FileDialog that is responsible for choosing the files to upload
update	Button object for editing the patient
set_diagnosis	Button for setting diagnosis
Methods	
EditPatientView()	Default constructor for the EditPatientView class.
getPatientName()	Accessor method for the patient_name
getPatientSurname()	Accessor method for the patient_surname
setAnonymity(anonymity)	Mutator method for the anonymity_box
getAnonymity()	Accessor Method for the anonymity_box
onClickAttach()	Method that will display the file_chooser and initiate upload process
onClickUpdate()	Method that will update the changed values
onClickSetDiagnosis()	Method that will initiate the diagnosis setting process

3.1.2 Controller

Class	MainController
This class acts as a global manager for all the controller components of LIBRA.	
Attributes	
VariantQueryController	The VariantQueryController instance the module will control
AccountController	The AccountController instance the module will control
MatcherController	The MatcherController instance the module will control
Methods	
UIManager()	Default constructor for the UIManager class.
initialize()	Initilalizer method for the module. Creates and connects tall the attribute instances
coordinate()	Coordinator method for the class.Switches between tabs and communicates necessary information to controllers

3.2 Server

3.2.1 Logic

3.2.1.1 Genetic Variation Query Interface

Class	PreprocessorController
Controller class for the Preprocessor	
Attributes	
Preprocessor	The Preprocessor instance that will be controlled.
Methods	
PreprocessorController()	Default constructor for the PreprocessorController class.
initialize()	İnitializer method that class the necessary methods for setting the Preprocessor class up.
handleEvent ()	Handler method for the class. Its functionaility is based on the event triggered by the user interface

Class	Preprocessor
This class is responsible for loading and parsing vcf files for getting them ready to being annotated. It has the ability to load bulk data with specific formats	
Attributes	
vcfParser	The instance of a vcf parser
loadFormat	Data format for bulk loading
Methods	
Preprocessor()	Default constructor for the Preprocessor class.
parseVCF()	Method for parsing the vcf file.
loadVCF()	Method for loading a vcf file, this has been seperated from the parser since the files can be quite large
loadBulkVCF()	This method is used for loading bulk data.
setLoadFormat()	Mutator method for the load format used in bulk loading.
getLoadFormat()	Accessor method for the load format used in bulk loading.

Class	AnnotatorController
Controller class for the Annotator	
Attributes	
Annotator	The Annotator instance that will be controlled.
Methods	
AnnotatorController()	Default constructor for the AnnotatorController class.
initialize()	İnitializer method that class the necessary methods for setting the Annotator class up.
handleEvent ()	Handler method for the class. Its functionaility is based on the event triggered by the user interface

Class	Annotator
This class is responsible for annotating the preprocessed vcf files	
Attributes	
snpEffObject	snpEff instance that will be used for annotation
openDBs	List of databases that will be consulted during the annotation process
Methods	
Annotator()	Default constructor for the Annotator class.
setSnpEff(snpEffObject)	Mutator method for the snpEffObject
getSnpEff()	Accessor Method for the snpEffObject
getOpenDBs()	Access the list of open databases
addDB(database_name)	Add database to the database list
removeDB(database_name)	Remove database from the database list
annotate()	Annotate the vcf using snpEff by consulting the databases
showHPO()	Also shows the possible HPO descriptions based on the annotated variants.

Class	Analyzer Controller
Controller class for the Analyzer	
Attributes	
Analyzer	The Analyzer instance that will be controlled.
Methods	
AnalyzerController()	Default constructor for the AnalyzerController class.
initialize()	İnitializer method that class the necessary methods for setting the Analyzer class up.
handleEvent ()	Handler method for the class. Its functionaility is based on the event triggered by the user interface

Class	Analyzer
This class is responsible for doing analysis on an annotated vcf file	
Attributes	
annotatedObject	Instance of an object that contains the annotated file
analysisMethod	Different analysis methods can be specified via this object
Methods	
Analyzer()	Default constructor for the Analyzer class.
setAnnotatedObject(object)	Mutator method for the annotatedObject
getAnnotatedObject()	Accessor Method for the annotatedObject
setAnalysisMethod(method)	Mutator method for the analysisMethod
getAnalysisMethod(database_name)	Accessor Method for the analysisMethod
de_novo()	Method for considering de_novo mutations
Mendelian_error()	Method for calculating mendelian error
x_linked_recessive/dominant()	Method for considering x linked recessive or dominant inheritances.

