Technical documentation

Genomizer

Version 3.0

Publication date: 2015/11/10

Contents

Preface

This documentation describes the *Genomizer* project conducted during the spring of 2015. The project is a part of the course in software engineering named *Programvaruteknik(5DV151)* given by *Umeå University*. The course is given to a mix of students. Some studying for a master and some for a bachelor degree in computer science. Hence different kind of knowledge exist within the student groups.

The documentation has two purposes. **1**) To give the possibility for tutors to assess the project and grade the students on their work. **2**) To describe the project and all parts of the developed system. The target audience of the system description are three subgroups: end users, system administrators and developers.

End users are epigenetic researchers. Developers are students of the course (current and future). System administrators are both students and personnel responsible for maintenance of the system.

The origin of the project is a perceived need from the epigenetic research department at *Umeå University*. The wish is to make a more efficient pipeline for the computational parts of their research. An automated pipeline where knowledge of the parts involved is kept to a minimum. The automated pipeline would result in less time spent on data entry and more time available for analysing data or conducting experiments.

Changes last version

Only Chapter 1 is left completly untouched.

Acknowledgments

- Jonas Andersson: Great technical support and workflow support.
- Jonny Pettersson: Great support on the group dynamics.
- Jan-Erik Moström: Great feedback on documentation.

Developers spring 2015

Data Storage	Processing	Business Logic
Nils Gustafsson	Adam Dahlgren Lindström	Johannes Ekman
Albin Råstander	Carl-Evert Kangas	Alexander Frisk
Jimmy Sihlberg	Emil Nylind	Tim Hedberg
Martin Larsson	Mikhail Glushenkov	Mikael Johansson
Erik Samuelsson	Saimon Marouki	Mikael Karlsson
Fredrik Uddgren		Stefan Lindström
		Robin Ramquist

Desktop	Mobile Applications	Website
Viktor Bengtsson	Erik Berggren	Ludwig Andersson
Maximilian Bågling	Jesper Bilander	Niklas Fires
Christoper Fladevad	Victor Bylin	Andreas Günzel
Jonas Hedin	Pål Forsberg	Pascal Hansson
Petter Johansson	Petter Nilsson	Patrik Hörnqvist
Marcus Lööw	Mattias Scherer	Anna Jonsson
Oscar Ottander		Björn Pers

Editorial Staff: Mikael Johansson, Erik Samuelsson, Marcus Lööw, Jesper Bilander, Patrik Hörnqvist.

2015/11/10Mikael Johansson

1 Introduction

Genomizer is a system for storing and analysing *DNA*-sequence data. It was designed for researchers in the field of epigenetics, who are interested in where on a *DNA* string certain proteins binds. In order to get this information, experiments are conducted and *raw* data files collected. These data files are then converted, in a series of steps, to files suitable for analysis. These files are hence refered to as *profile* data. *Genomizer* allows the researchers to upload *raw* files to a server and automate the generation of analysis data as well as store the generated analysis data in a database for later access.

The documentation contains three main parts. Introduction chapters that explain the goal of the project as well as a non-technical description of the project implementation. The development part of the document where the current implementation of each part of the project is explained how they look and work as well as an attempt to explain why certain design choices where done. Then finally there is a big collection of appendicies that goes deeper in their explanation of certain details of the implementation as well as maintenance guides.

2 Target group and needs

The *Genomizer* system was designed with a specific target group in mind: Epigenitic researchers. This chapter will explain the needs of these users, the problems they faced before this system was provided and the requirements that were collected and taken into account during the project.

2.1 Target group

The target group for the *Genomizer* system is *Epigenetic Cooperation Norrland (EpiCoN)*, a diverse group of researchers at *Umeå University* made up of many different nationalities. Their main communication language is English.

EpiCon are involved in the research of how proteins bind to *DNA* strings and its effects. Experiments are carried out which yield large amounts of raw data. This information, combined with knowledge about the location of genes within a given genome, enable the researchers to gain valuable information about which proteins are active in enabling and disabling genes. These results are important in the study of how cells ”remember” which genes should be enabled after cell division.

Previous to the *Genomizer* project the raw data files retrieved from experiments were manually processed by the researchers using inefficient *Perl* scripts. This process also involved using *Bowtie*[1], a program used to unscramble the *DNA* data, and *LiftOver*[2] which is used to adjust results to conform to different *genome releases*.

The researchers at *EpiCoN* have varying computer skills. While they all have basic computer knowledge, not all are familiar with more advanced computing tasks such as running scripts at command line level. As such, some researchers have become dependent on others to process the raw data. At *EpiCon* the researcher that has the knowledge to use all the scripts and software performs many of these time consuming tasks for other researchers.

From time to time students of molecular biology are interested in working with the data, however their access is limited to viewing and analysing the data.

2.2 Client needs

The researchers at *EpiCoN* need a system to structure the large amount of genetic data they use daily. The requirements, as described below, were collected and handled as a number of *user stories*, each of which describe a desired function from the end users perspective. A complete list of the *user stories* are presented in Appendix C on page 112. A overview of the requested system may be seen below in Figure 2.1. Where orange colored nodes are must have features while gray nodes are visions of the clients that may be implemented if time allows for it.

There are three main data types used in the research and that the system should handle: *raw*, *profile* and *region* data. *Raw* data is the raw output from an experiment and cannot be analysed directly. It is first processed to so called *profile* data. *Profile* data describes the amount of reads found for every base–pair in an organism’s genome. *Region* data is further processed *profile* data consisting of the regions where every base–pair’s read strength is above a given threshold and fault tolerance. The region gets a value based on the average of the base-pair reads for the given region.

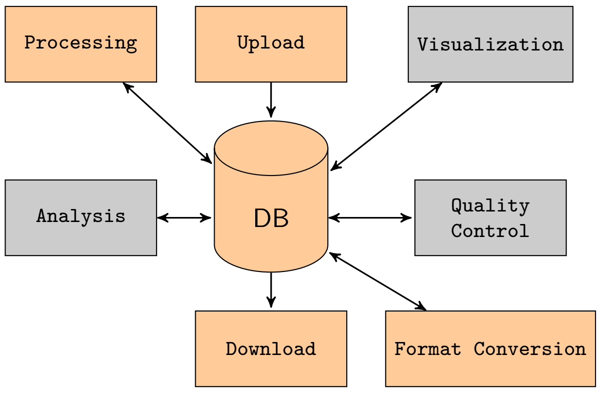


Figure 2.1: Overview of targeted system

2.2.1 Upload & Download

When conducting experiments the researchers generate *raw* data that generates what they call *Raw*-files. These files along with profile data, region data and genome release data may be added and related to an experiment. The requested functionality is to be able to upload these files to the database from multiple sources. The sources may be directly from an experiment conducted by the researchers or from official publications.

When results are published in scientific articles the *raw* data from the experiments are often also provided. One location where these *raw* data files can be published is the *GEO* (*Gene Expression Omnibus*) database. A desire to be able to initialize an upload to *Genomizer* with the source of the upload beeing *GEO*.

2.2.2 Database

The `Database` module requested has the purpose to archive experiment data in a way of easy access. To allow for this the experiments and files associated with them needs to have information vital for good readability. This is solved with the help of annotations. The researchers must add annotations to files related to an experiment. This data is the foundation for further research and so must be stored securely. To ensure security the client requested a system for authorization that protects the data from outside tampering. To protect against hardware failure there exists a request for a backup system.

2.2.3 Processing

The unordered *raw* data gained from an experiment requires processing in order to be analysed. The researchers have written a number of scripts and, when combined with the *BowTie* algorithm, generate *profile* data. In this format the *DNA* pieces are ordered and mapped to the *DNA* string. It is important that the system automates this process so that all researchers can easily process the large *raw* files.

As new discoveries are made in the area, new standards for the order of the base pairs in a *DNA* string are set. This results in a new *Genome Release* for a specific species. These are obtained as a set of files specifying this order and are used in the processing of *raw* data. *Genomizer* must support the uploading of new sets of *genome release* files to be used in processing otherwise the system will very quickly become outdated.

It would also be an advantage if the system could carry out further processing from *profile* to *region* files.

After processing, the resulting data files should be annotated and saved in the database alongside their parent files. It is important that the parent files remain traceable and that the parameters used in processing are saved so that the process can be repeated and confirmed.

2.2.4 Format Conversion

Genomizer should also provide a way to convert *profile* data files between different genome releases. This involves the ability to upload new *Chain Files* which enable conversion using *LiftOver* and the embedding of this program. The *LiftOver* program compares the differences between two genome releases and converts it to one update genome release.

It is not uncommon to discover errors in a new release after publication, thus it is important to store files generated using older genome releases, even though newer releases has been published to allow for *LiftOver* conversion.

3 Service description

This chapter will present an overview of the services that the *Genomizer* system currently provides.

3.1 Usage

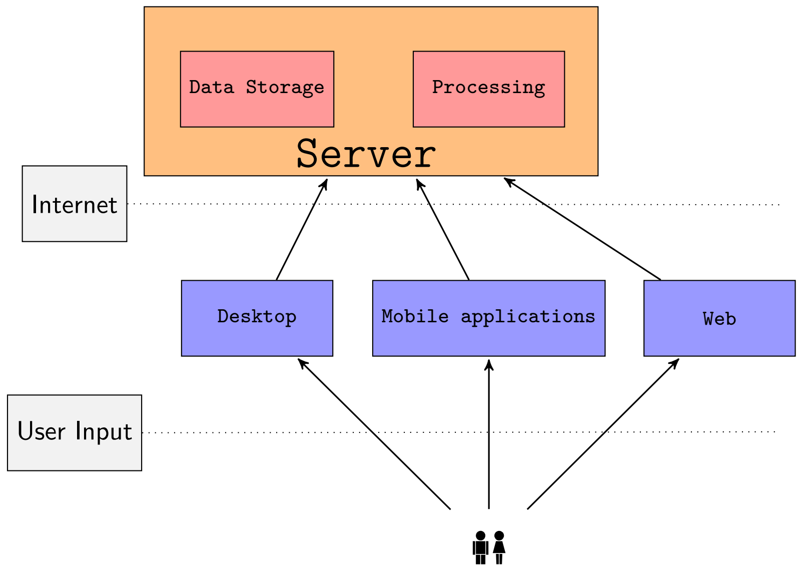


Figure 3.1: Communication diagram of the product

In order to give the users flexibility when using the service there are clients for many different platforms (Windows, Linux, OSX, Web, Mobile devices). When a user chooses a given task, for example start *raw* to *profile* processing. The task i sent via Internet to the server as shown in Figure 3.1. The server then handles the request and return a response back to the user.

3.2 User Input

The user input in the *Genomizer* system may be done with four different clients: a desktop client, a web client, an *Android* client and a *iOS* client. The last two clients are collected under mobile application since they offer the same functionality.

3.3 Desktop

The desktop client is the main client for the *Genomizer* system. It offers the following implemented functionality.

- Login and logout.
- Searching for experiments and different files.

- Create new experiments.
- Modify existing experiments.
- Upload files to experiments.
- Download files from experiments.
- Process files from raw to profile.
- Delete files and experiments from the database.
- Add annotations to experiments.
- Remove and modify annotations.
- Search annotations by name.
- Upload, remove and rename genome release files.

3.4 Web

The web client runs, without any installation, in a web browser and mimics most of the behaviour and functionality of the desktop client. However, adding and removing users is not possible in the web client. Neither are there different user rights that give users different access and permissions. There is only one right for all users.

3.5 Mobile application

Due to the limited storage available on mobile devices it is not appropriate to enable uploading and downloading of files, however the mobile applications enable the searching of files in the database and the scheduling of processing procedures for the conversion of *raw* to *profile* data.

3.6 Server

The purpose server is to take care of the organizations of files and experiments as seen in the top part of Figure 3.1 as the part *Data storage*. The server also serves as a processing tool of the files added to the *Data storage*.

3.6.1 Data storage

The main purpose of the *Genomizer* system is to centralize all data. To enable this a user can annotate and upload data to the server using both desktop and web based clients. Uploaded files are organized in *Experiments* which are the representation of actual experiments in a laboratory. An experiment acts as the logical container for files that are related to each other - it can hold many files of different types (*raw*, *profile* and *region*). These files can either be uploaded directly by the user or generated from processing by the server.

Advanced database searches can be performed on the annotations to find previously uploaded data. When the required data is found the user can choose to download the files or request that they be processed on the server.

3.6.1.1 Annotations

Annotations is a way for the researchers to keep track of what an experiment consists of as well as files associated with an experiment. *Genomizer* has two kinds of annotations. There is *multiple-choice* annotations which have a defined name and choices. An example is the annotation named *species* that have the choices human, fly and rat. Then there is *free text* annotations where the user may enter what they want.

Dynamic annotations must also be managed in order to keep the system clean and up to date. *Genomizer* therefore provides full editing options for existing annotations if the user have the credentials. This includes the editing of *mulitple-choice* annotation choices and the removal of unused annotations.

Example 1 *If the user has an experiment that was conducted in zero gravity and the database does not have the annotation field "Zero Gravity" the user can add this as a new annotation. In this case a Drop Down annotation type may be appropriate, with the simple choices "yes" or "no". Of course it is also possible to leave the annotation type as Free Text which enables users to write freely the value of the annotation.*

3.6.2 Processing

Users can request that a *raw* file set be processed to *profile* files. This procedure is carried out by the server to avoid heavy workload and the requirements of certain programs on the clients side. The processing carried out between *raw* data and *profile* data involves a number of different steps. The user can choose which steps are carried out and the various parameters used.

3.6.3 Profile data conversion

The server can be requested to convert between different formats of profile data. The conversions currently handled by *Genomizer* are *gff*, *sgr*, *wig* and *bed*. These formats are specified by the Genome Bioinformatics Group at UC Santa Cruz[7]. Since different formats carry different amount of information or meta data, conversions from each file format to every other is not possible.

Part I

Development

4 Architectual design

To get an understanding on how the system is designed as a whole, this chapter will try to explain the architecture of the system on a more broad level and design choices that is to expensive to change at the current point in the project.

4.1 System overview

The *Genomizer* is a server-client architecture. It consists of four different clients: a web client, a desktop client and a two kinds of mobile clients. The mobile clients are implemented in *Android* and *iOS*. The server side consists of three components, a web server that acts as a proxy, a *Java* server that connects to the final component a *postgresql* database.

The web server is a *Apache* system in the current implementation.

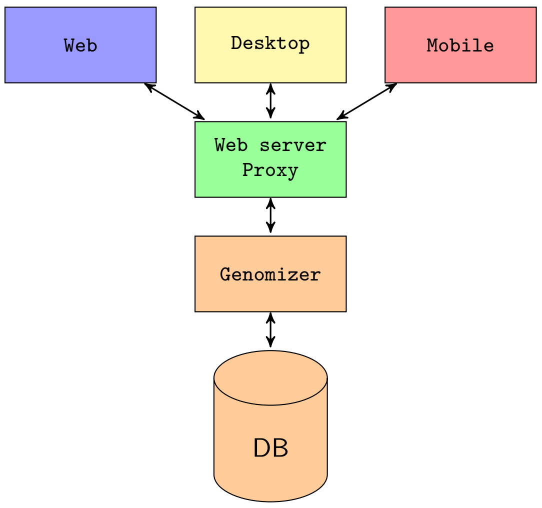


Figure 4.1: Overview of the system architecture

The connection to the server from any client goes through the web server which acts as a proxy as well. This to insure a *SSL* connection between the web server and the clients and a regular connection between the server and web server. The proxy protects the web client from *XSS* (Cross site scripting).

