# Group Project Report

## Group 1

## Abstract:

We aim to develop a web application that provides drug information for users in this group project. Based on the variant analysis pipeline workflow, the output annotated file was selected to match with variant annotations provided by pharmGKB. A three-tier architecture with Java interface simplifying this code structure are used for software design. Relevant data are retrieved from the database and the acquired information is then displayed in the user interface. Key functions include searching, user login registration, query record operation and drug related information display. Testing is separated into two parts, each of which has shown generally nice test results. Low-level uniting test focuses on the code and validates whether each unit can perform as expected. High-level system function testing covers the functionality and usability, interface, compatibility, performance and security of the project.

To conclude, our project can run successfully with little errors, but it still has some limitations. The web application needs to connect with a local database, and we tend to deploy it to cloud servers. The performance and the accuracy can be improved by using Spring MVC pattern in later development and using WebSocket to realize real-time refresh of the front-end pages when database is updated, respectively.

## Introduction:

### Background of objective:

People who have a family history of a certain inherited disease and have high risks of developing it usually hope to get drug recommendation. Medical workers also need drug information like dosing guidelines for research. Therefore, a web application can be developed for the public to get information related to drugs.

### Objective:

To develop an online web application that enables the general public to upload files about variant annotation data or just search for the drug information. This web application should also include basic functions such as signing in and showing the searching history.

### Background of technologies:

1. ***Variant Effect Predictor*** (VEP web interface version) is a toolset for the analysis,

annotation, and prioritization of genomic variants.

2. ***Java Development Kit (JDK)*** (v12.0.2) is a software development environment for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.
3. ***JetBrains IDEA*** is an integrated development environment (IDE) for Java. It understands and provides intelligent coding assistance for a large variety of other languages such as SQL, HTML, and JavaScript (IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains, 2020).
4. ***Apache Tomcat*** (v7.0) provides a HTTP web server environment in which Java code can run. When using *JetBrains IDEA, Tomcat server* is edited in configurations to run the web application.
5. ***MySQL*** (v8.0) is a common open-source relational database management system which performs SQL queries, database inserts and deletes.
6. ***JDBC*** is a Java-based data access technology that offers methods to query and execute data in a database
7. ***Apache Maven*** (v3.1) is a software project management and comprehension tool based on the concept of a project object model (POM).
8. ***Junit*** (v4.11) is a unit testing framework for the Java programming language. It is used to write repeatable tests for the application code units.
9. ***Servlet & JSP****:* Servlet is a server-side module that deals with client requests and implements servlet interfaces. It can respond to any requests and are commonly used to extend the applications from web servers (Vaidya, 2020). JSP is a more functional extension to Servlet. Web applications including dynamic webpages can be based on Java via Servlets and JSP.
10. ***JSTL*** (v1.2) extends JSP development by adding a tag library of JSP tags for usual tasks including XML data processing, conditional execution and database access.
11. ***SLF4J*** (v1.7) stands for Simple Logging Facade for Java, which gives a Java logging API and allows users to work with any logging frameworks such as Log4j.
12. ***Gson*** (v2.8) is an open-source Java library developed by Google to serialize and deserialize Java objects to JSON based on a data passing mechanism.

**Why these technologies fit:**

*Variant Effect Predictor* deals with variant annotation after variant calling and filtering for the raw sequencing data from the patients in the initial phase.

*JDK*, *JetBrains IDEA* and *Apache Tomcat* set up a basic development environment to start a web application project. *JetBrains IDEA* is based on pom.xml from *Maven* to automate the management of dependencies for a web-based Java project.

Information about drugs, drug labels and dosing guidelines can be stored and updated in *MySQL. JDBC* gives the database connection to let information display in the user interface which is based on *servlets and JSPs* extended by *JSTL. Gson* is used as the data should be in a text format JSON when exchanging between a browser and a server.

*Slf4j* meets the requirement of developing a system that clean logs are necessary to help understand what errors are being triggered, and what information is being processed at runtime. *Junit* enables the developers to find errors and get code optimization during the whole developing process. These technologies work together to ensure the development of an operable project.

## Methods:

### Sequence analysis:

Raw data processing, variant calling, and variant annotation were performed. The Variant Analysis Pipeline (VAP) workflow, a comprehensive guidance for variant identification designed for RNA sequencing results, was used as the reference for the workflow applied in this program (Adetunji et al., 2019). Based on the VAP workflow, we made some modifications to maximize variant identification efficiency and avoid errors caused by data or software incompatibility.

