Gene Service Project

Github Repository

Please navigate to our Github Repository (<https://github.com/christophergalli/Team_Habsburg>) where you can clone our solution for usage.

To do so, create a new folder, open you shell and direct yourself in it, using the following command:

*cd C:\Users\user\GitHub\_GeneService*

To clone the content, use the following command:

*git clone* [*https://github.com/christophergalli/Team\_Habsburg*](https://github.com/christophergalli/Team_Habsburg)

Working with Github, the following pipeline of commands ensures a smooth collaboration:

*git pull* to update your folder

*git add .* to add your new files

*git commit -m “type your message”* to commit them

*git push origin main*  to push the files into your repository

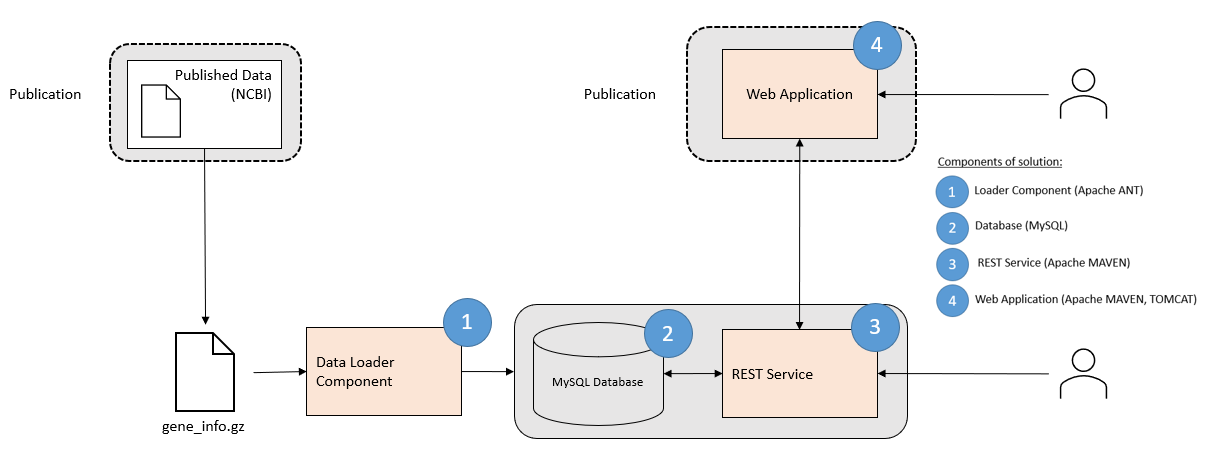
You have now successfully stored a copy of our work onto your local drive.

Data

During our project, we are working with a dataset provided by the NCBI. It is accessible for download using their ftp server:

<https://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz>

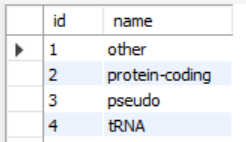
Download size of the file: ~4gb

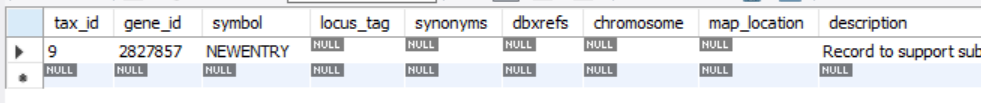
Architecture and overview

After downloading the gene\_info.gz file, we use a “Data Loader Component” that populates our MySQL database. The delivered REST interface allows direct queries of the database, using the 8080 port. For more convenient queries, consider using the delivered web application component. All the parts of our solutions will be explained in detail further below.

Database (MySQL)

In a first step, we will set up the database to be populated by the “Loader Component”. The database holds two tables “Genetype” and “Allgenes”.

Genetype Allgenes



Setting up the two tables is done either in the shell or the MySQL workbench using the following commands:

*create table* ***genetype****(*

*id int,*

*name varchar(128),*

*PRIMARY KEY(id)*

*);*

*create table* ***allgenes****(*

*tax\_id int,*

*gene\_id int,*

*symbol varchar(128),*

*locus\_tag varchar(128),*

*synonyms varchar(1024),*

*dbxrefs varchar(1024),*

*chromosome varchar(128),*

*map\_location varchar(128),*

*description varchar(2048),*

*type\_of\_gene int,*

*sfna varchar(128),*

*fnfna varchar(512),*

*nomenclature\_status varchar(1),*

*other\_designations LONGBLOB,*

*mod\_date TIMESTAMP,*

*feature\_type varchar(512),*

*PRIMARY KEY(gene\_id),*

*FOREIGN KEY(type\_of\_gene) REFERENCES genetype(id)*

*);*

As it is quite big, the “Allgenes” table is indexed (sorted), allowing faster querying. Indexing the database can be done using the shell or the MySQL workbench. The following two statements are performed:

*use geneinfo;*

*CREATE INDEX symIndex on allgenes(symbol);*

You can directly query the database and its tables, using queries such as:

*select \* from allgenes where gene\_id = 2827857;*

*select \* from genetype;*

Loader Component

As it is quite a big load, the tasks are performed in batches. The code is contained within the src folder “Loader\_v2”. It is used to transfer the data contained in the gene\_info.gz file to the database created above.