All of the clients use the *RESTful* protocol where the message body consists of a `JSON` object. These messages are requests that gets passed through the web server to the *Genomizer* server. The *Genomizer* server communicates with the database to perform the request. Then it generates a response following the *RESTful* protocol. The *RESTful* protocol is used to promise that the server will be state-less. State-less meaning that a request from a client can not lock the server for other clients requests.

The different requests that can be sent to the server are defined in Appendix I on page 132 that goes into deeper detail about the API.

The architecture of the system can be seen in Figure 4.1 where the clients are the colors blue, yellow, and red. Web server is green and the *Genomizer* server and its database are colored orange.

4.1.1 Genomizer clients

All the clients that are part of the *Genomizer* service are implemented using the architectural design pattern MVC (`Model View Controller`). This pattern is based around keeping the logic and data separate from its representation and user interface. The basic idea is shown in figure Figure 4.2.

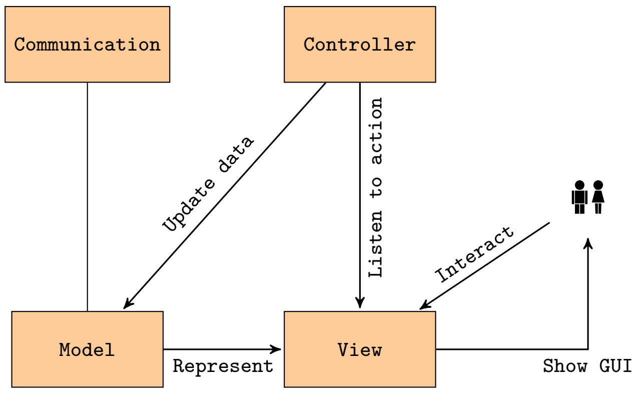


Figure 4.2: Basic MVC pattern.

The view will contain lists of experiments and annotations, data regarding process status, and other information. It will also present the user with windows and buttons that present and allow data to be manipulated. When a user interacts with the graphical interface the action will be handled by the controller using listeners. This in turn modifies the model; representing the data and information, as well as the operations available on it.

For this project a communication part is also necessary; often as a part of the model. As mentioned in the overview section above, the communication is performed via a *RESTful* protocol using `JSON` requests and responses. `JSON` is used because it is easy to parse and is directly compatible with the standard frameworks used in the web client.

The advantage of this approach lies in the separation of the interface- and logic-code; allowing one to be more easily replaced or modified. In our case the communication data and interface can be kept the same, while the client representation of it can be changed freely.

4.1.2 Genomizer server

The server is devided up in three main components, the front end, the data storage and the processing.

The front end consist of a handler for the *API* requests mentioned above. The handler interpret incoming request and passes them through to either processing or data storage depending request type.

Data storage makes up the back-end of the server. Its purpose is to communicate with the file system and the database to return the correct information that was requested by the front-end.

The processing component of the server has the purpose of interpreting the *raw* files and convert them to *profile* data. This conversion is very heavy on the hardware of the server, thus the processing needs to be scheduled to times when user activity is low.

Figure 4.3 shows a simple flow diagram which describes how the client and server communicates. The particular example shows the data flow when the client process a file. In the figure the front-end is the communication node while data storage makes up the database node and processing the final process node.

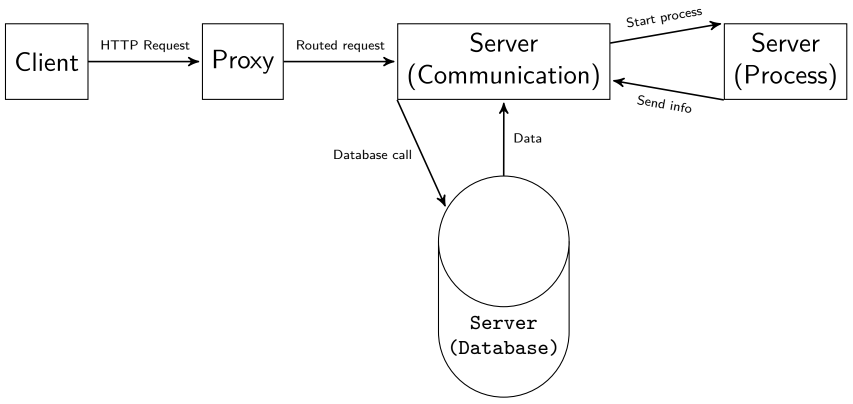


Figure 4.3: The flow of a request through the server.

Every request the client does creates a non persistent connection to the server. When the server receives a request it checks which kind of request it is and routes it to the communication part of the server.

When the request is routed to communication a specific command is created. The command is an object which consists of information from the *RESTful*-header and `JSON` body sent from the client. The command is then parsed and sent to different parts of the server, usually the database first, which returns information from a *SQL query*. Depending on the requests this information can later be used, for example, to process a file or be sent back to the clients directly.

The clients are always going to receive a response code after each request, but in some cases the respond also contains a `JSON` body with information which can be shown to the user. This is the case for requests like `getAnnotations`. The response can also contain error messages, describing what went wrong when executing the command.

After a client receives the response the connection with the server is closed and a new is opened with the next request.

5 Interaction design

This chapter goes into detail on how the graphical and interactive parts of the clients are designed. It starts with a general view of the interaction design and then divides into chapters based on the different clients.

5.1 General view

Since the Genomizer application is first and foremost about handling important data, it is crucial to allow the user to be in control but still to protect and preserve the data that is already in the system from non-authorized users. That is why Genomizer uses a log in screen to reassure that only authorized personnel can access the server data.

The workflow of Genomizer is intended to be as natural as possible for the users and to be easily integrated into their daily work routine. Furthermore, simplicity is favored, the clean and consistent design that stretches over platforms facilitating the tasks without adding the distraction of any unneccessary features.

5.2 Desktop client

Screen clients use a tab based navigation between views, these tabs are shown at the top of the user interface. The common views in the current system are search, upload and process.

Search results are displayed in a table, experiments can be expanded to reveal the files contained in the experiment. The files in an experiment are grouped by types where each type consists of a row in the table that may be expanded to reveal the files of that type.

The upload view consists of experiment groups. Each experiment group contains a set of input fields for annotation and a list of files added to this experiment. The user may create new experiments in this view or add files to an existing experiment, multiple files may be added to multiple experiments simultaneously.

The base for the process view contains a set of input fields for the parameters that are to be used when processing a file.

5.2.1 *Windows/OS X/Linux* application

The first thing a user will see when starting the desktop client is the login window (see Figure 5.1). The window prompts the user for user name, password, and a server IP to connect to. If the correct credentials are entered, the user will be logged in and taken to the next screen. If the user enters an invalid password, user name, or server, an appropriate error message will be displayed as seen in Figure 5.2. This feedback informs that user the login was unsuccessful.

After the user has been successfully logged in, the main window will appear (see Figure 5.3). The main window is built with tabs to simplify work by letting the user easily switch between different views for different work tasks. Each tab is described by appropriate name and contains related functionality. The main window also has a log out button. This button is of little importance, and is therefore located in the upper right corner.

At the bottom of the main window a status panel is located. The status panel gives feedback from different tasks executed by the user. When a task is executed successfully, the color of the status bar will turn green, and display a message. In case of an unsuccessful execution, the status bar will turn red, to indicate that something went wrong.

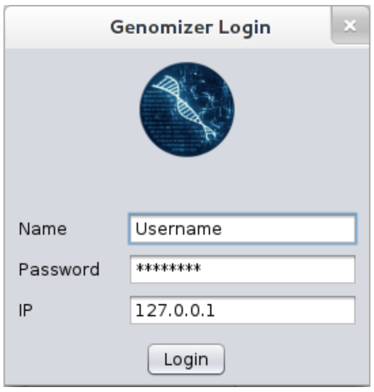
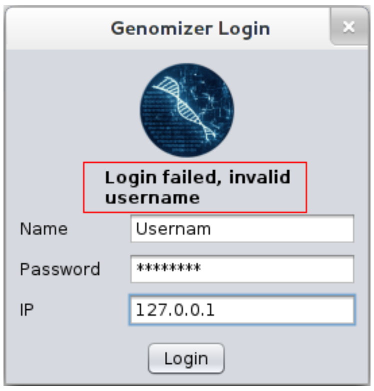


Figure 5.1: The login window.



Login failed, invalid username

Figure 5.2: The login window with an error message after a failed login.

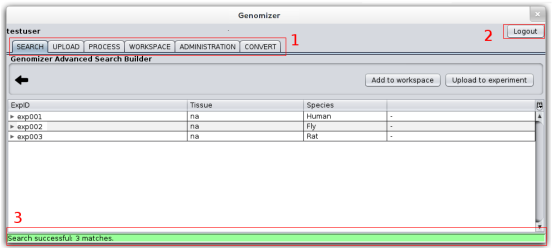
From the search view (see Figure 5.4) the user can build search queries and look up existing experiments. The search view has been designed to have a similar layout and interaction as the advanced search tool on the site `http://www.ncbi.nlm.nih.gov/pubmed/advanced`. The researchers are familiar with this site, so they can recognize interaction elements from it when using the search function of the desktop client.

To search for experiments the user can click the magnifying glass. This icon is well-known and often associated with searching. The icon is located next to the search field, so that the user can easily understand that the search field and the icon are connected. The user can also press the enter key to perform a search, without letting go of the keyboard, making the interaction faster. Next to the magnifying glass is a button for emptying the search field. The button has an icon depicting a trash can – a well-known metaphor for removing or emptying things.

From the upload view (see Figure 5.5) the user can create new experiments and upload files to them. When creating a new experiment the user is forced to fill in some fields. These fields have been given a bold-texted label, to indicate that they are of more importance than the others. A text below the fields also states that bold fields are forced. Non-forced fields are labeled with non-bold text.

To inform the user that information is missing, constraints has been put on the buttons for creating the experiment. If any forced field has not been filled in, or no files have been added for upload, these buttons will be grayed-out and cannot be clicked.

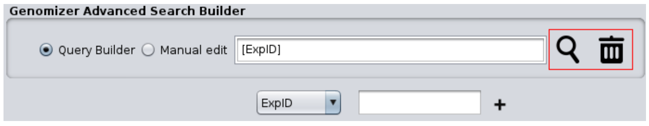
For each file added to the experiment there is a progress bar. This bar gives the user feedback



ExpID	Tissue	Species	
exp001	na	Human	-
exp002	na	Fly	-
exp003	na	Rat	-

Search successful: 3 matches.

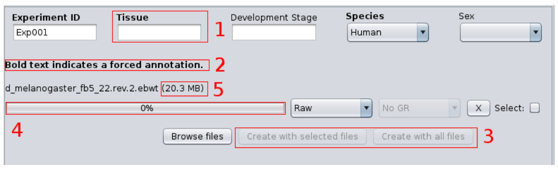
Figure 5.3: The Genomizer desktop client's main window. The window have tabs for different views (1), a log out button (2), and a status panel for feedback (3).



[● Query Builder] ○ Manual edit [ExpID]

Figure 5.4: The search view of the Genomizer desktop client with the icons for search and emptying the search field highlighted.

on upload process. Each file also has its size displayed after its name. This gives the user an idea of how time consuming the upload will be.



Experiment ID: Exp001
Tissue:
Development Stage:
Species: Human
Sex:

Bold text indicates a forced annotation.

d_melanogaster_fb5_22.rev.2.ebwt (20.3 MB)

0%

Select: ☐

Figure 5.5: The upload view of the Genomizer desktop client. An empty forced field (1), as stated by the bold-texted label (2), makes the upload buttons (3) grayed-out. The upload progression bar (4) and the size of the file (5) gives the user an idea of how time consuming the uploading process will be.

The workspace tab lets the user easily manage files and experiments. Files and experiments in the work space are listed the same way as search results in the search view, making the design consistent throughout the system. The workspace view has easy access to the download and process functions.

The administration view (see Figure 5.6) is divided into two seperate views, one for managing annotations and the other one for managing genome releases. The division of the views makes the interface less cluttered and less confusing, and also increases the cohesion of the views. The user can easily switch between the views by clicking the the tabs on the left-hand side.

To improve the feedback when errors occur, an error dialog (see Figure 5.7) will be shown. The dialog explains what went wrong and why the error occurred. The user can find additional information about the error by clicking a button labled 'More info'. This information can be useful for system administrators, developers or support, but not for regular users (which is why it is initially hidden).

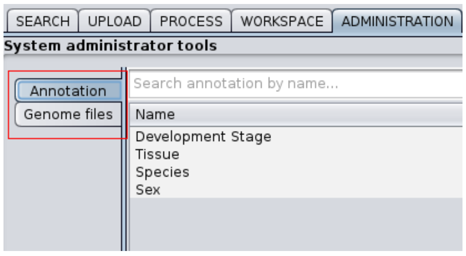
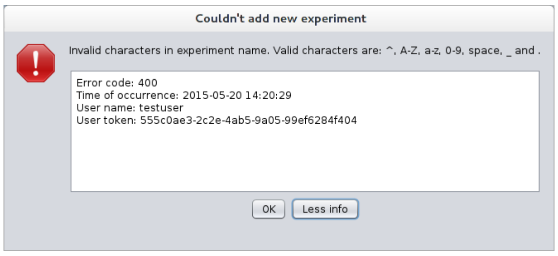


Figure 5.6: The administration view showing part of the view for managing annotation. The highlighted tabs to the left let's the user switch between views.


Couldn't add new experiment

Invalid characters in experiment name. Valid characters are: ^, A-Z, a-z, 0-9, space, _ and .

```
Error code: 400
Time of occurrence: 2015-05-20 14:20:29
User name: testuser
User token: 555c0ae3-2c2e-4ab5-9a05-99ef6284f404
```

Figure 5.7: An error dialog that explains that the user have entered invalid characters for an experiment name.

5.3 Web application

Generally, the design of the user interface for the web application is an integration of the principles previously described with core design elements of web and the twitter bootstrap element library.

5.3.1 Layout and Structure

The structure of the application is in most cases shallow, the navigational depth is usually two steps but sub views with modal views may result in a depth of 3. There are three types of views which are hierarchical in some way, main views contain sub views and modal views, sub views may contain modal views.

- **Navigation bar:** A navigation bar is a menu that contains an overview of the functionality in the form of, in this case, tabs that leads the user to other views and that is always visible for the user to allow easy navigation. The navigation bar for the application can be seen in Figure 5.8. Since the most important parts of the application is to be able to upload, process and convert files, these are each a natural part of the navigation bar (although conversion is not yet implemented and therefore not shown in the figure).
- **Main views:** A main view covers the entire page except for the navigation bar. The structure among main views is shallow and the user may freely navigate between all main views using the navigation bar. Typically a main view contains a toolbar and a

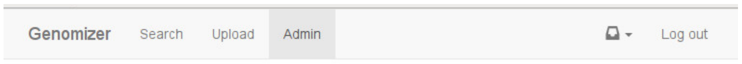
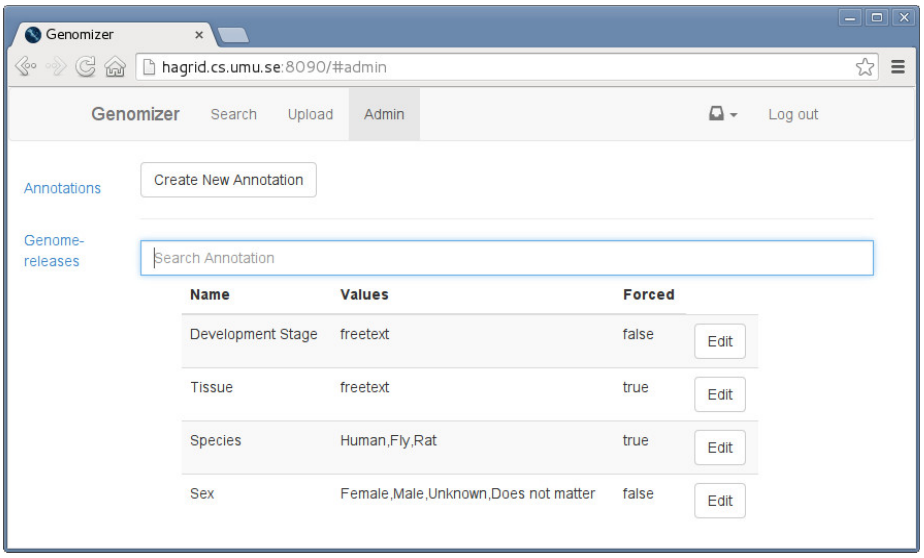


Figure 5.8: The web client navigation bar.