### Code:

We used a three-tier architecture as the standard for writing code. A 3-tier architecture is a client-server software architecture pattern that contains the UI (User Interface) layer, the BLL (Business Logic) layer and the DAL (Data access) layer (Fowler, 2002). The UI layer is the topmost level of the software, which is used to collect user input information and operation instructions, and to display the results of various services of different functions to the user. The BLL layer is the middle layer that connects the user interface with the data access layer. It gets user instructions and data from the UI and executes the business logic, and then retrieves the data from the DAL for UI display. The DAL layer is used to perform data operations including connecting to a database using *JDBC*, matching, selecting, or deleting specified data using SQL statements. It is the most critical layer of software functionality implementation.

Moreover, we used the Java interface to simplify the three-tier structure. Using Java interfaces and implementations helps to make the code structure clearer.

The implementation of our software adopts the 3-tier architecture. First, the UI layer is constructed according to the research and analysis of software requirements.

Then, the BLL layer services get the query and information from the UI and call the DAL to retrieve the relevant data from the database. Finally, the servlet displays the acquired information in the user interface (Figure 1).

Furthermore, the different three-tier architectural patterns that each functionality implementation relies on will be discussed in more detail in ‘Results’ part.

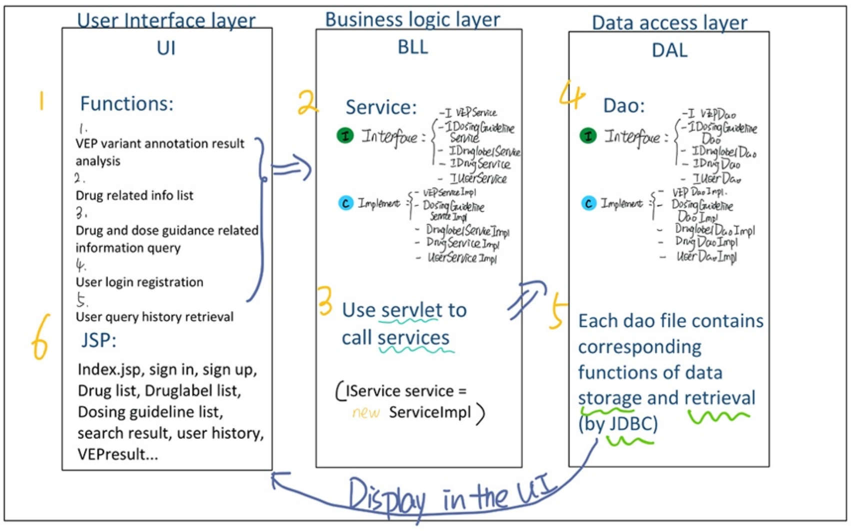

User Interface layer
UI
1
Functions:
1. VEP variant annotation result analysis
2. Drug related info list
3. Drug and dose guidance related information query
4. User login registration
5. User query history retrieval
6
JSP:
Index.jsp, sign in, sign up, Drug list, Druglabel list, Dosing guideline list, search result, user history, VEPresult...
Business logic layer
BLL
2
Service:
I Interface: - I VEPService - IDosingGuideline Service - IDruglabelService - IDrugService - IUserService
C Implement: - VEPServiceImpl - DosingGuideline ServiceImpl - DruglabelServiceImpl - DrugServiceImpl - UserServiceImpl
3
Use servlet to call services
(IService service = new ServiceImpl)
Data access layer
DAL
4
Dao:
I Interface: - I VEPDao - IDosingGuideline Dao - IDruglabelDao - IDrugDao - IUserDao
C Implement: - VEP DaoImpl. - DosingGuideline DaoImpl - DruglabelDaoImpl - DrugDaoImpl - UserDaoImpl
5
Each dao file contains corresponding functions of data storage and retrieval (by JDBC)
Display in the UI

Figure 1: The implementation of functions adopts the software architecture pattern of three-tier architecture. We build the UI layer according to requirement research and analysis, and then call the corresponding services in the BLL through servlets. After that, the DAL layer is called to realize database data operation, and finally the returned data is displayed on each web page.