*Note: the loading process takes about 45mins.*

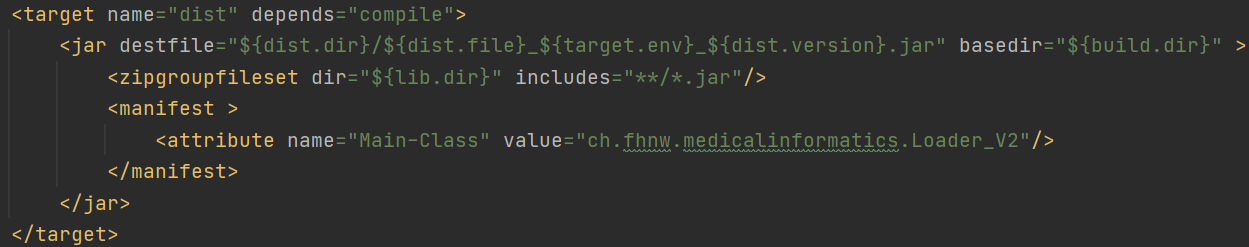
Build-process:

For the build-process of the loader component, Apache Ant (build.xml) is used. The build-file can be executed using IntellJ or the shell. Using the shell, the following information and commands are necessary:

*C:\apps\apache-ant-1.10.9\bin* location of your ant-environment

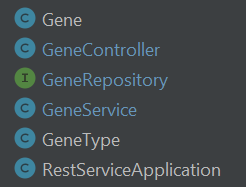
*cd C:\Users\user\GitHub\_GeneService\genedataloader C:\apps\apache-ant-1.10.9\bin\ant dist*

file location, environment, statement

The statement “ant dist” will execute the target specified in the build.xml file 🡪

This target has dependencies (compile) and thus executes all statements within the file.

REST Service

The core of the REST service is found in the main folder of the file. It consists of:

Gene Mapping to the “Allgenes” table in the database.

GeneController Request possibilities (byID, bySymbol, byDescription)

GeneRepository Queries for requests of the GeneController

GeneService Do not know

GeneType Access to the “Genetype” table in the database.

RestServiceApplication Set up of the SpringBootApplication.

Build-process:

For the build-process of the REST service, Apache Maven (pom.xml) is used. The build-file can be executed using IntellJ or the shell. Using the shell, the following information and commands are necessary:

*C:\apps\apache-maven-3.6.3\bin* location of your maven-environment

*cd C:\Users\user\GitHub\_GeneService\generestservice C:\apps\apache-maven-3.6.3\bin\*

*mvn clean package -DskipTests*

file location, environment, statement

*E.g.: C:\Users\urska\GitHub\_GeneProject\Team\_Habsburg\generestservice>C:\apps\apache-maven-3.6.3\bin\mvn clean package -DskipTests*

The statement *mvn clean package -DskipTests* will execute the build process (the -*DskipTests* allows to skip the tests that would normally be performed during the build process).

From the build process, you will use the .JAR file. A JAR (Java ARchive) is a package file format typically used to aggregate many Java class files and associated metadata and resources

Starting the service

Using the command line, you can start the service with the following commands:

*cd C:\ Users\user\GitHub\_GeneService\generestservice*

*java -jar target/JAR\_FILE.jar*

*E.g.: java -jar target/generestservice-0.0.1-snapshot.jar*

If you are using IntelliJ, you can also start the service directly by right clicking on the RestServiceApplication.

Port configuration

The default port for the server component is 8080. We will leave this as is. For the web application further below, we will change the port to 8090.