Name	Values	Forced	
Development Stage	freetext	false	Edit
Tissue	freetext	true	Edit
Species	Human,Fly,Rat	true	Edit
Sex	Female,Male,Unknown,Does not matter	false	Edit

Figure 5.9: The web client administrator main view.

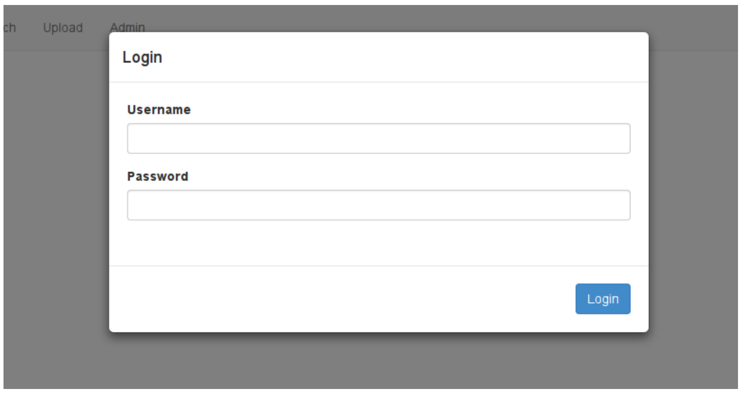


Figure 5.10: The login modal.

set of panels. Figure 5.9 shows the administration main view.

- **Sub views:** A sub view is a part of a main view. In the case of the administrator view seen in Figure 5.9, the main view has a vertical navigation bar on the left side used to navigate between sub views, sub views may not be directly navigated outside of its main view. The user may navigate to other main views from a sub view. Except for the sub navigation bar the sub view covers the entire main view, replacing its content, as does the annotation view in this case.
- **Modal views:** Modal views are opened on top of the current main view and are used for specialized operations. Modal views can be navigated to using buttons inside main views and sub views. Usually the user will be taken back to the previous view when the modal is closed but navigation in a sequence of modal views could be implemented in the future. An example of a modal view is the login view seen in Figure 5.10.
- **Panels:** Content that belongs together is grouped using so bootstrap panels. Main views and sub views should contain one or more panels.
- **Toolbars:** In main views and sub views we use a toolbars where operation controls available to the user are presented, for example, add and search experiment functionality in the upload view.

- **Popovers:** Elements that belong to a view but have no need to be visible at all times are shown in bootstrap popovers. Popovers that do not belong to a specific view may be placed in the navigation bar, that is, the main menu.

5.3.2 Colors

Grayscale colors are mostly used, black or dark gray is used for text, icons and borders while white or light gray is used for backgrounds. Colors of different hues are used to distinguish elements from each other and to highlight important elements. Colors with high saturation are reserved for smaller elements while colors with lower saturation can be used regardless of element size. Light gray of varying brightness may also be used to highlight or distinguish elements.

5.3.3 Icons

Buttons that perform actions should always contain an icon as well as text so that the experienced user may more quickly desired actions by identifying buttons at a glance instead of having to read the button text.

5.3.4 Batching

For operations performed on objects that there are multiples of e.g. experiments or files, let the user perform these operations on multiple objects at the same time in cases where it makes sense using checkboxes.

5.3.5 Processing

The interaction flow of the processing is adapted from the actual processing steps in order to help the researchers by increasing usability through providing a well-known but more optimized approach.

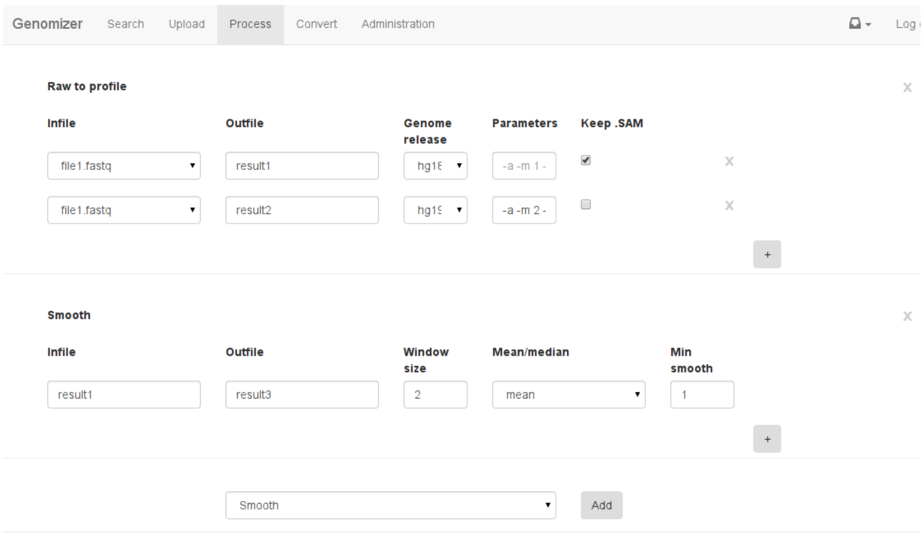
After having chosen an experiment for processing and entered the processing view, the user can choose the processing steps wanted and enter the correct files and parameters for each processing step as shown in Figure 5.11.

5.3.6 System administration

The admin page is built up by a number of components: the main view, the side bar, the create annotation view, the edit annotation view and the genome-release view. The first one is the main view which consists of a sidebar and an empty div-tag. The empty div-tag is then replaced with the annotation list view which has a Create new annotation button and a list of the available annotations on the database with an option to edit.

When the user clicks on for example Create New Annotation, the div tag in the main view is replaced with the create annotation view. The same goes for the Edit buttons on each annotation. This way we only have to render that specific div-tags current information and the sidebar is unaffected.

The design is made so that the user should be able to avoid mistakes. For example in the create annotation page the user is not able to create an annotation without filling in all the fields. Futher more the field for Items in drop-down list is disabled if the user don't choose Drop-down list as the annotation type.



Raw to profile

Infile	Outfile	Genome release	Parameters	Keep .SAM
file1.fastq	result1	hg18	-a -m 1 -	✓
file1.fastq	result2	hg19	-a -m 2 -	☐

Smooth

Infile	Outfile	Window size	Mean/median	Min smooth
result1	result3	2	mean	1

Smooth

Figure 5.11: Files selected for upload.

In the Edit annotation view the same principles apply, but also there is a Delete Annotation button on this page which will delete the entire annotation from the database.For that reason we decided to ask if the user is sure of this action and of course made the button red.

The back buttons on the different views work as one would expect and the sidebar option Annotations takes the user back to the main adminview.

The sidebar item ”Genome-releases” takes the administrator to the page for adding and editing genome-releases. This page have the same look and feel as the previous. The delete buttons are red and will prompt a confirmation-popup.

The ”Select files to upload” will as expected open the file explorer and the user chooses files according to normal operativesystem standards, then the ”Upload” button will prompt the user for information about the files such as species and genomeversion before uploading.

5.4 Android

The *Genomizer Android* application was designed to allow for a quick search of the database while on the move. It also makes it possible to start processes in advance so that the data is ready when further work and analysis is to be done. The application will also provide a way to continuously view the status of active processes on the server.

The application was designed in close collaboration with the *iOS* application in order to provide a consistent experience on both plattforms. We did, however, find it necessary to take into consideration some of the *Android* specific design paradigms which distinguish *Android* applications from other smart phone platforms. In this section the layout and design decisions will be described.

5.4.1 Login view

There are two textfields available for the user to type username and password and a button to click when user is ready to log in. This is a popular layout for many login screens and thus a design many users are familiar with.

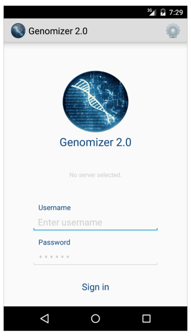


Figure 5.12: Login view

5.4.2 Search view

The design illustrated in Figure 5.13 show the search view, which is also the view the user is presented with upon successful login. The search annotations are displayed in a list and it is easy to learn how to search. Scroll bars are used for multiple options and textfields are used for free text. At the bottom of the view there is a button to press in order to start the search.

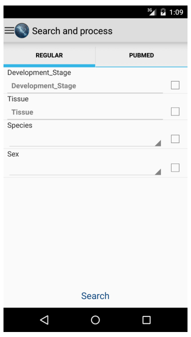
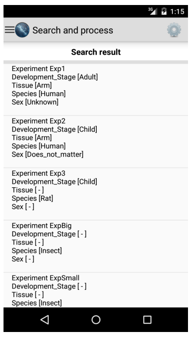


Figure 5.13: Search view

5.4.3 Search result view

The design illustrated in Figure 5.14 show the search result view. The result is shown in a list, sorted by experiments. The list displaying search results is large to facilitate usage for user and to take advantage of the screen space. It is easy to learn how to navigate the list. Scrolling is available if the list is long and if the user clicks on an experiment they are redirected to the experiment view displaying more information about that experiment.



Search result

Experiment Exp1
Development_Stage [Adult]
Tissue [Arm]
Species [Human]
Sex [Unknown]

Experiment Exp2
Development_Stage [Child]
Tissue [Arm]
Species [Human]
Sex [Does_not_matter]

Experiment Exp3
Development_Stage [Child]
Tissue [-]
Species [Rat]
Sex [-]

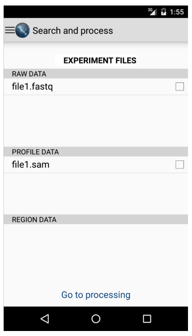
Experiment ExpBig
Development_Stage [-]
Tissue [-]
Species [Insect]
Sex [-]

Experiment ExpSmall
Development_Stage [-]
Tissue [-]
Species [Insect]

Figure 5.14: Search result view

5.4.4 Experiment view

The design illustrated in Figure 5.15 shows more information about a specific experiment. All files associated with the experiment is displayed here and sorted by type (raw, profile and region). The button ***Go to process*** at the bottom takes the user to the process view, where the user can process all raw files associated with the experiment. If no raw files should exist, the button will be disabled.



EXPERIMENT FILES

RAW DATA
file1.fastq ☐

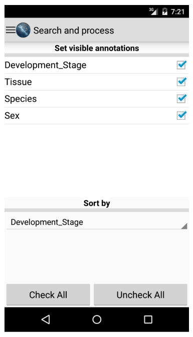
PROFILE DATA
file1.sam ☐

REGION DATA

Figure 5.15: Experiment view

5.4.5 Search settings view

The design illustrated in Figure 5.16 is showing the view for search settings. This is a way for the user to select annotations to be displayed in the search result view. The user can select annotations by checking the checkbox next to the annotation name. The user can also select on how to sort the experiments. This functionality gives the users the possibility to design the search result view the way they want to have it, which often is appreciated.



Set visible annotations

Development_Stage ✓
Tissue ✓
Species ✓
Sex ✓

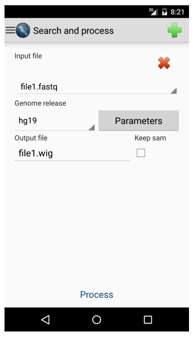
Sort by

Development_Stage

Figure 5.16: Search settings view

5.4.6 Process view

The design illustrated in Figure 5.17 is showing the view for processing. This enables the user to start a process from raw to profile. The only selectable input files are the ones that exist in the experiment. The ***Parameters*** button enables the user to change the `Bowtie` flags. The plus icon in the action bar adds a new entry, and the cross removes the associated entry.



Input file: file1.fastq

Genome release: hg19

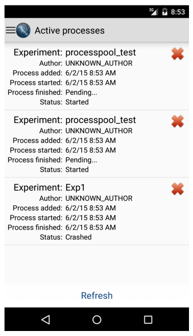
Output file: file1.wig

Keep sam ☐

Figure 5.17: Processing view

5.4.7 Active processes view

The design illustrated in Figure 5.18 is showing the view for active processes on the server. The user is automatically navigated here when the ***Process*** button in the process view is clicked. The user can here choose to remove any process on the server.



Experiment: processpool_test
Author: UNKNOWN_AUTHOR
Process added: 6/2/15 8:53 AM
Process started: 6/2/15 8:53 AM
Process finished: Pending...
Status: Started

Experiment: processpool_test
Author: UNKNOWN_AUTHOR
Process added: 6/2/15 8:53 AM
Process started: 6/2/15 8:53 AM
Process finished: Pending...
Status: Started

Experiment: Exp1
Author: UNKNOWN_AUTHOR
Process added: 6/2/15 8:53 AM
Process started: 6/2/15 8:53 AM
Process finished: 6/2/15 8:53 AM
Status: Crashed

Figure 5.18: Active processes view

5.5 iOS

Focus has been on making a nice looking application with an intuitive workflow and to follow the *iOS* design principles. Some of the design decisions are motivated in the text below.

5.5.1 Tab bar

A tab bar is used to make access to different main functionalities available at all times. Big and clear icons are used to show the user which view they all represent. It is also possible to simply swipe between the different views to increase the speed of which the advanced user can use the system. The tab bar is designed to always be located at the bottom of the screen after the user has logged in. It can be seen in most pictures of the *iOS*-application, a good example is at the bottom of Figure 5.19a.

5.5.2 Login Screen

The login screen has two responsibilities; to make a nice first impression and to make it easy for the user to login. The design is kept simple and clean to avoid distractions.

5.5.3 Search View

The search view is designed to be usable for both advanced and new users. A list with available annotations is displayed to make it easy to do basic searches fast. Some annotations can only be selected with a picker view, while others are edited by typing free text. The reason for the occurance of the picker views is to simplify searches and help the user to make correct search requests. For example, the sex of an individual can only be male, female or unknown. Other values for the sex annotation would be nonsence! The search button disappears when no annotation is selected to decrease the chance of user sending empty searches and to increase the understanding of the switches.

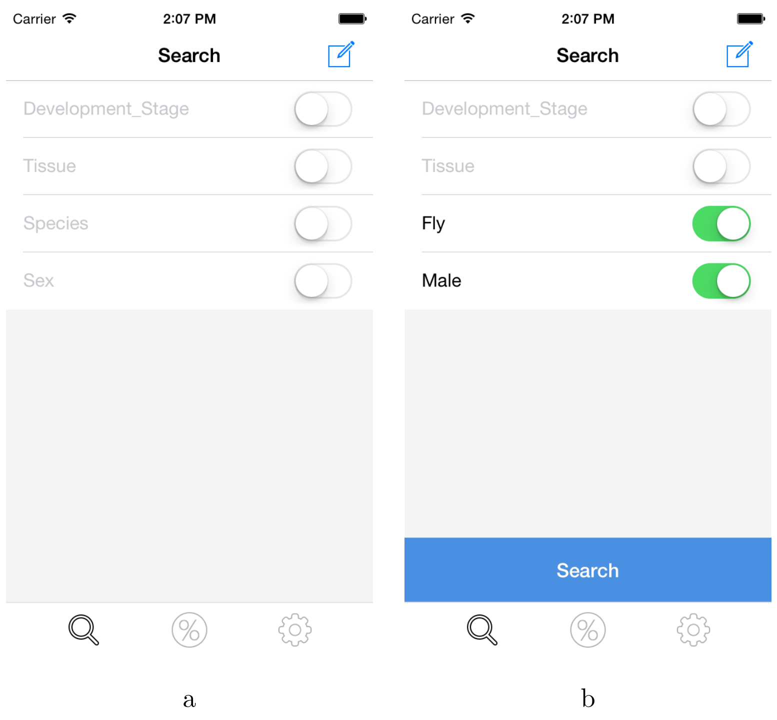


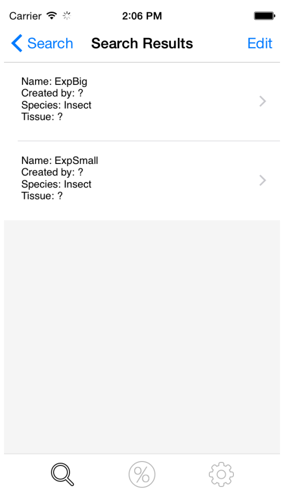
Figure 5.19: The search screen.