**Test:**

There are two parts of testing: unit testing and system function testing. The low-level uniting test focuses on the code and aims to validate that each unit of the code can perform as expected, while the high-level system function testing focuses the usability and performance of the web application.

**Unit testing (low-level, white-box testing)**

In the unit testing part, *Junit4*, an open source unit testing framework for JAVA, is used to write repeatable test codes. *Junit* provides many methods to test the correct performance of the unit such as *asserTrue{}, assertEquals{}* and so on. Besides, *Junit* allows developers to create test cases and group them as test suites if they are related. In the unit test, we create test cases for methods in the ‘Appconfig’, ‘dao’, and ‘domain’ classes, and test cases for the same classes are grouped into an individual test suite, which means that by running the individual test suite all test cases inside it can be executed at the same time. In some of the test cases, we use *assertEquals{}* to test whether the output of the function is equal to the expected values. While in the other cases, we print out the result directly to check whether they are as expected.

Besides, an additional test case (DBtest) is designed to check whether the database has been created as expected, in which we compare the metadata of the created database with that of the raw data.

Meanwhile, we want to use *EasyMock* to generate mock objects to mock interfaces, which later can be added with functionality and used in unit testing. By recording, replying, and verifying the mocks, the correctness of the process can be validated. We originally planned to apply *EasyMock* in the test for 'SigninServlet', but finally this part is not completed.

**System function testing (high-level, black-box testing)**

As for the system function testing, we referred to an online 6-step guide to web application testing and made some modifications based on that. Therefore, there are five steps in the system function testing: functionality and usability testing, interface testing, compatibility testing, performance testing and security testing.

***Functionality and Usability testing***

The aim of this part is to test the functionality of the whole web application as well as the user experience. Based on the main functions of the system, we designed a series of test strategies to cover all the functionality (Table 2). Our group members are the test participants and run the tests.

***Interface testing***

This step of test examines whether the communication processes can run smoothly and whether error messages will show out if the user does not manipulate the application correctly.

***Compatibility testing***

The browser compatibility, operating system compatibility and mobile compatibility of the application are considered in this part of test.

***Performance testing***

We test the performance of the application in varied conditions such as different internet speeds (in the different times of a day) and different sizes of input file.

***Security testing***

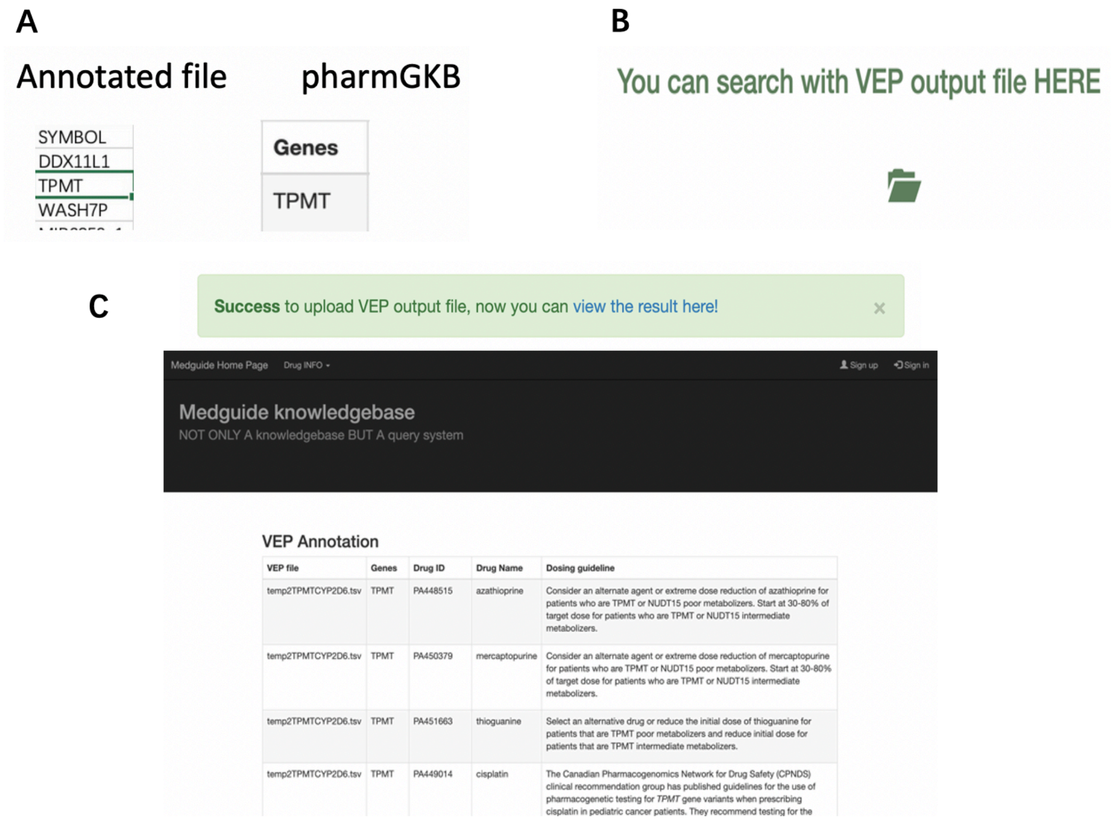
The security requirement of our web application is not high, therefore we only test whether the pages with authentication can be accessed if the user does not log in.