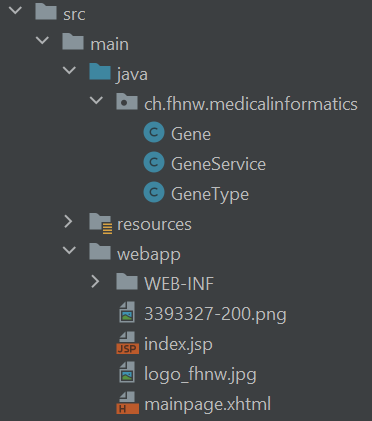
Queries:

You can directly query the REST service on its 8080 port using queries such as:

[*http://localhost:8080/geneservice/byID?ID=NDAI0A08340*](http://localhost:8080/geneservice/byID?ID=NDAI0A08340)

[*http://localhost:8080/geneservice/byid?id=11494045*](http://localhost:8080/geneservice/byid?id=11494045)

[*http://localhost:8080/geneservice/bysymbol?symbol=NDAI0A08340*](http://localhost:8080/geneservice/bysymbol?symbol=NDAI0A08340)

Web Application

The web application holds the different java classes needed for the queries. Furthermore, the webapp itself is defined here:

Build-process:

For the build-process of the web application, Apache Maven (pom.xml) is used. The build-file can be executed using IntellJ or the shell. Using the shell, the following information and commands are necessary:

*C:\apps\apache-maven-3.6.3\bin* location of your maven-environment

*cd C:\ Users\user\GitHub\_GeneService\genewebapp C:\apps\apache-maven-3.6.3\bin\*

*mvn clean package*

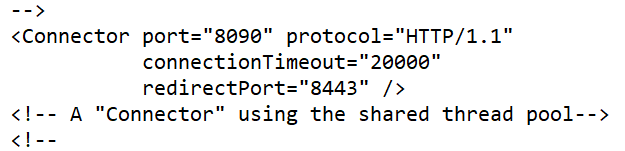
file location, environment, statement

*E.g.: cd C:\Users\urska\GitHub\_GeneProject\Team\_Habsburg\genewebapp C:\apps\apache-maven-3.6.3\bin\mvn clean package*

The statement *mvn clean package -DskipTests* will execute the build process.

From the build process, you will use the .WAR file. A WAR file (Web Application Resource or Web application Archive) is a file used to distribute a collection of JAR-files, JavaServer Pages, Java Servlets, Java classes, XML files, tag libraries, static web pages (HTML and related files) and other resources that together constitute a web application.

Port configuration

The default port for Apache Tomcat is 8080. As this port is already occupied, we need to change it. Within the config file, the server.xml file can be changed:

*C:\apps\apache-tomcat-9.0.45\conf*

Starting Apache Tomcat

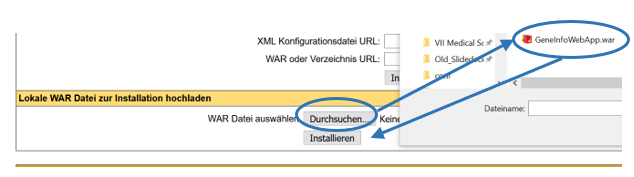
The server can be started by using the command line:

*cd C:\apps\apache-tomcat-9.0.45\bin*

*startup*

It can then be accessed on its port: <http://localhost:8090/>. Use the *shutdown* command to end the session.