Each annotation has a corresponding switch button as seen in Figure 5.19a-b. The button determines if the annotation should be included in the search request. This make it easy to make small changes to the search, while not clearing the annotation values.

The advanced user can customize the search query sent to the server. This gives the user the possibility make more complex search queries and possibly make use of already acquried *PubMed*-search proficiency.

5.5.4 Search Result View

The main purpose of the search result view is to give an overview of the search results. The challenge with this view was to summarize large amount of information in a small area. The small screen of the *iPhone* made it impossible to have columns for each annotation. Instead a decision was made to group the files by experiment as seen in Figure 5.20. The table with the experiments will only expand vertically, both when the number of shown annotations and the number of experiments grows. Thus, the user never has to scroll sideways which would be awkward.



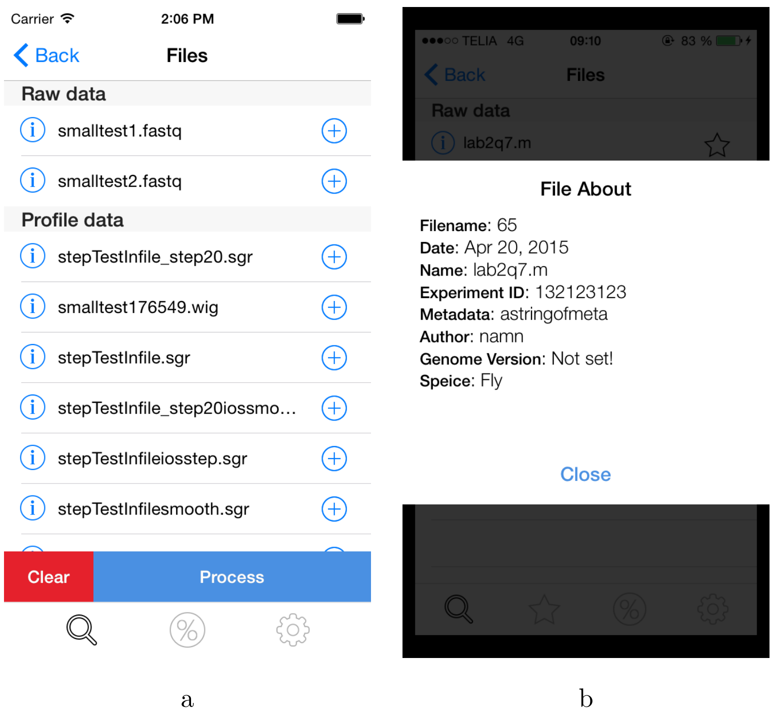
Name: ExpBig
Created by: ?
Species: Insect
Tissue: ?

Name: ExpSmall
Created by: ?
Species: Insect
Tissue: ?

Figure 5.20: The search result view.

The user can choose which annotations to display in the result view. This gives the user the possibility to only show the annotations which are interesting at the moment.

The files view (see Figure 5.21a), which is shown when the user selects an experiment, only contains the filename of the files in the specific experiment. The annotations is not shown in this view to avoid information overload and to give the user a good overview of the files. The purpose of the plus-symbol next to each file is to add as many files as the user wish to use when selecting processes. More information can be seen when the circled 'i' to the left of the filename is tapped, an example of this can be seen in Figure 5.21b.



Carrier 2:06 PM

Back Files

Raw data

smalltest1.fastq

smalltest2.fastq

Profile data

stepTestInfile_step20.sgr

smalltest176549.wig

stepTestInfile.sgr

stepTestInfile_step20iossmo...

stepTestInfileiosstep.sgr

stepTestInfilesmooth.sgr

Clear Process

a

TELIA 4G 09:10 83 %

Back Files

Raw data

lab2q7.m

File About

Filename: 65
Date: Apr 20, 2015
Name: lab2q7.m
Experiment ID: 132123123
Metadata: astringofmeta
Author: namn
Genome Version: Not set!
Speice: Fly

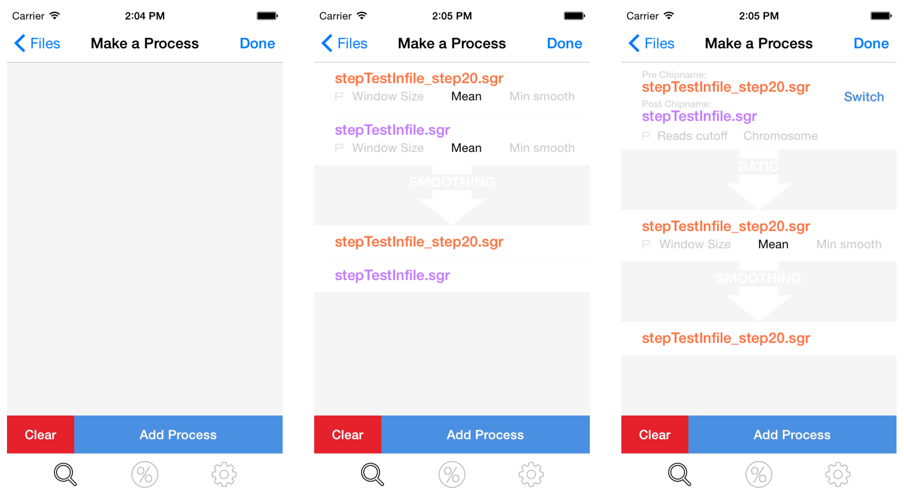
Close

b

Figure 5.21: The file view.

5.5.4.1 Create processes

The view to create a process is focused on the user's current workflow. The user wants to use the selected files from the files view as input and then select a specific process on those files, which creates output-files to be used as input-files in another process. This creates a sequence of processes which the server will execute one process at a time. Moreover each input-file for a process will be executed parallel with eachother. The input-files and the output-files are separated by the process which will be executed on the input-files. To make it easy to understand what will be executed on each step of the sequence the separator between input-files and output-files is the name of the process and an arrow from the input-files to output-files. The color of an output-file will have the same color as its corresponding input-file to track a file's conversion-process, from start to finish, see Figure 5.22a-c.

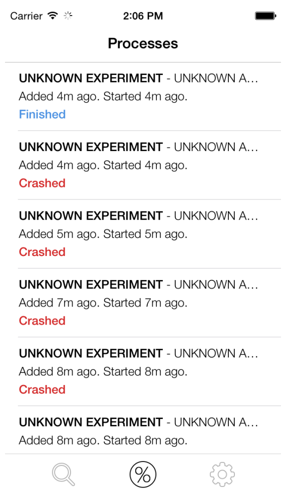


a b c

Figure 5.22: Creating a process

5.5.4.2 Processes

As visible in Figure 5.23, we chose to build the processing status view with a simple tableview, to make it dynamic and easy. We also think this gives the user the best possible overview of current processes. The status is color coded to make it as easy as possible for the user to see which processes are in which state.


Carrier 2:06 PM

Processes

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 4m ago. Started 4m ago.
Finished

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 4m ago. Started 4m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 5m ago. Started 5m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 7m ago. Started 7m ago.
Crashed

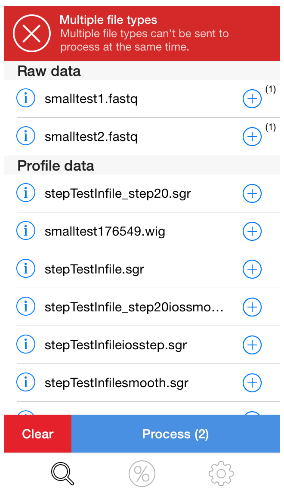
UNKNOWN EXPERIMENT - UNKNOWN A...
Added 8m ago. Started 8m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 8m ago. Started 8m ago.

Figure 5.23: The process status view.

5.5.4.3 Alerts

Alerts are simple banners animating down from the top to give the user a heads-up of what is going wrong and what is going the way it is expected. The user can tap the banner to dismiss it or if nothing is done it will animate away in 2 seconds. The banners were introduced to not stop the user with prompts which the user has to react with to be able to further use the system. A red banner, as seen in Figure 5.24, indicates an error and a white with a green icon, indicates something went as expected. The colors are used to allow the user to just notice which color pops down and give them somewhat of an understanding of what is going on without having to read the text every time.



Multiple file types
Multiple file types can't be sent to process at the same time.

Raw data
smalltest1.fastq (1)
smalltest2.fastq (1)

Profile data
stepTestInfile_step20.sgr
smalltest176549.wig
stepTestInfile.sgr
stepTestInfile_step20iossmo...
stepTestInfileiosstep.sgr
stepTestInfilesmooth.sgr

Figure 5.24: Examples of alert

6 System design

A more indepth look at how the system is designed with UML- and class-diagrams. It is divided into two main sections for the server and clients. The client section contains the different clients. After that follows the server section that is divided into different parts that makes up the whole server.

6.1 Desktop application

The desktop client is constructed around the model-view-controller pattern. It relies heavily on action events being performed in the graphical interface which is then handled by the controller. The model is the part handling the communication and the storing of important information such as ongoing downloads and the user token (used for communication authorization). In Appendix F a UML-diagram of the desktop client is presented. A basic overview can also be seen in Figure 6.1.

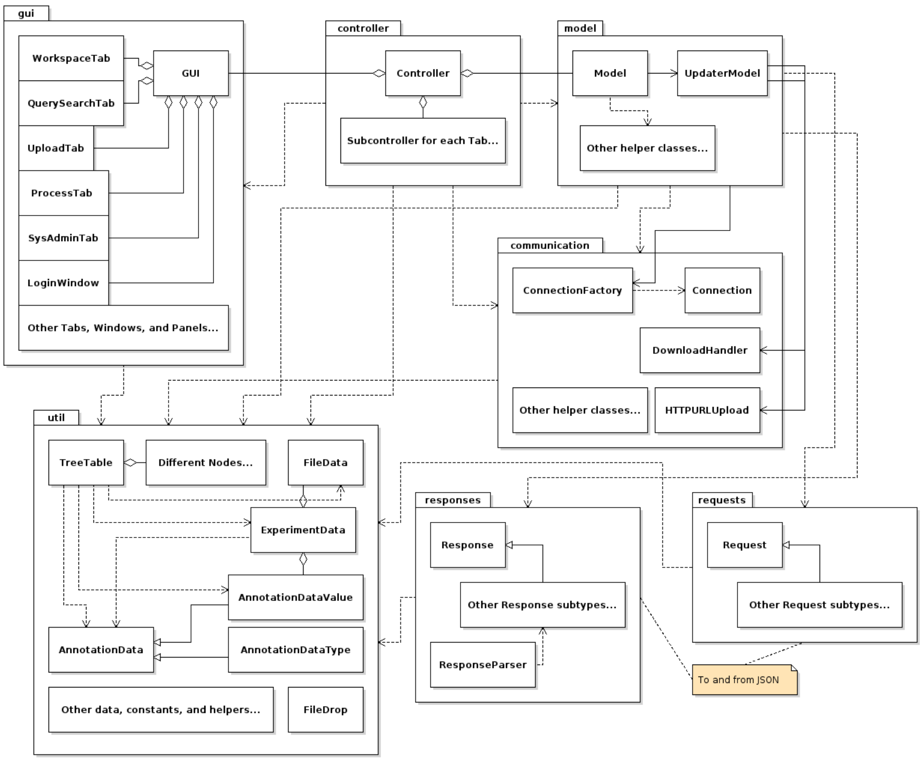


Figure 6.1: Overview of desktop client design. (Excluding SysAdmin related parts.)

6.1.1 View

The view of the *Genomizer* Desktop client is constructed with tabs. There are 5 different tabs. These are Search, Process, Upload, Workspace and Administration. In the *gui* package of Figure 6.1 these are shown.

Each tab in the view is represented by its own *Java* class. The `QuerySearchTab` class which represents the search tab can display both a search view and a results view. It uses the `QueryBuilderRow` class to construct the rows in the query builder which is used to construct search queries. The `QueryBuilderRow` class represents a row in the query builder and each row is dynamic and can change accordingly to user interaction. The search results are also implemented in the `QuerySearchTab` and the results are displayed with the `TreeTable` class which is further described in the utilities section below.

The `UploadTab` class represents the upload view of the *GUI*. It has functionality to both upload a file to an existing experiment (which is separately handled in the UploadExistingExpPanel) and to create and upload a new experiment.

The `ProcessTab` class represents the process view in the *GUI*. It contains a list where files to be processed can be stored and a large number of processing parameters which can be changed by the user. There process tab also contains a console for displaying direct feedback on processes and an area which contains the status of all current processes which are being handled on the server. The later can be updated manually with a refresh button.

The major part of the `WorkspaceTab` class consists of a `TreeTable` which holds all the experiments and the corresponding data which the user has added to the workspace. Then there is also five buttons implemented which allows the user handle the data in the `TreeTable`. These buttons are `Remove from workspace`, `Delete from database`, `Upload to`, `Download` and `Process`. The `TreeTable` view can be changed to a view which displays all current and completed downloads. This is made using a tabbed pane containing the `TreeTable` view and the downloads view.

6.1.2 Model

The model part of the system contains methods for doing most of the logic in the system. For example there are methods for sending login requests and for downloading files. There are separate classes for downloading and uploading files as well as a class for regular communication with the server called `Connection`. New connections are created with the `ConnectionFactory` class. The model also acts a storage for importing information such as the user token and list of ongoing downloads and uploads. This is shown in the *model* and *communication* packages of Figure 6.1.

6.1.3 Requests

The `Request` package contains the `Request` class, the `RequestFactory` and all the classes that extends the `Request` class. `Request` is the super class and can make a `JSON` package that all the other `Request` classes can use. All requests must have a name, type and an *URL*, but can consist of more information. For example `LoginRequest` also has username and password. `RequestFactory` is a class that can create all objects from all types of requests. It is a way to easily create all requests from the same place.

6.1.4 Response

This package consists of all types of responses that the server can send to the client-program. There is a class named `Response` that all the other response classes extends from. For example there is a response class for the login request called `LoginResponse`. All types of responses have different properties. There is also a class `ResponseParser` that can parse the responses so that the important information can be taken out of a `JSON`-package. This information can then be used to tell the client program what should happen next in the user interface.

6.1.5 Controller

The controller part of the system consists of `ActionListeners` for the different buttons and functionalities in the view. For example there are `Listeners` for searching, downloading and processing. The `Controller` class has access to both the view and the model and acts as a middle hand between those two parts of the system. Usually a `Listener` in the controller reacts upon user input and then modifies the model and gives information about the change to the view. Many of the action listeners have been divided into tab-specific classes.