## Results:

### Sequence analysis：

With raw sequencing data from the patient, quality control was done with *FastQC* (v0.11.9). According to the quality report generated by *FastQC*, read trimming was performed with *cutadapt* (v1.9.1). After the data passed the quality control process, alignment was performed with *HISAT2* (v2.1.0). The reference genome recommended is GRCh38, as it is more compatible with some software used in downstream analysis. After converting the SAM input from *HISAT2* into BAM file with *samtools* (v1.2), the SortSam function of *Picard tools* (v2.22.3) was used to sort the BAM file. Read groups were then added with the addOrReplaceReadGroups function of *Picard tools*. Duplicates were marked with the MarkDuplicates function of *Picard tools*. For RNA sequencing data, the SplitNCigarReads function of *gatk* (v4.1.7.0) was called to hard clip reads with mismatches in their cigar string to reduce false variants calls in downstream analysis. After read processing, variant calling was performed with the HaplotypeCaller function in *gatk*. The VCF file produced was filtered according to quality score, read depth, and strand bias with the VariantFiltration function in *gatk*. After variant calling and filtering, variant annotation was performed with Variant Effect Predictor (VEP web interface version). The SYMBOL column of the output annotated file was then extracted to match with variant annotations provided by pharmGKB called Genes (Figure 2A). The user can upload annotated variant files by launching the “search with VEP output file” function (Figure 2B). After uploading the file, the user can preview the matching results (Figure 2C).

Data and software compatibility such as reference genome and annotation should be considered. Otherwise, incompatible data formats may produce truncated or even empty variant calling results.


A
Annotated file
pharmGKB
SYMBOL
DDX11L1
TPMT
WASH7P
Genes
TPMT
B
You can search with VEP output file HERE
C
Success to upload VEP output file, now you can view the result here!
Medguide Home Page
Drug INFO
Sign up
Sign in
Medguide knowledgebase
NOT ONLY A knowledgebase BUT A query system