Class	QueryInterfaceController
This class is responsible for controlling the process of building the queries, sending queries to the databases and fetching results.	
Attributes	
QueryInterface	The QueryInterface instance the module will control
QueryView	The QueryView instance the controller will communicate results to
Methods	
QueryInterfaceController()	Default constructor for the QueryInterfaceController class.
initialize()	Initilalizer method for the module. Creates and connects the two attribute instances
handleEvent()	Handler method for the class. Its functionality is based on the event triggered by the user interface

Class	QueryInterface
This class is responsible for building the queries, sending queries to the databases and fetching results.	
Attributes	
QueryFilter	The instance of QueryFilter that this class controls
QueryBuilder	The instance of QueryBuilder that this class controls
DBConnector	The instance of DBConnector that this class controls
Methods	
QueryInterface()	Default constructor for the QueryInterface.
initialize()	Initilalizer method for the module. Creates and connects the three attribute instances
activate()	Activator method for the class. It makes sure the connection with the server is established and commands can be run and data can be retrieved
getData()	Gets the data collected from the QueryBuilder in association with the DBConnector
setChanged()	Modifies any model that needs to be changed in the core database. This method might not be necessary in case nothing is modified during querying.

Class	QueryBuilder
This class creates a database query based on the user input.	
Attributes	
active_query	Provides access to the current query, which might be a subquery of the query member
database_schema	Provides access to the interface of the Database Schema Tree
parameters	Allows to retrieve information about parameters that were used in the query
query	Provides access to the main query
result	The result of the specified query
Methods	
QueryBuilder(parameters)	Default constructor for the query builder.
load(datasetID)	Loads a stream from a dataset
count()	Calculates the number of roes that match the query criteria
get(projection)	Query by selecting specific attributes
union(unionNodes)	Combine multiple result sets into one result set
cogroup(cogroupNodes, groups)	Two input tables are grouped independently and arranged side by side

Class	QueryFilter
This class sets the filters for a specific query.	
Attributes	
gt_filter	Boolean flag for filtering on genotypes
show_samples	Boolean flag for finding out which samples have a variant
region	Boolean flag for restricting a query to a specific region
sample_delim	Boolean flag for changing the sample list delimiter
show_families	Boolean flag for finding out which families have a variant
carrier_summary_by_phenotype	Boolean flag for summarizing carrier status
format	Boolean flag for reporting query output in an alternate format
sample_filter	Boolean flag for restricing a query to specified samples
format_sampledetail	Boolean flag for providing a flattened view of samples
Methods	
QueryFilter(parameters)	Default constructor for the QueryFilter class
getGt_filter()	Accessor for gt_filter of the QueryFilter.
setGt_filter()	Mutator for gt_filter of the QueryFilter.
getShow_samples()	Accessor for show_samples of QueryFilter.
setShow_samples()	Mutator for show_samples of QueryFilter.
getRegion()	Accessor for region of QueryFilter
setRegion()	Mutator for region of QueryFilter
getSample_delim()	Accessor for sample_delim of QueryFilter
setSample_delim()	Mutator for sample_delim of QueryFilter
getShow_families()	Accessor for show_families of QueryFilter
setShow_families()	Mutator for show_families of QueryFilter
getCarrier_summary_by_phenotype()	Accessor for carrier_summary_by_phenotype of QueryFilter
setCarrier_summary_by_phenotype()	Mutator for carrier_summary_by_phenotype of QueryFilter
getFormat()	Accessor for format of QueryFilter
setFormat()	Mutator for format of QueryFilter
getSample_filter()	Accessor for sample_filter of QueryFilter
setSample_filter()	Mutator for sample_filter of QueryFilter
getFormat_sampledetail()	Accessor for format_sampledetail of QueryFilter
setFormat_sampledetail()	Mutator for format_sampledetail() of QueryFilter

Class	DBConnector
This class is responsible for establishing the connection with the database of the user. Statements are executed and results are returned within the context of a connection.	
Attributes	
connection	Connection instance of the DBConnector classs
Methods	
DBConnector(parameters)	Default constructor for the DBConnector.
commit()	Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this DBConnector object.
close()	Closes the connection.
createStatement()	Creates a Statement object for sending statements to the database.
getMetadata()	Retrieves a DatabaseMetaData object that contains metadata about the database to which this Connection object represents a connection.
prepareCall(String sql)	Creates a CallableStatement object for calling database stored procedures.
rollback()	Undoes all changes made in the current transaction and releases any database locks currently held by this DBConnector object.