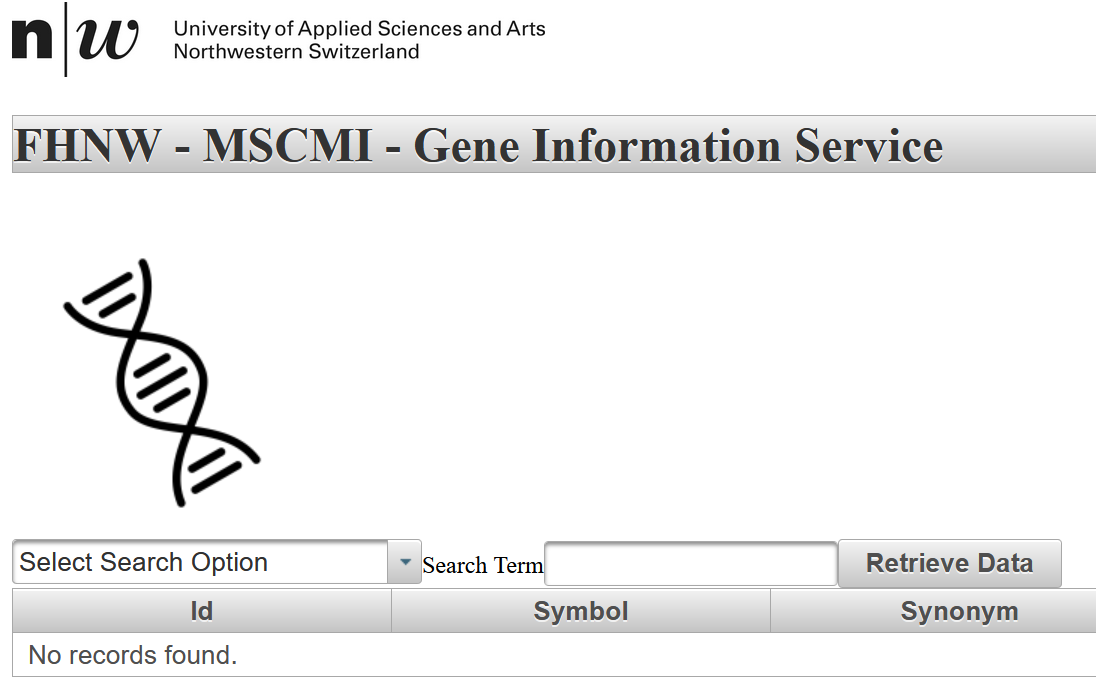
Add the .war file

**Within the “Manager App” of the Tomcat server, add the .war file built by the Maven command above:

*Manager App*

*Tomcat Webanwendungs-Manager*

The GeneInfoWebApp will then be available:



<http://localhost:8090/GeneInfoWebApp/>

Running all components

Starting the MySQL database

*use geneinfo;*

Starting the REST service

*cd C:\ Users\user\GitHub\_GeneService\generestservice*

*java -jar target/GeneRestService-0.0.1-SNAPSHOT.jar*

Starting Apache Tomcat

The server can be started by using the command line:

*cd C:\apps\apache-tomcat-9.0.45\bin*

*startup*

Additional Features

Logs

|  |  |
| --- | --- |
| Level | Description |
| OFF | Highest possible rank (intended to turn off logging) |
| FATAL | Severe errors causing termination (ahead of plan) |
| ERROR | Runtime errors and unexpected conditions |
| WARN | Warning signs displayed on a console (extension necessary) |
| INFO | Interesting events occurring during runtime |
| DEBUG | Information about how the system is running |
| TRACE | Most detailed information |

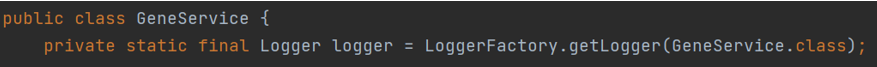
Apache Log4j is a Java-based logging utility. There are different levels of logs that can be displayed:

The gene project produces logs within:

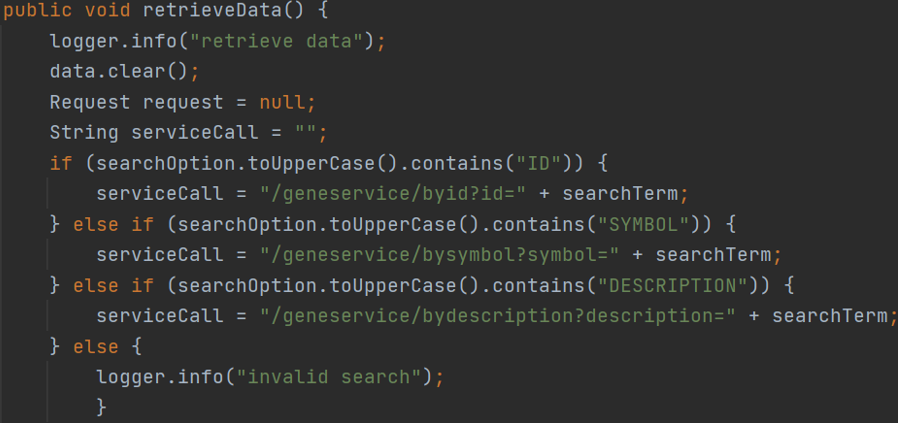
* REST Service
* Web Application

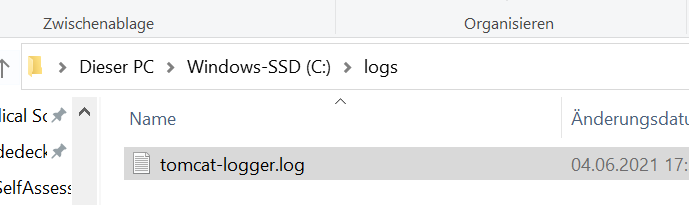
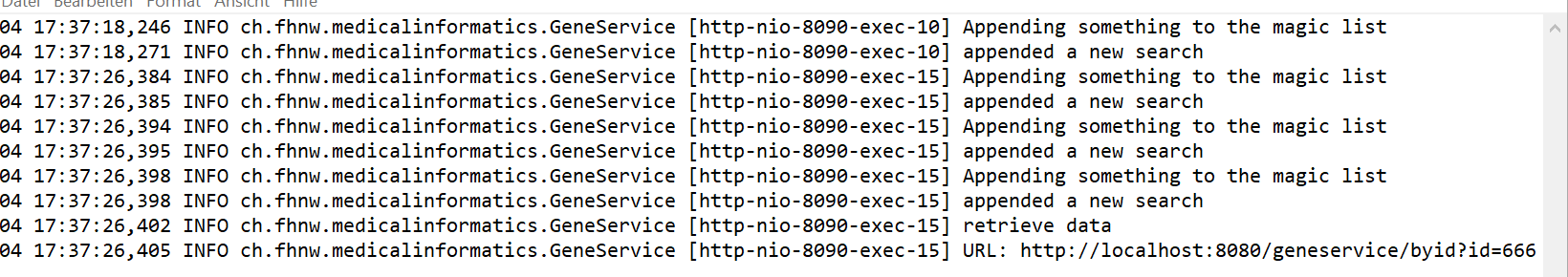
Set Up