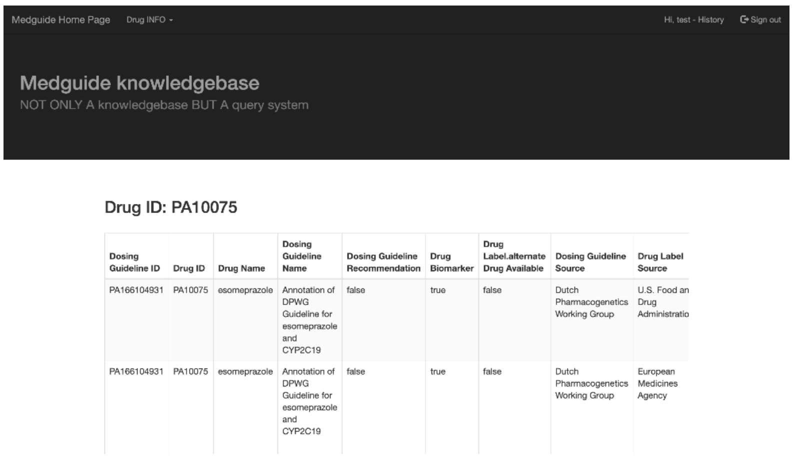
VEP Annotation

| VEP file | Genes | Drug ID | Drug Name | Dosing guideline |
|---|---|---|---|---|
| temp2TPMTCYP2D6.tsv | TPMT | PA448515 | azathioprine | Consider an alternate agent or extreme dose reduction of azathioprine for patients who are TPMT or NUDT15 poor metabolizers. Start at 30-80% of target dose for patients who are TPMT or NUDT15 intermediate metabolizers. |
| temp2TPMTCYP2D6.tsv | TPMT | PA450379 | mercaptopurine | Consider an alternate agent or extreme dose reduction of mercaptopurine for patients who are TPMT or NUDT15 poor metabolizers. Start at 30-80% of target dose for patients who are TPMT or NUDT15 intermediate metabolizers. |
| temp2TPMTCYP2D6.tsv | TPMT | PA451663 | thioguanine | Select an alternative drug or reduce the initial dose of thioguanine for patients that are TPMT poor metabolizers and reduce initial dose for patients that are TPMT intermediate metabolizers. |
| temp2TPMTCYP2D6.tsv | TPMT | PA449014 | cisplatin | The Canadian Pharmacogenomics Network for Drug Safety (CPNDS) clinical recommendation group has published guidelines for the use of pharmacogenetic testing for *TPMT* gene variants when prescribing cisplatin in pediatric cancer patients. They recommend testing for the |

Figure 2. Upload annotated variant file and preview the matching results. (A), annotated file matches with pharmGKB data. (B), The user can upload the annotated variant file by launching the "search with VEP output file" function. (C), The user can preview the matching results.

**Searching:**

Based on the drug search function in DAO, our web application supports two kinds of drug search function, including drug ID query and dosing guideline ID query. When users input a completely correct ID sequence, the client sends a request to the server. The server starts the servlet to call the service layer, which further calls the search function to obtain the corresponding drug information from drug database and then servlets transfer the content to the server. Finally, server sends the response to the client. The query results include dosing guideline ID, drug ID, drug name, dosing guideline name, dosing guideline recommendation, drug biomarker, dosing guideline summary markdown, etc. (Figure 3).



Medguide knowledgebase

NOT ONLY A knowledgebase BUT A query system

Drug ID: PA10075

| Dosing Guideline ID | Drug ID | Drug Name | Dosing Guideline Name | Dosing Guideline Recommendation | Drug Biomarker | Drug Label.alternate Drug Available | Dosing Guideline Source | Drug Label Source |
|---|---|---|---|---|---|---|---|---|
| PA166104931 | PA10075 | esomeprazole | Annotation of DPWG Guideline for esomeprazole and CYP2C19 | false | true | false | Dutch Pharmacogenetics Working Group | U.S. Food an Drug Administratio |
| PA166104931 | PA10075 | esomeprazole | Annotation of DPWG Guideline for esomeprazole and CYP2C19 | false | true | false | Dutch Pharmacogenetics Working Group | European Medicines Agency |

Figure 3. Query results about drug information in different aspects.

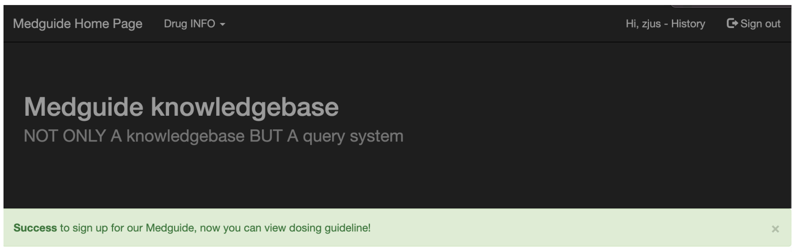
**User login registration and query record operation:**

User operation related functions of the software include login, registration and search record query, all three functions are implemented by a three-tier architecture.

The registration function requires the user to provide the email, unique username and password in the UI interface, the servlet then calls the BLL layer's service. The service executes the business logic, calls the ‘Userdao’ class of the DAL layer, performs the ‘saveUserInfo’ function to store the user information into the database. Finally, the filter class will permit the user to view the dosing guideline (Figure 4).

Once you have registered, the next time you use this software, you can log in with the username and password. The login function is also implemented through the three-tier architecture. Unlike the registration function, the dao file in the DAL layer executes the ‘findPassword’ function, which is used to match the registered user.