3.2.1.2 Matchmaking System

Class	MatcherController
This controller is responsible for executing operations of matchingpatients algorithms based on the events triggered through the views	
Attributes	
matcher	instance of Matcher class which handles patient matching
db_builder	instance of DBBuilder class which provides adding new databases to the matcher
Methods	
MatcherController( )	Default constructor for the MatcherController.

Class	Matcher
This class contains instances of many objects related to patient matching and orchestrate them	
Attributes	
patients	array of patients which will be matched according their genes and phenotypes
matching_alg	instance of MatchingAlgorithm object which decides to match two patients
s_manager	instance of SharingManager object which handles shared information issues
Methods	
MatcherController( )	Default constructor for the Matcher.
compare_genes( )	method which matches patients according to genes
compare_phenotypes( )	method which matches patients according to phenotypes
runCustomAlgorithm(index)	method which matches patients according to custom algortihm with index

Class	MatcherAlgorithm
This controller is responsible for executing operations of matchingpatients algorithms based on the events triggered through the views	
Attributes	
custom_algs	list of algorithms' ratio of phenotype and genotype factors
Methods	
MatcherController( )	Default constructor for the MatcherAlgorithm.
add_alg(ratio)	add custom algortihm with ratio
remove_alg(ratio)	remove custom algortihm with ratio

Class	DBBuilder
This class is used for setting the database up and running on LIBRA servers.	
Attributes	
List<String> databases	List of databases that will be built.
Methods	
DBBuilder(List<String> databases)	This constructor will initiialize the database list that will be set up and running
buildDB()	Build the databases that are on the list and return true if it is successfull
addDBToBuild(String DB_name)	A method to add a new database that currently built
removeDBToBuild(String DB_name)	A method to remove a database that currently built

3.2.2 Data

Class	User
Model Class to contain the data common to all users.	
Attributes	
username	The username of the user.
password	The password of the user.
affiliatedInstitution	The institution the user works for.
accountStatus	The status of the user account that denotes whether the account is verified, not verified suspended etc.
Methods	
User()	Default constructor for the users.
User(username, password, affiliatedInstitution, accountStatus)	Custom constructor that creates a user based on the input entered.
setUsername(newUsername)	Mutator method for the username. The user has to enter a new, unique username
getUsername()	Accessor method for the username.
setPassword(newPassword)	Mutator method for the password.
getPassword()	Accessor method for the password.
setInstitution(newInstitution)	Mutator method for the institution the user is working for.
getInstitution()	Accessor method for the institution the user is working for.
setAccountStatus(newAccountStatus)	Mutator method for the status of the user's account.
getAccountStatus()	Accesor method for the status of the user's account.

Class	SystemAdmin
Model class that contains data of system administrators.	
Attributes	
username	The username of the system administrator.
password	The password of the system administrator.
emailAddress	E-mail address of the user in case another doctor has to contact the user.
phoneNo	Phone number of the user in case another doctor has to contact the user.
Methods	
SystemAdmin()	Default constructor for the system administrator.
SystemAdmin(username, password, emailAddress, phoneNo)	Custom constructor that creates a system administrator based on the input entered.
setUsername(newUsername)	Mutator method for the username. The user has to enter a new, unique username
getUsername()	Accessor method for the username.
setPassword(newPassword)	Mutator method for the password.
getPassword()	Accessor method for the password.
setEmail(newEmailAddress)	Mutator method for the user's email address.
getEmail()	Accessor for the email of the user.
setPhoneNo(newPhoneNo)	Mutator method for the user's phone number.
getPhoneNo()	Accessor for the phone number of the user.