Step 1: Import needed packages in webapp

Step 2: Instantiate in class

Step 3: Use within method



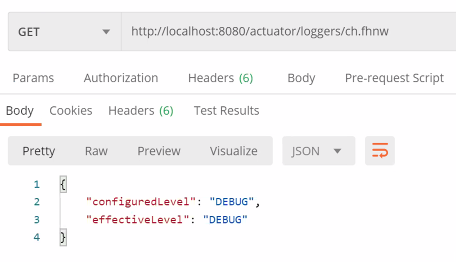
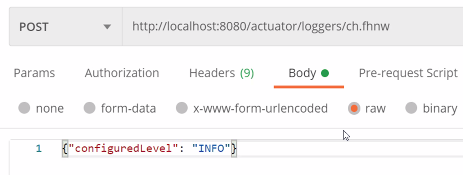
Step 4: Collect and review within file

*Note: If you are not a windows user, please modify the location of the log file as follows:*

* *Logback.xml (within the resource folder)*
* *Line 4: value=”path”*

Change

Within the REST Service, it is possible to change the level of the logs during runtime. This is particularly important to receive information in case of problems from clients. It is possible to implement such a feature for the Web Application, but the spring boot framework is easier. The general procedure is as follows:

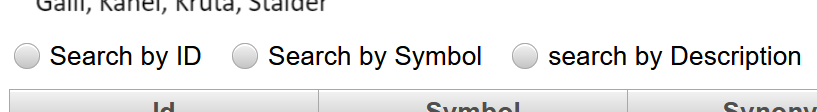
Step 1: GET Request (Information) Step 2: POST Command (Change) Step 3: Get Request (Confirm)

Current level

Change level

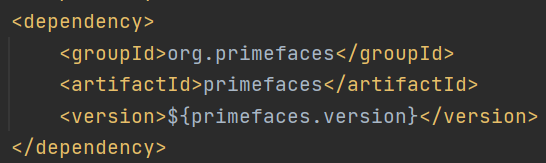
New level

Layout

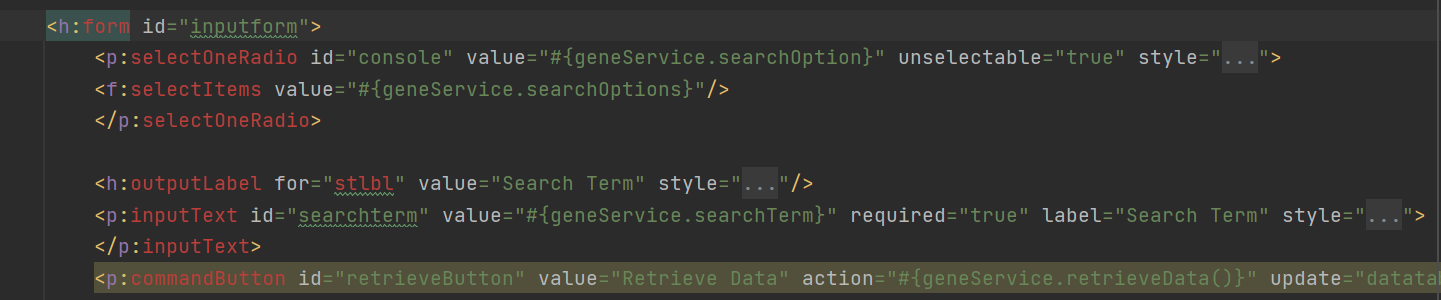
Changes in the layout of the webapp are:

Radiobutton

Step 1: Add dependencies to the pom.xml file



Step 2: Add Radiobutton the mainpage.xhtml file



Logo

Step 1: Replace the picture in the mainpage.xhtml