When registered or logged in, users can view their medication query history and choose whether to delete the history. The functions related to the historical record view and operation are realized by the corresponding BLL layer service class by calling the history-related functions such as ‘saveHistory’, ‘findHistory’, ‘countHistory’ and ‘deleteHistory’ in ‘Userdao’, and finally, the historical record information is returned and displayed on the user interface (Figure 5).

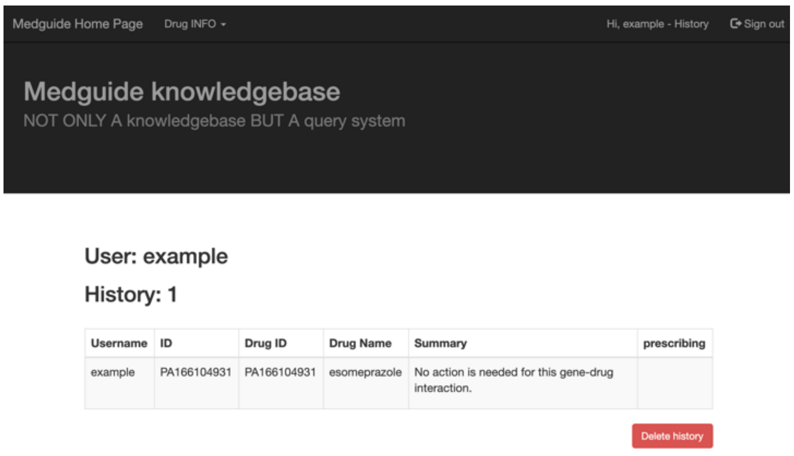


Medguide knowledgebase

NOT ONLY A knowledgebase BUT A query system

**Success** to sign up for our Medguide, now you can view dosing guideline!

Figure 4. The success page of sign up



Medguide knowledgebase

NOT ONLY A knowledgebase BUT A query system

User: example

History: 1

| Username | ID | Drug ID | Drug Name | Summary | prescribing |
|---|---|---|---|---|---|
| example | PA166104931 | PA166104931 | esomeprazole | No action is needed for this gene-drug interaction. | |

Figure 5. The user search history page

## Drug related information display:

The implementation of this function relies on the methods in five classes: ‘AppConfig’, ‘PharmGKBImporter’, ‘dao’, ‘DBUtils’, and ‘domain’. First, based on the url, username and password provided by the methods in the ‘AppConfig’ class, the getConnection() method in the ‘DBUtils’ class will get connection to the corresponding database. By calling the save***() methods in ‘dao’ class through the service and combining the methods in the ‘domain’ class, the import***() methods in the PharmGKBImporter class can get the content from the data file and import them into the tables in the connected database. In the servlets, the findAll() method, which is called through service, can get all the data from the corresponding table in the database. Finally, the return data will be displayed on the web page.

**Test:**

We got generally nice test results. For unit testing part, all tests for the functions, except that for *'*Signinservlet'*,* can pass successfully. However, it is must be mentioned that in the test cases for the 'dao' class, we test by adding new content into the created database, which means that the original database will be changed. Therefore, to ensure the functionality of our system, the new added contents should be deleted after every execution of these test cases.

For system function test, the test results in each step are summarized in Table1. Detailed results in the functionality and usability testing are provided in Table 2-7.

| Testing steps | Results |
|---|---|
| **Functionality and usability testing** | 25 out of 27 test cases pass successfully. Details are shown in Table 2-7. |
| **Interface testing** | The communication between interfaces are nice. However, two expected error messages are missing as shown in Table 4 and 6. |
| **Compatibility testing** | Browser compatibility: the application can be accessed in Microsoft edge and chrome and safari.<br><br>Operating system compatibility: the application can be accessed in both Windows and MacOS systems<br><br>Mobile compatibility: the application can run successfully in five different devices |
| **Performance testing** | the speed of showing the annovar results analysis gets lower when the internet speed gets lower or the size of the input file gets larger. The performance of other functions are good. |
| **Security testing** | The Drug doing guideline page will not be viewed for users who does not sign in or sign up. |