Class	Doctor
Model class to contain data specific to users classified as "Doctor."	
Attributes	
patients	The list of patients created by and associated with the Doctor.
emailAddress	E-mail address of the user in case another doctor has to contact the user.
phoneNo	Phone number of the user in case another doctor has to contact the user.
Methods	
Doctor()	Default constructor for the doctor.
Doctor(username, password, affiliatedInstitution, accountStatus, emailAddress, phoneNo)	Custom constructor that creates a doctor based on the input entered.
setEmail(newEmailAddress)	Mutator method for the user's email address.
getEmail()	Accessor for the email of the user.
setPhoneNo(newPhoneNo)	Mutator method for the user's phone number.
getPhoneNo()	Accessor for the phone number of the user.
addPatient(patient)	Doctor adds a patient he is handling to the system.
getPatients()	Returns all of the patients that are associated with the Doctor.
removePatient(Patient)	Removes a patient from the patients list.

Class	Patient
Model class that contains data related to patients.	
Attributes	
PATIENT_ID	Unique patient ID that is used to identify unique patients and cannot be used to leak privacy of the patient. It has constant value.
vcf	VCF file of the patient that is added by the user.
name	Name of the patient.
surname	Surname of the patient.
diagnosis	Diagnosis of the patient as decided by the patient's doctor/s
phenotypes	A list of phenotypes of the patient.
variants	A list of variants observed in the patient.
isAnonymous	A boolean flag that sets the privacy setting of the patient's personal information.
Methods	
Patient()	Default constructor for the patient.
Patient(name, surname, diagnosis, phenotypes, variants, isAnonymous)	Custom constructor that creates a patient based on the input entered.
setName(newName)	Mutator method for the name of the patient.
getName()	Accessor method for the name of the patient.
setSurname(newSurname)	Mutator method for the surname of the patient.
getSurname()	Accessor method for the surname of the patient.
setDiagnosis(newDiagnosis)	Mutator method for the diagnosis of the patient.
getDiagnosis()	Accessor method for the diagnosis of the patient.
addPhenotype(newPhenotype)	Adds a new phenotype to the patient's list of phenotypes.
removePhenotype(removedPhenotype)	Removes a phenotype from the patient's list of phenotypes.
getPhenotypes()	Returns the list of phenotypes observed in the patient.
addVariant(newVariantName)	Adds a new variant to the patient's list of phenotypes.
removeVariant(variantToBeDeleted)	Removes a variant from the patient's list of phenotypes.
getVariants()	Returns the list of variants observed in the patient.
setAnonymous(newAnonimitySetting)	Mutator method for the anonimity of the patient.
getAnonymous()	Accessor method for the anonimity setting of the patient.

Class	Lab Researcher
Model class that contains data related to the lab researcher.	
Attributes	
savedQueries	A list of complex queries the user saved for later reuse.
patients	The list of patients created by and associated with the Doctor.
Methods	
LabResearcher()	Default constructor for the lab researcher class.
LabResearcher(username, password, affiliatedInstitution, accountStatus)	Custom constructor for the lab researcher class.
addQuery(newQuery)	Adds a new query to the patient's list of saved queries.
removeQuery(removedQuery)	Removes a query from the patient's list of saved queries.
getQueries()	Returns the list of queries observed in the patient.
addPatient(patient)	Lab Researcher adds a patient he is handling to the system.
getPatients()	Returns all of the patients that are associated with the Lab Researcher.
removePatient(Patient)	Removes a patient from the patients list.

Class	InstitutionAdmin
Model class to contain data specific to an institution admin.	
Attributes	
canSuspend	Boolean flag that permits instituiton admin to suspend a user.
canVerify	Boolean flag that permits instituiton admin to verify a user.
canDeleteAccount	Boolean flag that permits instituiton admin to delete a user account.
Methods	
InstitutionAdmin()	Default constructor for the institution administrator.
InstitutionAdmin(username, password, affiliatedInstitution, accountStatus, canSuspend, canVerify, canDeleteAccount)	Custom constructor that creates a institution administrator based on the input entered.
getCanSuspend()	Accessor for suspension permission of the institution admin.
setCanSuspend()	Mutator for suspension permission of the institution admin.
getCanVerify()	Accessor for verification permission of the institution admin.
setCanVerify()	Mutator for verification permission of the institution admin.
getCanDeleteAccount()	Accessor for account deletion permission of the institution admin. Deletable accounts belong to other users.
setCanDeleteAccount()	Mutator for account deletion permission of the institution admin.

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/

[3] "Unified modeling language." [Online]. Available: https://en.wikipedia.org/wiki/Unified_Modeling_Language

[4] "Citation styles - apa mla chicago turabian ieee - ieee style." [Online]. Available: https://pitt.libguides.com/citationhelp/ieee