6.1.6 Utilites

There are several classes which represents different data in the system. There are classes for experiment data, file data and annotation data. For example when a search response is received from the server it is parsed into experiment data and the experiment data contains file data and annotation data. There is also a class representing process feedback data. As we can see in Figure 6.1, a lot of the other packages will use some of the data or functionality within the *util* package.

The `TreeTable` class represents the table which displays experiment data, annotation data and file data in the `Search` and `Workspace` tabs. It is specially constructed to handle the data classes and it allows vertical sorting.

6.1.7 System Administration

The system administration is developed separately from the rest of the *GUI*, and therefore has a slightly different way of communicating.

Communication with the Server

All communication between the server and the system administration tab follows a line of steps. See Figure 6.2 below.

1. An event is triggered by the user clicking something.
2. The listener for the active tab receives the event and sorts out which type it is, and calls the appropriate method in the `SysadminController` class.
3. The `SysadminController` has the connection to the `Model`, and calls the associated method there.
4. The `Model` creates the corresponding request for the server, and then creates a new connection.
5. The `Connection` receives the request from the `Model` and sends the request to the server.

If the event triggers a request for data, the *Model* will use a parser to parse the data before sending it back to the *GUI* to present it to the user.

A communication example

As a more detailed example of Figure 6.2. Assume that the user clicks the 'Genome Files' tab in the 'ADMINISTRATION' tab. This will trigger an event (1) to be handled by the `SysadminTabChangeListener` (2) who will receive the event and execute the desired behavior of the tab, which is to directly show the available genome releases. This is done by sending a request to get available genome releases to the server and then parse the response.

In order to contact the server the `SysadminTabChangeListener` (2) calls the `SysadminController` (3) who uses a reference to the class `GenomeReleaseTableModel` (4) to call the method `getGenomeReleases()`. `getGenomeReleases()` will create a `GetGenomeRequest` using the `RequestFactory`.

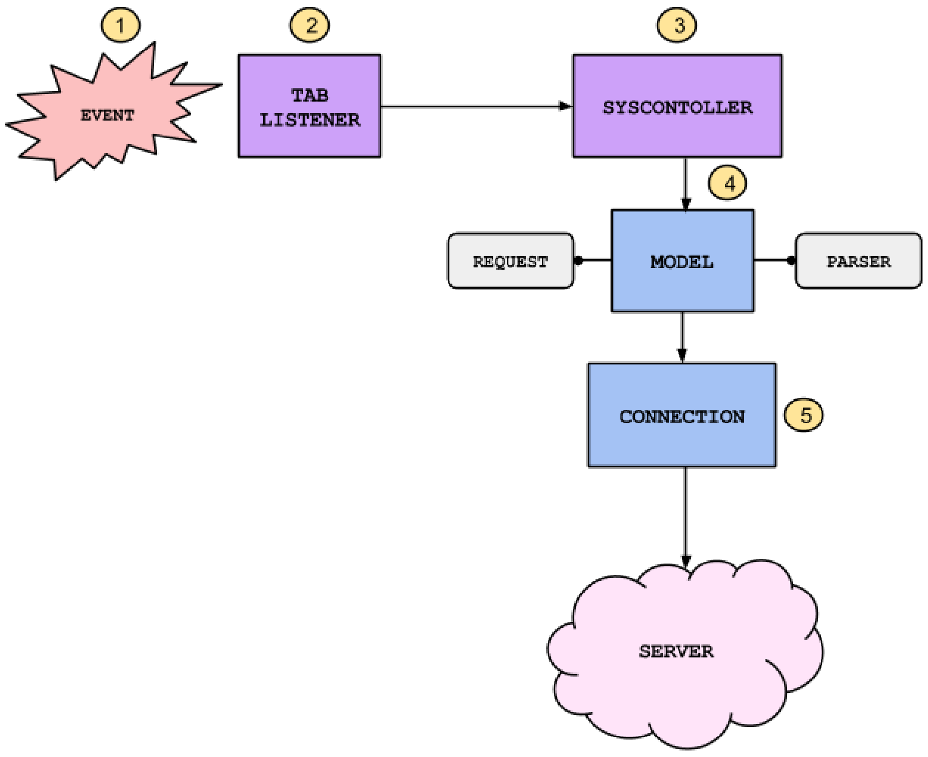


Figure 6.2: Communication Overview

The request is then sent to the server through the `Connection` class (5). The response from the server is passed to the `ResponsParser` that parses the JSON respons into wanted `GenomeReleaseData[]` object. The genome release array is return all the way back to `SysadminController` (3) which updates `GenomeReleaseTableModel` with the new `GenomeReleaseData[]` and at last lets the *GUI* know that the data has changed through a new event (1). This will trigger the *GUI* to repaint and show the available data.

Building the Administration Tabs

All tabs under the Administration tab are built in a similar fashion and then added to a `JTabbedPane` in the `SysadminTab` class. Each tab has it's own package containing all classes associated to the particular tab. All tabs are also built step by step by using smaller methods creating panels and components. Each tab has at least one main listener that is added to all components that require listeners. Once an event is triggered in a tab the corresponding listener simply use a switch case based on button/tab names to decide which action to take. The main listeners have an instance of the `SysadminController` to be able to further handle requests from the user and send them forward to the *Model* if neccessary.

Important classes The system administration part of the desktop application depends on quite a few classes and is based loosely on the model-view-controller design pattern. Here follows a list of the most important classes and a short desciption of their function and responsibilities.

- `SysadminController` - Handles the communication between the `SysadminTab` and the `GenomizerModel`. The `SysadminController` creates all `ActionListeners` for the buttons in the different views. Some minor commands are handled within the `sysadmin` package, but user commands requiring input or output from the server are recieved from the different components of the `SysadminTab` and sent to the `GenomizerModel` which converts them to `Request` objects and sends them on to the server.

- `SysadminTab` - Builds all of the different views that are displayed within the system administration tab. When creating the views it also adds the `ActionListeners` to the buttons and fields. It also holds a reference to all of the view components it has created so that information can be sent to and from the controller when needed.
- The listener classes - These are added to all of the components of the view that the user can interact with. When an action is performed, the listener performs the action that is assigned to the command string associated with the action. All of the command strings are stored in the `SysStrings` class for easy access.

Button and Tab names

To simplify the naming of buttons and tabs a class called `SysStrings` is used. All buttons or tabs are named here and then this class is used when setting the actual names. This is to avoid inconsistencies as well as making names easy to change.

6.1.8 Flow of the system

The sequence diagram in Figure 6.3 describes the flow of the system when the user presses the download file button and the diagram in Figure 6.4 describes how the desktop clients reacts to a login.

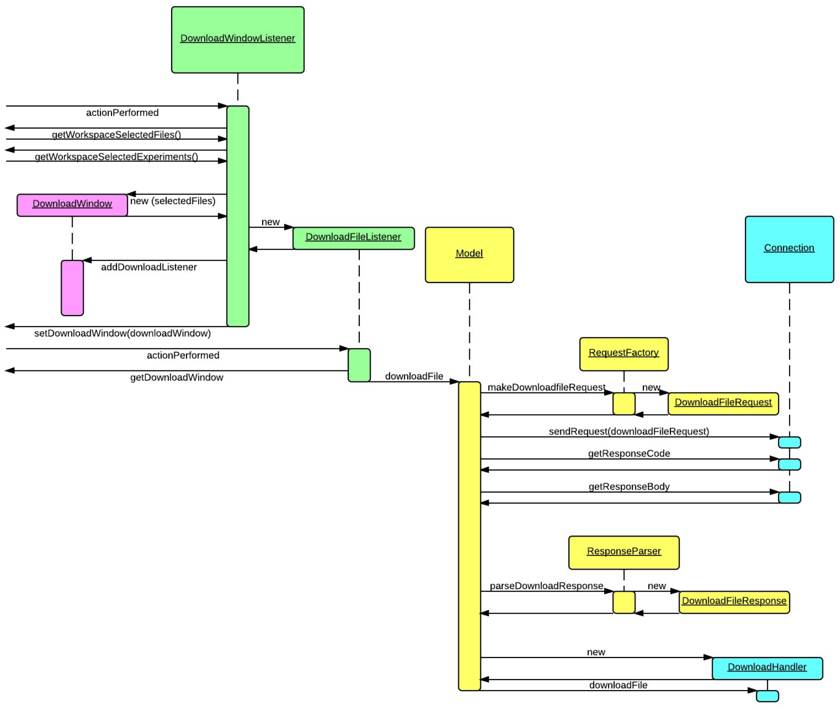


Figure 6.3: UML sequence diagram of downloading a file

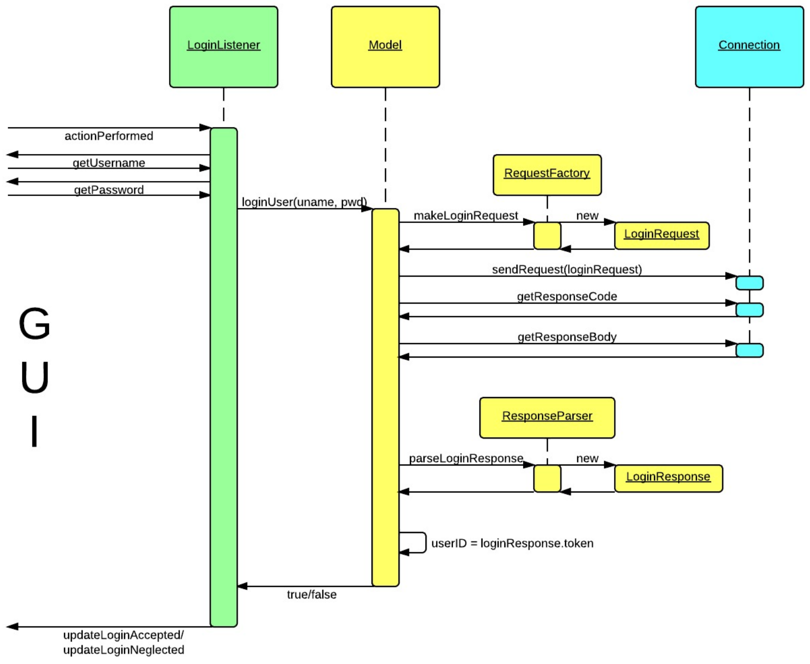


Figure 6.4: UML sequence diagram of login

6.2 Web application

This section describes the overall design of our system, first with a system overview and then with more in depth information about our tabs.

6.2.1 How the web application works

Figure 6.5 shows how the web application works in general. There is a user that interacts with a browser. A browser renders the *DOM* (Document Object Model, a convention for representing and interacting with objects in *HTML*) of the web application. How it does this is up to the browser. Different browsers might display it differently. The web app is based on the *MVC* pattern, but with the controller merged into the view and with a component called *Collection* being introduced. A collection is simply a ordered set of models. Models and collections will talk to the server to update themselves. Out of the components that go into this figure, we are in charge of (and only capable of) changing a few of these; `View`, `Template`, `Collection` and `Model`. See *Backbone* in section 7.2.1 more information.

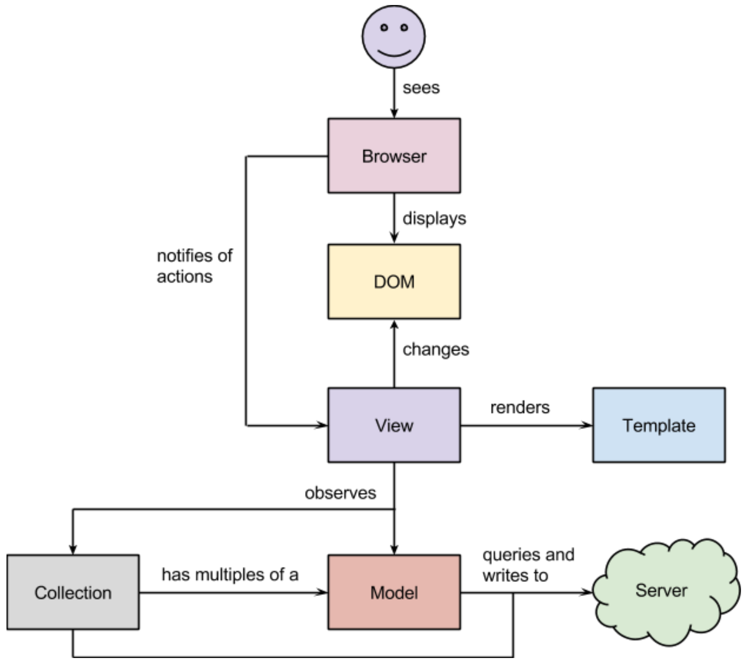


Figure 6.5: A general build of a backbone web app.

Example 2 *In the web app, there is a collection called* `Experiments` *which contains a set of* `Experiment` *models. Each of these models contains info about a specific experiment (for example, the name of the experiment and which files and annotations are incluced in it). The collection will retrieve experiments from the server and update itself with a simple call to its* `fetch()` *method. After this, the collection is synced with the server data.*

6.2.2 System overview

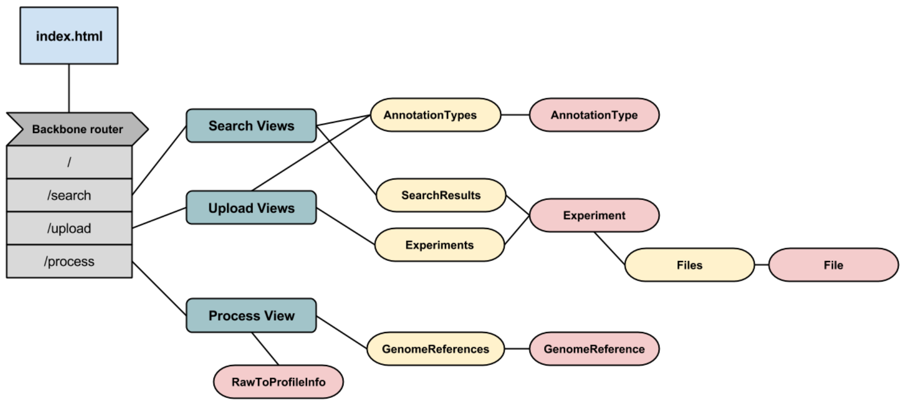


Figure 6.6: Overview of the relations between the different Javascript prototypes in the system.

The web app is divided into the parts `Misc`, `Views`, `Collections` and `Models`. In Figure 6.6, an overview of the system is shown. The views are the parts in green, the collections the parts in yellow and the models the parts in red.

> **Example 3** *In Figure 6.6, the collection `Files` contains a set of `File` models. The model `Experiment` contains a `Files` collection. An `Experiment` model may be used by the collections `SearchResult` and `Experiments`.*

The parts in grey represent the *router* which belongs in the `Misc` category. It is responsible for rerouting links. This is done mainly when the user wants to go to another webpage by clicking a link. The router is ”clever”, however, and does not need to load a whole web page whenever a rerouting is triggered - instead it only loads the necessary parts of the new web page.

> **Example 4** *When a user clicks the search tab, the router navigates to `/search`. But instead of loading the whole `/search` on top of the page we are currently on, the router will keep the old navigation bar, open the search tab alone and put the search tab below the already existing navigation bar.*

The `Misc` category also holds the `main.js`, which is in charge of setting up and starting the app. The views are responsible for the user interface, displaying information and handling events. The collections and models are responsible for holding the data.

6.2.3 Log in

Log in has a single view that is a *modal*, meaning that it is not a full page like the tabs but a pop-up that appears over the entire main view when a user enters the page.

6.2.4 Search

The search tab has three views that together make up the `Search Views` as they have been denoted in Figure 6.6. When searching for data, the models and collections will update themselves to contain the new annotations, experiments and files pertaining to that particular search, so the search views can display them. Once new data has been retrieved, the user can perform a number of actions on the displayed experiments and files.