*Table 1 | Summary of system function testing results.*

| Function | Action (Test Cases) | Expected response | Actual response | Test results |
|---|---|---|---|---|
| Display the drug, drug labels and dosing guideline information | Click on the 'Drug INFO' | A select menu appears | A select menu appears | PASS |
| | Click on the Drug List in the menu before login | The page showing drug information appears | The page showing drug information appears | PASS |
| | Click on the DrugLabel List in the menu before login | The page showing drug label information appears | The page showing drug label information appears | PASS |
| | Click on the DrugGuideline List in the menu before login | The page asking the user to sign in/ sign up appears | The page asking the user to sign in/ sign up appears | PASS |
| | Click on the Drug List in the menu after login | The page showing drug information appears | The page showing drug information appears | PASS |
| | Click on the DrugLabel List in the menu after login | The page showing drug label information appears | The page showing drug label information appears | PASS |
| | Click on the DrugGuideline List in the menu after login | The page showing drug guideline information appears | The page showing drug guideline information appears | PASS |

*Table 2 | Test cases based on the displaying function and the test results.*

| Function | Action<br>(Test Cases) | Expected response | Actual response | Test results |
|---|---|---|---|---|
| Drug search | Type in drug id or dosing guideline id | The page displaying corresponding appears | The page displaying corresponding appears | PASS |
|  | Type in the id which is not included in the current database or others | "no match" appears | "no match" appears | PASS |

*Table 3 | Test cases based on the searching function and the test results.*

| Function | Action<br>(Test Cases) | Expected response | Actual response | Test results |
|---|---|---|---|---|
| VEP variant annotation results analysis | Click on the file picture on the home page | The page asking the user to upload a file appears | The page asking the user to upload a file appears | PASS |
|  | Click on the "upload your file" button | The file folder appear | The file folder appear | PASS |
|  | Selected the file with the same format in the folder | The file name show out on the page | The file name show out on the page | PASS |
|  | Click on the "upload" button after selecting | A response asking users to view the results appears | A response asking users to view the results appears | PASS |
|  | Click on the "view the results here!" | The page showing the results appears | The page showing the results appears | PASS |
|  | Upload a file with incorrect format | An error message will appear | An error message will appear | PASS |

*Table 4 | Test cases based on the VEP variant annotation results analysis function and the test results.*

| Function | Action<br>(Test Cases) | Expected response | Actual response | Test results |
|---|---|---|---|---|
| User login, registration and logout | Click on "sign in" button on the home page | The form appears | The form appears | PASS |
|  | Click on the "sign in" after filling the form | A response showing sign in successfully | A response showing sign in successfully | PASS |
|  | Click on the "sign up" button on the home page | The form appears | The form appears | PASS |
|  | Click on the "sign up" after filling the form | A response showing sign up successfully | A response showing sign up successfully | PASS |
|  | Type in an existed username | An error message appears | An error message appears | PASS |
|  | Type in an email address with incorrect format | An error message will appear | Sign up successfully | FAIL |
|  | Click on the "Sign out" button on the home page | Home page | Home page | PASS |

*Table 5 | Test cases based on the user login, registration and logout function and the test results.*

| Function | Action (Test Cases) | Expected response | Actual response | Test results |
|---|---|---|---|---|
| User history | Search drug information after login and click on the "Hi, user--history" on the home page | The page showing the search history of the user appears | The page showing the search history of the user appears | PASS |
| | Click on the "Hi, user--history" on the home page | The page showing the search history of the user appears | The page showing the search history of the user appears | PASS |
| | Click on the "delete history" | No history in the list | No history in the list | PASS |

*Table 6 | Test cases based on the user history function and the test results.*

| Function | Action (Test Cases) | Expected response | Actual response | Test results |
|---|---|---|---|---|
| Contact us | Click on the "contact us" on the bottom of home page | A form appears | A form appears | PASS |
| | Fill in an email address with wrong format | An error message appears | Contact successfully | FAIL |

*Table 7 | Test cases based on the contact us function and the test results.*

## Discussion:

**Rationale behind functional design & Rational behind technical design**

The functional and technical design of our web application are irrelated and the design development can be divided into following five parts.

1. **Data obtaining & importing**

We obtained drug information through crawler, which depends on HttpUrlConnection to access web resources and *com.google.gson.Gso*n to parse JSON data. The parsed data was then formatted and stored in the local database through *JDBC* and *DBUtils* to connect with *MySQL* and using "insert into xxx" sql statements to save the data.

2. **Drug information display & Drug search**

Since we adopted three-tier architecture, we used servlets to call the service to use functions in DAO layer to access the database and obtain all the drug information or the drug query information requested by the client and display them through *JSP* files with *JSTL*.

3. **File upload & Analysis**

FileUpload is an upload component provided by apache's Commons components, through which we realized the file upload function. The content of the file will then be stored in the database through 'save' function in DAO. After that the gene symbols will be extracted through 'getSymbol' function, which will then be used to match corresponding drugs through 'findDrug' function in DAO.

4. **User**

To realize user relevant functions, we mainly used 5 functions in DAO, including 'findUsername', 'findPassword', 'saveUserHistory', 'findhistory', 'deletehistory'. They add, delete or query user's history or information from database.

5. **Page style**

To complete front-end development, Bootstrap, currently the most widely used CSS coding framework, is used for web style design. With Bootstrap, typography, navigation bar construction and overall page beautification were completed. In addition to the Bootstrap framework, we built separate CSS files to manage page styles for the beautification of user registration and login interfaces, search boxes, and contact page.

**Limitations & Potential development:**

Although our project can run successfully without any error, it still has some limitations. Here we propose the following four constraints, together with their feasible solutions.

1. **Practicability:**

To begin with, the web application cannot be accessed by other computers, which means it can only be shown to users who run the codes. Moreover, since the web application need to connect with the local database, users must import the data into his/her own database firstly, which heavily influence the practicability of the web application. Therefore, we can try to deploy it to cloud servers.

Secondly, it cannot support password recover function though we design to use emails to retrieve password. Therefore, we will add email sending together with captcha verification code to realize password recover function in future development.

2. **Performance:**

Firstly, we record all the user's query histories into one table called 'user_history'. Once the number of users or query histories is large, the user's history query function speed will significantly decrease, badly influencing the performance of the web application. Therefore, we can try to use cookies to save user's histories on client-side, which will not only reduce the memory space in database but also improve the performance of the project.

Secondly, we adopted three-tier architecture but did not use any architecture pattern, such as MVC. Since our project is not complex, it is not necessary to use Spring MVC at this stage. However, in the later project development, it may be better to use Spring MVC to rebuild the project, which can encapsulate the code and lead to low maintenance cost as well as be conducive to the division of labor in development.

**3. Convenience & Accuracy**

There are several function limitations in our web application. Firstly, it cannot provide advanced search function of drug query since we match full string at this stage, which is not convenient to use. Therefore, we will try to use regular expression matching or add more search qualifiers.

Secondly, it cannot provide the latest drug information since we obtained the data from PhramGKB in advance and then stored it into local database, which will influence the data accuracy. Therefore, we will update the database regularly and use websocket to realize the real-time refresh of the front-end pages when database is updated.

**4. Unit testing**

The unit testing was not carried out following each step of the code development, therefore it did not cover all the logics of the system and just focuses on some methods and functions in the code. It is better to carry out unit testing following the completion of each unit of the code, which will be beneficial for the whole system development process.

## REFERENCE:

Code URL:
https://github.com/YanranQiu/DST2_2019-20_PrecisionMedicineProject/blob/master/DST_real_final_v10.zip

Slides about requirement analysis, software design and tests are also uploaded to GitHub.
https://github.com/YanranQiu/DST2_2019-20_PrecisionMedicineProject/tree/master/presentation_ppt

ADETUNJI, M., LAMONT, S., ABASHT, B. & SCHMIDT, C. (2019). Variant analysis pipeline for accurate detection of genomic variants from transcriptome sequencing data. PLOS ONE, 14(9), p.e0216838.

Fowler, Martin. 2002 "Patterns of Enterprise Application Architecture". Addison Wesley.

JetBrains. 2020. Intellij IDEA: The Java IDE For Professional Developers By Jetbrains. [online] Available at: <https://www.jetbrains.com/idea/>

McLaren W, Gil L, Hunt SE, Riat HS, Ritchie GR, Thormann A, Flicek P, Cunningham F. The Ensembl Variant Effect Predictor. *Genome Biology* Jun 6;17(1):122. (2016)

Usersnap.com. 2020. [online] Available at: <https://usersnap.com/blog/web-application-testing/>

Vaidya, N., 2020. Servlet And JSP Tutorial | How To Build Web Applications In Java? | Edureka. [online] Edureka. Available at: <https://www.edureka.co/blog/servlet-and-jsp-tutorial/#Introduction>