> **Example 5** *When searching for experiments, they and their contained files will be displayed. The user may choose to delete a file by marking it and then click a delete button. If this happens, the `Search Views` will receive the event and tell the model of the marked file to destroy itself. The model will then send a delete request to the server and disappear.*

In Figure 6.7 is a simple sequence diagram for the search tab. If a user enters a query in the search field and then presses the search button, the `Search` view will update the `SearchResults` collection to have a new query. Once `SearchResults` has a new query, it will try to fetch search results corresponding to the query from the server. If successful, new experiment models for every experiment retrieved will be created and set in the `SearchResults` collection. `SearchResults` then triggers a ”change event” that `SearchResultsView` listens to. When that event occurs, `SearchResultsView` knows that `SearchResults` has been changed, and rerenders itself.

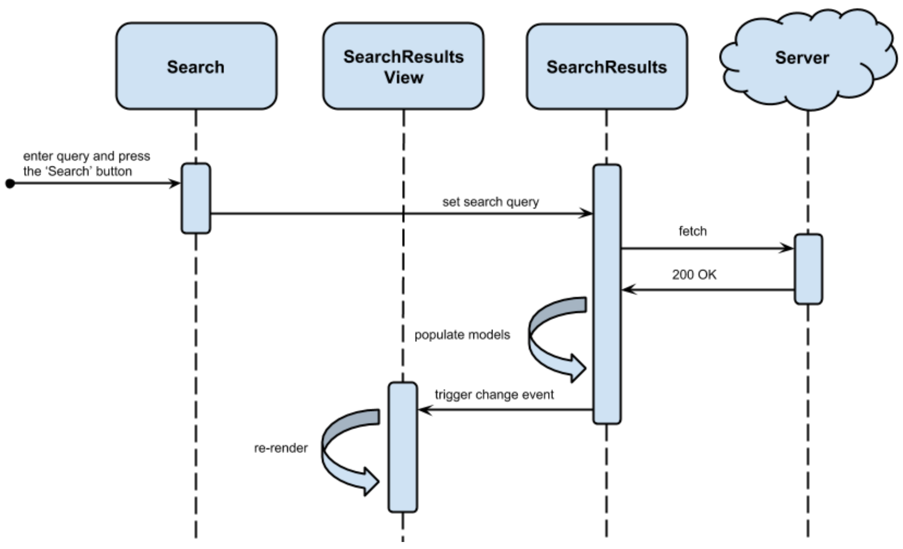


Figure 6.7: a sequence diagram showing what happens when a user enters a valid search query and results are fetched.

6.2.5 Upload

The upload tab has three views, that together make up the ”Upload Views” as we have denoted them in Figure 6.6. Unlike search (see section 7.2.1) that uses experiment and file models to retrieve information about experiments and files, upload uses the same models to create new experiments and files. To do this, it needs to be aware of what annotations are available, so it uses an annotation type collection to retrieve the current annotations offered when a user wants to create a new experiment.

6.2.6 Process

Process consists of two views that are reached either through the process tab in the main navigation bar or through the process button in the search view that allows the user to search for experiments prior to entering the process view. Process also has collections and models to store and send data necessary for a process, like genome releases available for the chosen species of an experiment.

The view reached through the tab is simply a textfield where you can enter an experiment name and a button which allows you to proceed to the process view for that experiment, which is exactly the same thing that happens when marking an experiment in the search view and pressing the process button there.

The process view is at first only a dropdown list where any process step can be chosen and added to the view by pressing the add button. Similarly, to each process step, a line can be added using the + button, which represents one run of that process step. Adding more than one line to one step allows simultaneous processing, but should any infile depend on an outfile a new process step of the same sort must be created below.

6.2.7 Convert

The convert tab is one single view that is accessible via the search view. When the user searches an experiment, selects files from the experiment and clicks the ”Convert” button, the file names and the file IDs are passed along as parameters in the URL. The convert view is then loaded and can fetch the file names and the file IDs from the URL. The file names are used to display to the user which files are selected, and the file IDs are used when sending process requests to the server. The user must also pick a new file format for the selected files, which is also sent to the server as a string. This way, the server knows which files are to be converted and to what format.

6.2.8 System administration - Web

The system administration part of the web client is developed using the same tools and frameworks as the rest of the web client. This admin part of the system is made up of view classes, model classes and collection classes. The classes are described below:

Classes used by all views

Gateway - this is a model class used solely for communication with the server. It is a static class in the sense that it doesn't have to be created. It only needs to be included and then its functions can be called immediately without having to be instantiated. The gateway class retrieves the URL from the main JavaScript file this way the URL only needs to be declared once. The URL can then be fetched by any class that includes the Gateway class.

SysadminMainView - the main view for the admin tab, this view is used together with every other admin view. It contains a sidebar menu used to navigate between different admin views.

Classes used to handle annotations

Annotation - this is a backbone model that represents an annotation. An annotation consists of three fields. A name, a list of values and a forced field. The name simply specifies the name of the annotation. The list determines whether this annotation is a drop-down list, or a free-text field. If the list contains one element called free-text, the annotation is a free-text field. Otherwise it is a drop-down list with the values in the list. The forced field determines if the annotation has to be filled in by the user when a file is uploaded.

Annotations - this is a backbone collections that consists of several Annotation models. It also has a URL that it uses to fetch annotations from the server, the URL is retrieved from the Gateway class.

AnnotationsView - this view is the basic view for displaying annotations. It has a search field and a button for creating new annotations. Pressing the button renders the newAnnotationView.

The AnnotationsView has a child view called AnnotationListView. This way the list view can be rendered separately from the search field when the user types in searches.

AnnotationListView - this view uses the Annotations collection to fetch all the annotations from the server and renders them dynamically in a list. In the list is an Edit button for every annotation, the edit button will retrieve the name of the desired annotation and navigate through the router to the EditAnnotationView with the name as a parameter. The view also has a button that will take the user to the NewAnnotationView.

EditAnnotationView - this view uses the name parameter received from the Annotation-ListView to retrieve a specific annotation from the collection of annotations. It then renders the fields with the values from the annotation. This view has a button to delete an annotation.

It will send a delete message to the server using the Gateway model to delete the annotation. An annotation can also be modified in different ways.

NewAnnotationView - this view is used to create a new annotation. It consists of a couple of fields and a create button. Pressing the create button renders a ConfirmAnnotationModal which displays the values for the annotation.

ConfirmAnnotationModal - this class extends the ModalAC class. It is simply used to display information that the user has to confirm. Pressing confirm creates a message using the Gateway class and sends it to the server.

Classes used to handle genome releases

GenomeReleaseView - this view is used for viewing, adding and deleting genome releases. It contains a button ”Select files to upload” which opens up file explorer and lets the user select one or multiple files for uploading. When the user then presses upload the UploadGenomeReleaseModal will open. Below the button the view has a table showing the current genome releases available on the server. The user can hold the mouse over files too see all files included in that genome release. A ”Delete” button is shown next to every genome release and if pushed sends a delete request to the server through the Gateway class.

UploadGenomeReleaseModal - this modal shows the user which files has been selected for upload and asks for information about which species and genome version they are for. Then at the press of ”Upload” the files starts to upload and the user will see a progress bar over the complete upload progress.

GenomeReleaseFiles - this is a collection with GenomeReleaseFile as models. It handles the ordering and filtering of its models.

GenomeReleaseFile - this model represent a genome release and can contain multiple files in itself since one genome release is almost never just one file. This class takes case of uploading itself to the server and thereby also updates the progress bar through events that propagate up to the GenomeReleaseFiles collection.

6.3 Android application

The following sections describe the system design of the *Android* application. All functionality of the system components are described in this section. Worth noting is that the figures referred to in this section can be found further down in the document.

6.3.1 System overview

The *Android* application is divided into seven *packages*. These packages are **`default`**, **`com`**, **`login`**, **`model`**, **`process`**, **`processStatus`** and **`search`**. All packages (except for `com` and `model`) contains one or several fragments. A `Fragment` is an object that helps to modularize the code and brings more sophisticated user interfaces.

The user will interact with an activity that holds a fragment. The fragment (which contains some logic) will tell the class `ComHandler` what action that should be performed. The `ComHandler` will construct a message by using `MsgFactory`. The message is then passed on to `Communicator` which sends the message to the server with *REST*. The `Communicator` returns the response to the `ComHandler` that parses it by using the class `MsgDeconstructor` and then returns it to the fragment. Hopefully, Figure 6.8 will bring some clarity.

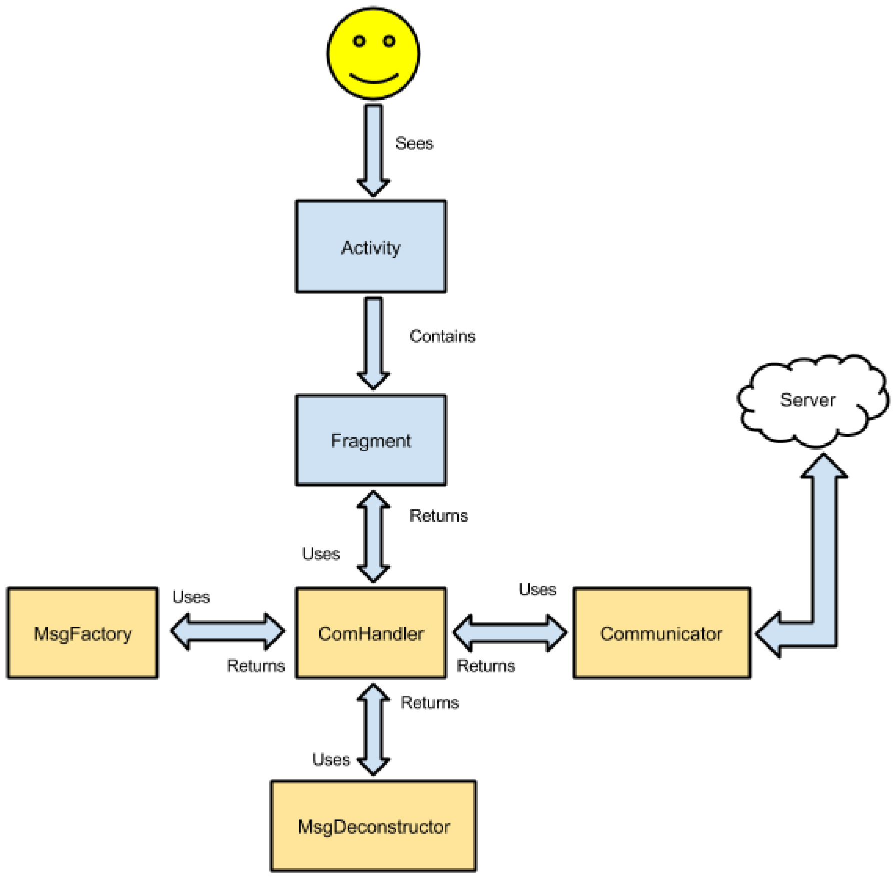


Figure 6.8: A generalization of how the Android application works.

6.3.2 Package overview

The `default` package contains the `MainActivity`. This class is the base of every screen in the application after logging in. The top level navigation is handled from here.

The `com` package is responsible for communication with the server. It also contains classes and methods for construction and deconstruction of `JSON`.

The `login` package contains the *GUI* and controller for the login screen. It also enables the user to select, add, edit and delete server URLs.

The `model` package holds information about experiments, annotations, files etc. found on the server.

The `process` package is responsible for displaying processing parameters etc. for when the user wants to start a process.

The `processStatus` package shows the running, failed and succeeded processes on the server.

The `search` package handles searches by either selecting annotations, or by manually typing in *PubMed* style. It also handles the search results by displaying the found experiments in a list.

6.4 iOS application

The following sections describes the system design of the iOS application. The overall system design is discussed followed by a more detailed description of how the segues are controlled.

6.4.1 Overall system design

The system is designed using the model-view-controller principle. Each view is controlled by its own controller class which reacts to user input and triggers changes in the model and updates the view accordingly.

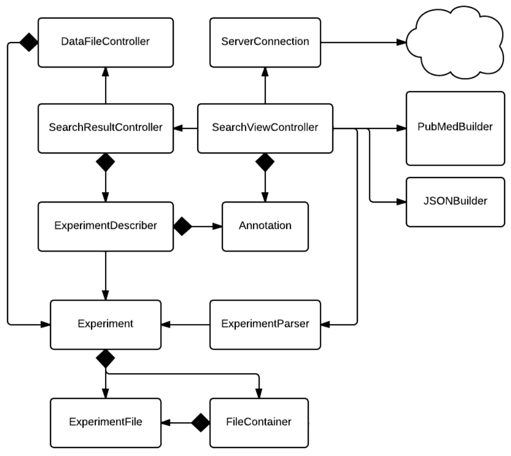


Figure 6.9: UML diagram.

Figure 6.9 gives an overall image of the system design. Some classes are excluded from the figure to make it easier to get an overall idea of the system. The controller classes of the table cells and some other controller classes are not illustrated in the diagram. The non-excluded classes are described in Table 6.1.

Class	**Description**
`Annotation`	Contains information about an annotation and can format the annotation name to an aesthetically more pleasing representation.
`DataFileViewController`	Controls the File view. It contains a reference to an experiment and lists all its files in a table.
`Experiment`	A class that contains information related to an experiment, as well as its files.
`ExperimentDescriber`	Generates a description of an experiment using annotations chosen by the user.
`ExperimentFile`	Contains information about a file from an experiment.
`ExperimentParser`	Parses experiment information from a NSDictionary to an Experiment object.
`FileContainer`	Contains files and sorts them by file type.
`JSONBuilder`	Creates different `JSON` requests.
`PubMedBuilder`	Creates a pubmed search query.
`SearchResultController`	A controller class for the Search Results view presented in Figure A.57. It configures the table which holds the information about the experiments a search resulted in. An ExperimentDescriber is used to generate a description of the experiments.
`SearchViewController`	A controller class for the Search view, see Figure A.54. It checks which annotation-fields are used and tells the `JSON`Builder to generate a corresponding search query when the user presses the search button. The class also contains a advanced search to allow the user to manually enter search queries.
`ServerConnection`	Handles sending `JSON` objects to the HTTP class and receives and handles `JSON` objects from the server.

Table 6.1: Description of some classes of the system.

A more detailed description of these classes, and the ones not mentioned here, can be found in comments in the source code.

6.5 Server

The design of the servers system is based around several parts. These parts consists of: communication, conversion, processing, storage and file transfer. Following is a more detailed description of each part.

6.5.1 Communication

The server is based around a `RESTful` protocol where all communication is handled over non-persistent connections which the clients initiate. Since the communication is non-persistent, the server has no way of contacting clients except for responding to requests. When a client wants to connect it sends the request to a proxy, which only accepts encrypted trafic, that is then forwarded to the actual server. Once inside the server, the request is parsed, executed and a response is sent back to the proxy which forwards the message back to the client.

To uniquely identify different logins a token is generated when a user logs in, the client now should identify itself with this token in all other requests for them to be executed. Otherwise, the server will not recognize who the client is and therefore can't know what server commands the client has permissions to execute.

Most commands are executed immediately when ther server recieves it, and the result is sent back as soon as the command is finished. There are however an exception to this, the process command, which is put in the back of a queue instead of being executed. The server continuously takes work from this queue and executes them as fast as it can, but due to the huge computing power requirement it cannot do them all at the same time.

For a visual reference of the flow between the different parts of the system, see Figure 4.3 on page 4.

Server Commands

The following eleven items are the main categories of commands that can be sent:

- *Connection*
- *Experiment*
- *Files*
- *File conversion*
- *Search*
- *User*
- *Admin*
- *Processing*
- *Annotation*
- *Genome release*
- *File upload/download*

Connection handles the *Login* and *Logout* commands, which are self-explanatory in their functions. There is also

Connection

Connection handles the *Login* and *Logout* commands, which are self-explanatory in their functions. There is also a deprecated command which can be used, but should not, to check if the clients token is still valid or if it has expired. This was used before, but was deemed unnecessary due to this check happening on every other command as well.

Experiment

Used to create new experiments, update or delete existing experiments as well as retrieving information about specific experiments. Deleting or retrieving information only requires the experiments ID, whilst creating new or updating existing experiments require annotations to be specified as well.

Files

Contains commands to create new file-posts, update or delete existing file-posts and retrieving information about specific file-posts, just as Experiment does for experiments, but for file-posts. A file-post is a database entry which keeps information about a file, as well as the path to the file. A file cannot be uploaded without having a matching file-post. When discussing files in general, file-posts and the file together will be refered to as a file.

File conversion

File conversion has a single command, which converts files from one file-format to another. The formats that can be converted from and to are: `.bed`, `.gff`, `.sgr` and `.wig`.

Search

Search is used for searching after experiments in the database, the search uses a PubMed-style query system which can be found and explained at `http://www.ncbi.nlm.nih.gov/pubmed`. All experiments that match the query are sent back to the client. No post-processing or ordering is done on the list ofexperiments by the server.

User

Only contains two commands at the moment, update and retrieve information. Via the update command users can updates their password, name (fullname, not username) and email. Any other editing of users is done via the Admin category.

Admin

The Admin commands are the primary way of creating, editing and deleting users. Creating a new user requires a username, password, privilege level, name and email. To make editing and deleting easy to use there is also a command to get a complete list of all the usernames in the system, which together with the get user command from User, a client can get all the information about any user.

Process

In order to process files, the client can send a process command which is a collection of sub-commands, one sub-command for each step of the processing pipeline. Each of these sub-commands contain all the information they need to run and a list of infiles and outfiles.

When a process command is executed, it executes the each sub-command in order. Since a sub-command might contain many input files and output files, it in turn executes on all the input files, producing all the output files before finishing, and thus, causing the process to be parallellized in each step, but each step is sequential in order.

Process also has commands to retrieve information about all the processes that are waiting, running or finished as well as canceling a running or waiting process.

Annotation

Annotation has two different sub-categories, annotation field and annotation value, the field is the name of the annotation and the value is the actual value. A annotation can only have a single field, but several values, and is displayed with dropdown menus in the clients. The reason for two different sub-categories is that both of the two need to be able to be created, edited, deleted separately. The retrieve only retrieves full annotations, i.e. both the name and all the possible values.

Genome release

Genome release is used to manage genome releases, works similarly to how file works, except a single genome release-post can have many files associated with it.

A more detailed specification of the API can be found in Appendix I.

6.5.2 Data Conversion

The *Genomizer* service needs to be able to convert, process and visualize data. This chapter explains how this is done in the system.

The `RawToProfileConverter`, `Smooth`, `Step`, `Ratio` and `Bowtie` extends the `Executor` class. The different processing commands can only start the corresponding processing method on these Executors.

6.5.2.1 Executor

The executor class, as seen in figure 5.2.1, is a abstract superclass that is an entity that is able to execute various commands. The executor class is able to run programs as well as scripts and shell commands. The commands are specified in the call to the methods in this class.

executeCommand	*executeCommand* is a protected method used in processing to make command line calls to external dependencies used in the various processing steps. Firstly a *processBuilder* is used to ensure a safe way to execute commands, after that the working directory is set and the error output stream is merged with the standard output. After a command has been started the output stream is then recorded with the help of a Scanner object and a *stringBuilder* object. When the command has been executed the termination status is checked and the recorded string is sent back to the caller. The command to be executed is represented as an array of strings.

6.5.2.2 RawToProfileConverter

The purpose of the *RawToProfileConverter* class is that it will be used by *RawToProfProcess-Command* and do all the steps in the process pipeline produce a *profile file* in *.wig* format. These steps are done by calling external dependencies such as programs and scripts which are executed with methods that is extended from *Executor* class.

6.5.2.3 Smoothing

The termSmoothing class is used on a *profile file* to smooth down the tips, making the data result less jagged.

6.5.2.4 Step

The termStep class is used on a *profile file* to lower the file's resolution.

6.5.2.5 Description of different scripts and processing steps

1. *BowTie*: Uses the external dependency Java tool Bowtie. Support for Bowtie2 is implemented but not fully tested. Bowtie creates unsorted *.sam* files from *.fastq* raw files. The files are created in a temporary folder with the name `result_X`, where X is the ID of the current thread. All other folders created is placed inside the folder from where the files used where placed.
2. *sortSam*: Uses external dependency Picard and sorts the *.sam* file and creates a new *.sam* files, sorted by coordinates. The files are saved in the same temporary folder as in the Bowtie step.
3. *RemoveDuplicates*: Uses the external dependency Python tool Pyicos. Takes a sorted *.sam* file and produces a new *.sam* file with all duplicate reads removed. It is optional to save this *.sam* file to the database but it is saved in the temporary directory in the mean time.
4. *Convert*: Uses external dependency Python tool Pyicos. This is the final step of raw to profile conversion and uses Pyicos to convert a given *.sam* file to *.wig* file. All intermediate files are removed except optionally the *.sam* file which can be returned together with the final *.wig* file. All saved files are moved to the given profile directory path.
5. *Smooth*: smooths the file and creates a large *.sgr* file, converted the customers *Perl script* by following the algorithm they sent us. This makes it more efficient. Puts the files in a folder called `smoothing`.
6. *Step*: Takes the smoothed *.sgr* file and takes samples from it with a specified interval and creates a smaller *.sgr* file. If stepping is done the files will be placed in the same folder as the previous step.

7. *Ratio Calculation*: Creates four *.sgr* files with the *Perl script* provided by the customer. Puts the files in a folder called `ratios`.
8. *Smooth*: After the ratio calculation, smoothing needs to be done again with different parameters. Puts the files in a folder called `smoothing`

procedure	Executes all the steps to make a profile .sgr file from a raw file, it checks the directory it gets as file-path so that it contains the raw files and that there are not more then two files, but at least one file to process. Does the procedure to create a profile data and move it to the folder thats specified as a parameter.
runBowtie	Constructs a long string with the full execution line for BowTie. It then uses this string as a parameter when calling the method parse. The resulting array is then used when calling executeProgram and the result of the execution is returned.
sortSamFile	Constructs a string with the full execution line to sort a sam file. It then calls parse to create a string array from the full string and sends it as parameter to executeShellCommand which runs a shell command to sort the file and creates a new .sam file that is sorted with the specified parameters. • *makeConversionDirectories* – Creates the necessary directories used by RawToProfile's procedure to put the temporary files needed to do all the steps to create a profile .sgr file. • *initiateConversionStrings* – Defines all strings needed for the directories created when procedure is doing its work. Also defines a string for each step in the procedure, which gets passed to the corresponding execute methods.
getRawFiles	Constructs a File object with the parameter inFolder that should be a directory where the .fastq files that the procedure should run on are. returns an array of File objects with all the files procedure will be using.
makeConversionDirectories	Creates the necessary directories used by RawToProfile's procedure to put the temporary files needed to do all the steps to create a profile .sgr file.
initiateConversionStrings	Defines all strings needed for the directories created when procedure is doing its work. Also defines a string for each step in the procedure, which gets passed to the corresponding execute methods.

validateParameters	Validates all parameters for the steps procedure should run on. Checks whether a step should be run. If so, validates that steps parameters, returns true if everything is correct.
checkBowTieFile	Checks that bowtie succedded to run and that the result is ok. Checks that bowtie created the file it should and that the size of the file is not zero. If everything was correct it returns true.
validateInFolder	Perform a check on the parameter inFolder that should be a string with a path to where the files to be processed should be. If the string ends with a "/" it gets removed from the string.
runSmoothing	When implementing the scripts to create profile from raw we realized that the smoothing script used alot of memory to run, so we decided to convert it and try to optimise it. The result was a improvement and in this method we uses the smoothing a version of the smoothing that got approved from the customers. The method sets up all parameters the way the smoothing class wants them in and fixes all the paths then we run the smoothing, The method checks whether ratio calculation have been run before smoothing or not and sets the paths and parameters accordingly.
isSgr	Gets a string that should be a filename with the file extension and checks if the file extension is ".sgr", if so it returns true.
correctInFiles	Takes an array of File objects and checks that it contains a correct amount of raw files to process.
doRatioCalculation	Initiates a string using the incoming parameters and executes the the script to do Ratio calculation, uses the method executeScript to execute.
checkBowTieProcessors	Bowtie has a parameter where the number of processors used can be specified, we want to restrict the user from being able to run bowtie on all the cores on the server cause that would slow it down. Instead we make it so that bowtie runs on all but 2 of the available cores on the system.
verifyInData	Makes a initial check so that all the incoming parameters to the procedure method not is null. Also checks that the array string with parameters is correct size, not zero and not bigger then eight.

6.5.2.5.1 BowTie

BowTie takes a raw *.fastq* file together with a genome release and converts the *.fastq* file to a *.sam* file, which is the first step to make the desired *.wig* file. After a *.sam* file is converted the external dependency *Picard* is run with its internal command *samSort*, which produces a sorted *.sam* file sorted by chromosome and position as needed to use the scripts.

6.5.2.5.2 After-processing scripts

The different functions of the Perl scripts is explained below. They are explained in the same order that they are executed. All scripts take a directory of files to be processed as input parameter. The given Perl scripts are modified and wrapped by expect scripts in order for better usability and callability from the Java implementation.

6.5.2.6 Ratio calculation

Does ratio calculation on the processed files, for each position in the IP sample with at least one mapped read, a ratio of IP - input (on a log2 scale) is calculated. If the read count in the input is below the read count mean (in the input sample) is calculated it is set to the mean (or double mean (2 x mean) as user specified). If the input mean is below four the minimum input value is set to four (to avoid division by near zero values. Calculated as (read length x approximate total number of reads in input samples(9 millin))/ genome size (for Drosophila melanogaster 120381546)). A random number between -0.5 and 0,5 is added to the read counts before log2 conversion to make them discrete for statistical analysis. All ratio values are then adjusted by reducing each value by median of the ratios. This linear adjustment is carried out in order to compensate for differences in IP and input sequencing depth. Also, to visualize ratios distribution, ratios are plotted by binning ratios with user specified numbers of bins and minimum and maximum ratio values (200bins,minimum ratio value: -10, maximum ratio value:10). Ratio values are printed in sgr format.

6.5.2.7 Smoothing and stepping

Both Smoothing and Step are implemented as separate classes calling external Perl scripts. The classes provide some validation and a clean interface towards the external dependencies. The programs can handle file corruption to some extent. If the file contains empty or wrongly formatted rows the program will not crash, it will simply ignore the corrupt rows.

6.5.2.7.1 Smoothing

Smoothing means that a trimmed mean value or median value for a position and its surrounding positions is calculated. The number of positions to smooth on is called the Window Size. For example: with a window size of 10, the smoothed value on position X is calculated on the interval (X-4, X+5). A number of position which below shouldn't be smoothed at all should also be provided. There's also one parameter called stepSize, if the stepSize is one the program will not do any stepping but if it's larger than 1 stepping will be done. Stepping is handled in this program by simply checking every time we are going to write to the new file if the current row's position is divisible with the stepSize, if it is we write to the file, otherwise the row is discarded.

6.5.2.7.2 Step

Step also takes a window size, the number of genome reads to skip. This afterprocessing reduces the granularity of the file and thus the file size, whilst information is lost of course.

6.5.2.7.3 Tuple

The tuple class is a data carrier that represents one row of data in an sgr file. It consists of the fields chromosome, position, signal and newSignal. Where signal is the signal-value read from the infile and newSignal is the updated value after smoothing have been done. The methods in this class are all standard getters/setters except for the method toString which formats a row for the outfile and rounds of decimal numbers. The constructor is also of interest since it parse a row on tabs. Thus the fields in an infile needs to be seperated by tabs and not spaces. The constructor will throw an exception if the line it tries to parse is either null or if it does not consist of three columns separated by tabs where the first is a string and the second and third is a double.

6.5.2.8 ProcessHandler

The ProcessHandler is a controller that handles process-calls. Depending on the name of the process it handles it differently. It acts as an interface between the process-module and the rest of the program.

6.5.2.9 Logic & interface

The main logic in the ProcessHandler is a switch-case that switches on the name of the process being called. For example if the name of the process is “RawToProfile” is sets up a RawToProfile-converter and calls it.

processName	A string that tells the handler which kind of process should be executed.
procedureParams	A list of string with the parameters to the different external programs/scripts that will be called during the execution. The first element will be a string with parameters/flags for the first external program that will be called, and so on.
inFile	A string with a path to the directory containing the files that should be operated on.
outFile	A string with a path to the directory where the result .sgr files should be put.

:

6.5.3 File-transfer

In the current version of the program the desktop clients and the web clients connect to different software on the server. The desktop clients connect directly to the server communication software whilst the web clients connect via the apache server and all non web requests that is to be calculated using the server software is automatically redirected by apache. The redirect is setup in a way that all GET requests that have a /api/ tag in the URL will be redirected. The exception for the desktop clients are file up- and downloads which are done through the apache server.

The download and upload will work for all platforms although this will not be implemented for Android and iOS clients due to hardware limitations.

If the client wishes to upload a file to the server they first send a request to the server-system which authenticates the client and stores the annotations for the file. The download and upload path is validated by the script to ensure that no invalid paths are sent to the scripts.

In Figure 6.10 below it is shown how the systems handles the different types of messages the client-systems can send. The big square represents the Apache server with different parts of the Apache server within. The iOS and Android clients can only send some requests to the server-system. Meanwhile, the desktop client can send requests to the server-system and upload and download to/from the web server. The web client sends all its messages to the Apache server and if it is a request to do some sort of computation it will be redirected to the server-system and if it is a download, upload or web-page message it will be sent to the web server.

The current version of the system utilizes a file structure to organize HTML- and file requests on the server, the structure is illustrated in Figure 6.11. The Web-root folder contains the

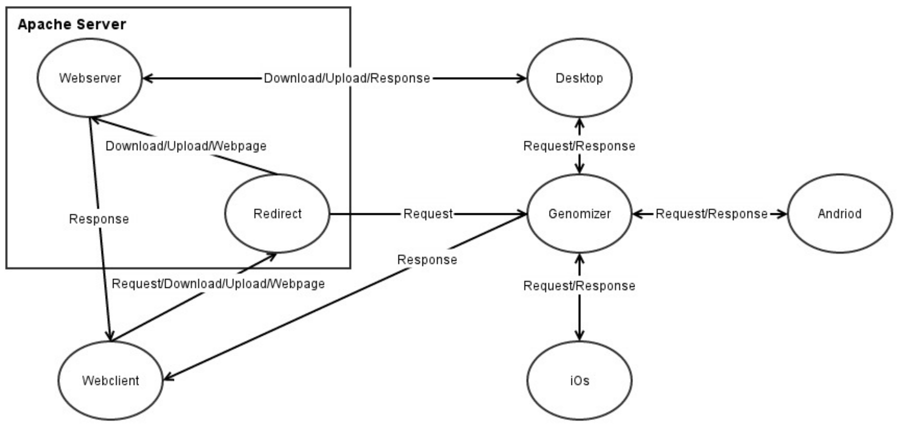


Figure 6.10: The different types of messages sent between the systems.

PHP-scripts for uploading and downloading files. The app folder contains the *Genomizer* web page. All folders of the experiments are located in the data folder, which contains folders for the different data-types.

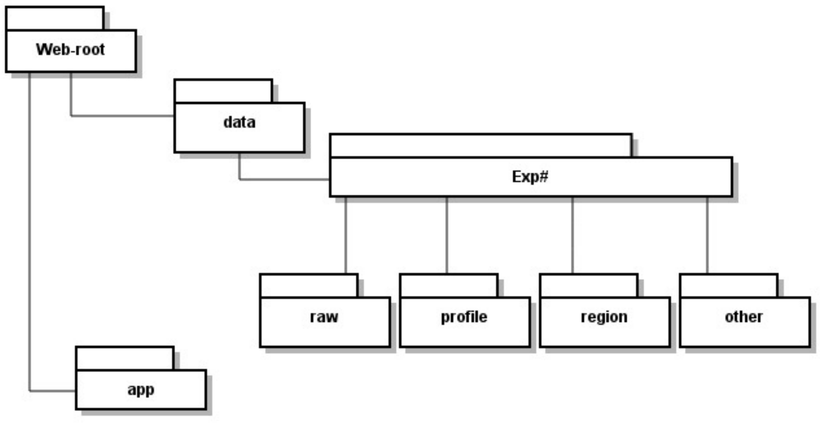


Figure 6.11: Illustrating the current file tree on the server machine.

6.5.4 Data Storage

In order to enable the annotation and subsequent searching for experiments and files the data stored on the server is complemented by a database of information. Each file uploaded to or generated by *Genomizer* belongs to an experiment which is identified by the experiment ID (expID). Each experiment created by the end user results in an entry in the database's *Experiment* table. Each experiment contains files that were either uploaded to the server (e.g. *raw* data from an experiment or from an external source) or processed by the server from the uploaded files (e.g. *profile* or *region* data). The full database schema is shown in Figure 6.12.

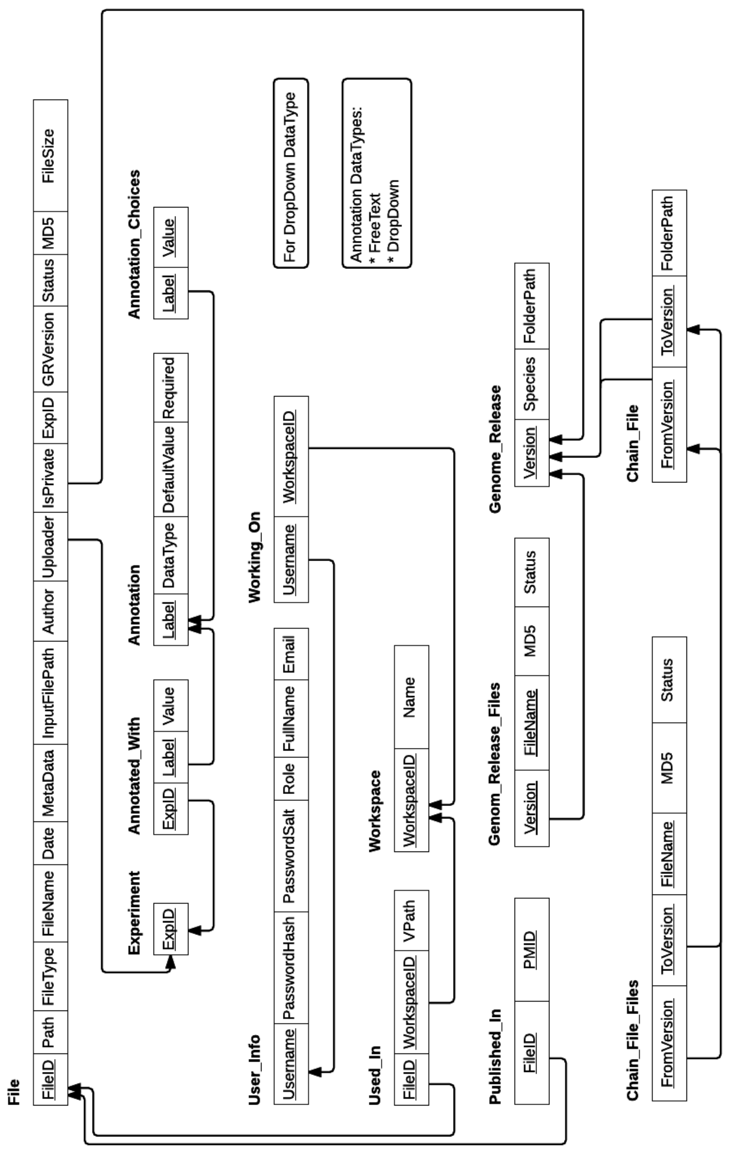


Figure 6.12: The database schema

6.5.5 Database Design

The following section will explain the less obvious columns and their intended use.

- `FileID` is the identification number for a specific file. The data type `SERIAL` is used and will therefore be auto–generate unique identifiers upon insertion.
- `Path` is the path to the corresponding file in the file system, for example: `/var/www/data/Experiment1/raw/rawFile1.fastq`
- `MetaData` is the string of parameters used in processing and should be `NULL` for all raw files.
- `Annotated_With` is the table that enables the annotation of experiments. The annotation in this table references the `Annotation`-table, to verify that new annotations are valid.

Example 6 *To set the Species-annotation to "Dog" for the experiment Experiment1, the following tuple would be inserted into the Annotated_With–table:*

ExpID	*Label*	*Value*
Experiment1	*Species*	*Dog*

- `Annotation` is the table containing all the possible annotations a user can use to provide extra information about an experiment. This includes the type of annotation which is *Drop Down* for annotations where the user can choose from a drop down list, or *Free Text* where the user can enter the value freely. There is also support for a default value and annotation forcing where users are forced to provide the information. For *Drop Down* annotations the table `Annotation_choices` specifies the valid choices.
- The `Genome_Release` table stores information about the *Genome Releases* available for use. This includes the unique version code for a *Genome Release*[21]. The `Genome_Release_Files` table stores the information about the files that make up the *Genome Release.*

6.5.6 The Data Storage Subsystem

All the classes used in the manipulation of the database and the creation of the file systems directory structure is contained in the java project's `database` package.

The other *Genomizer* subsystems execute all updates to the data storage through the `DatabaseAccessor` class. As a result there are many methods in this class, however most methods forwards the request to one of the classes in the `database.subclasses` package. Here the methods that modify the different areas of the data storage system are separated into different classes of more manageable sizes. An UML diagram of the `DatabaseAccessor` class and its subclasses is available in Figure G.1 in Appendix G.

The `DatabaseAccessor` utilizes a number of classes in order to return information to the method caller. These classes are contained in the `database.containers` package and are as follows:

- `Experiment`
- `FileTuple`
- `Annotation`
- `Genome`

An UML diagram of these classes is also available in Figure G.2 in Appendix G.

6.5.7 Interaction

Below are examples of typical interactions with the `DatabaseAccessor` class.

6.5.7.1 Adding an experiment

In order to add an annotated experiment the following steps must be followed:

1. First the *addExperiment* method must be called. This will add one experiment to the database without any annotations set for that experiment. If you try to add one experiment that already exist then the addition will be refused and an exception will be thrown.
2. If there are no annotations that can be used to provide extra information about the experiment they must first be added by calling the *addFreeTextAnnotation* or *addDropDownAnnotation* methods. If a *Drop Down* annotation already exists but there is no suitable choice for the experiment, a choice can be added by calling the *addDropDownAnnotationValue* method.
3. An available annotation can be used to provide extra information about an experiment by calling the *annotateExperiment* method.

Now that an experiment has been added files can be added added to it.

6.5.7.2 Annotation Handling

Annotations can be handled using the methods below.

getChoices	gets all the available annotation choices connected to a specific label. For example the possible choices returned for the label "sex" might be "Male, Female and Unknown".
getAnnotations	returns all annotation labels currently stored in the database. Examples could be "Sex,Species,Tissue,etc.".
getAllAnnotationObjects	Combines the two previous methods. Here an annotation object is returned that holds all the relevant information including the label, datatype, and the possible choices for a *Drop Down* annotation.
changeAnnotationLabel	updates the given label in the database. This will change the label for all experiments that use it. For example changing "specie" to "Species".
changeAnnotationValue	updates a value for a specific annotation label. For example changing "Human" to "Homosapien".
updateExperiment	Updates an annotation for one specific experiment. Example: "experiment1, Species, Homosapien" can be changed to "experiment1, Species, Fly".
deleteAnnotation	deletes an unused annotation from the database. This will also delete all the choices for that annotation.
removeAnnotationValue	removes a single annotation value for a particular label.

6.5.7.3 File Handling

The actions of adding and deleting experiment files or genome releases can be performed using the following methods.

addNewFile	To add a file you will need to have an experiment added before you call the *addNewFile* method. Raw files usually come in pairs and so they can be added together by specifying the input file name.
deleteFile	Deletes the given file from both the database and the file system. This can be done by either specifying the path or the file's ID number.
addGenomeRelease	Genome release files must be added one at a time by calling the *addGenomeRelease* method. This method returns an upload URL.
removeGenomeRelease	*removeGenomeRelease* removes all the files associated with a genome release. This can only be done if there are no files that have been generated using the specified genome release.

7 Implementation

This section contains descriptions of the implementation and how different parts of the system are designed, and what tests has been used to ensure its functionality. Here developers can get an understanding of how and why the different parts of the server was created.

7.1 Desktop application

The desktop client is implemented in `java 7`. The graphical part of the client is made with java *swing* and the external library *swingx*. The tree table which is used in the grapical interface is implemented using a modified version of the `JxTreeTable` found in *swingx*. The modifications made to the `JxTreeTable` is that a sorting mechanism has been added and it is possible for the user to choose which columns to show. Other packages needed for the user interface include the `MIG-layout` layout-manager.

The communication with the server is handled with a `http` protocol involving `JSON`-formatted bodies. The external library `GSON` and the *Apache Http Client* are used for the communication. Connections are done over the ecrypted `SSL` communication layer, using `java(x).net` modules. The `SSL`-certificates are however never verified, and validated ones not used, limiting the security provided.

For dragging and dropping files into the upload tab, the desktop client uses a modified version of the class *FileDrop*, which was originally written by Robert Harder and Nathan Blomquist and was released as public domain.

7.1.1 Testing

The testing of the system has been quite varied since a large part of the desktop client consists of a graphical interface. The graphical part of the client was tested throughout the developing process and the customers also had a part in testing the interface. Another difficult part of the testing was the communication with the server. A part of it was tested with JUnit tests but the larger part of the testing was made manually by interacting with the *GUI* and communicating manually with the server. A number of *JUnit* tests has been created concerning communication with server *API*.

Use cases of most functionality were put together to further formalize the testing of the implementation and to make sure the program is working as intended. These will also show all supported functionality in the system.

7.2 Web application

7.2.1 Frameworks

To ease implementation, a couple of frameworks have been used. The frameworks are described briefly below.

7.2.1.1 Backbone

Backbone[12] is a light-weight framework that loosely follows the MVC pattern. Out of the MVC components, Backbone only has models and views, and the view behaves much like a combination of both a view and a controller. `Models` are the parts of code that retrieve and populate data. `Views` are the HTML representation of models, and they change as models change.

Example 7 *When the `Experiment` model is populated, which may happen when the model fetches data from the server, it is immediately presented on the view that contains that experiment. The view itself does not have to manually get any data from the `Experiment` model.*

Backbone makes use of `Events`, where other objects can trigger events and listen to them, which is an effective way to promote decoupling between components. It also uses `Collections`, that are ordered sets of models. A collection will automatically be provided with underscore array and collection methods for convenient set manipulations (you can, for example, loop through a collection with `.each()` instead of writing a for-loop). Backbone is used because it allows more structure in the web application. With more structure, it is easier to collaborate as the work can be divided - keeping the Javascript code in various model, collection and view files.

7.2.1.2 Bootstrap

Bootstrap[13] is a front-end framework that contains HTML and CSS-based design templates for typography, buttons, forms, navigations, and the like. Instead of creating buttons from scratch, deciding on colors, how big they are, and micromanaging how they fit with everything else on the page, bootstraps templates that handles all of that, leaving the developers able to focus on architecture. Bootstrap is used to save time on development and make the design of the web app easily customizable.

7.2.1.3 RequireJS

RequireJS[16] is a file and module loader for Javascript. RequireJS lets files require other files much like `#include` in Java. This is very handy for the programmer. It is used because it helps to structure the application.

7.2.1.4 JQuery

JQuery[17] is a popular code library that handles AJAX calls and DOM manipulation, and makes the code more compact and readable.

7.2.2 Technologies used

A couple of technologies have been used in the development and are described below.

7.2.2.1 AJAX

AJAX[14] (Asynchronous Javascript and XML) is a technique for creating fast and dynamic web pages. Despite the name, the use of XML is not required; `JSON` is often used instead, which is the case in the Genomizer web app. AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server, so that you only update parts of a webpage without having to reload the entire page (like websites that do not use AJAX have to). For example, when the search button is clicked in the navigation bar of the web app, only the bottom half of the website is updated, and displaying the search view. The navigation bar does not have to be reloaded, but remains as it is on top.

7.2.2.2 `JSON`

`JSON`[15] (Javascript Object Notation) is a format that is primarily used to transmit data between a server and a web application instead of using XML or other formats. `JSON` is formatted as easily readable text consisting of attribute-value pairs. `JSON` was used in this application because `JSON` uses the same syntax as Javascript and therefore no parsing is needed, as opposed to the usage of e.g. XML. `JSON` also works well together with Backbone as it has integrated methods using the `JSON` format.

7.2.3 Testing frameworks

Three libraries are used to make testing easier: Chai, Mocha and Sinon. Together they let the developers make a page for testing where all tests and results will be shown visually. These libraries or testing frameworks will be discussed below.

7.2.3.1 Chai & Mocha

Mocha[19] is a test framework while Chai[18] is an expectation framework. While Mocha setups and describes test suites, Chai provides convenient helpers to perform all kinds of assertions against Javascript code. We use these frameworks to do unit testing on our models and collections.

7.2.3.2 Sinon

Sinon[20] is a framework used to “fake environment”. When doing unit testing, we do not want to depend on things that are external to the unit of code that we are testing. Sinon can be used for stubbing and mocking external dependencies and to keep control on side effects against them. For example, Sinon can be used to create spies to see if an event has been triggered, and to create fake servers that respond with fake pre-planned responses to queries.

7.2.4 Web app tests

Unit tests have been performed on all model and collection files that contain non-trivial functions. All unit tests can be found in the root folder under `/tests/`, more specifically `/genomizer-web/tests/`. To run the tests, simply open the index.html in a web browser and they will run. The views have not been unit tested since that is overly complicated; instead they have been continuously manually tested throughout the development process. In addition to these simple development tests, more official system tests have also been done by the desktop group.

7.3 Android application

The Android application has been implemented with `Java`, the default language for developing Android applications. The framework is called Android Application Framework and consist of view managers, resources and more. The application is developed for devices with, reaching over 90% of all devices (May 2015).

7.3.1 Environment

The application has been developed in Eclipse suited with Android Software Development Kit (SDK). The SDK consist of debuggers, libraries etc.

7.3.2 Emulation

The application has been emulated on and Android Virtual Device (AVD). This emulator comes bundled with the SDK and is a great tool simplifying Android development.

7.3.3 Android Support Library

The Android Support Library has been used in order to enable functionality introduced in later API levels to earlier API levels.

7.3.4 Technologies

The communication with the server is done by `HTTP` and `REST`, exchanging `JSON`-packages.

7.3.5 Testing

The model classes of the application has been tested with `JUnit 3`. Since the application is so heavy on graphical interaction (wich is hard to test) a lot of manual testing has been done, both on AVD and on real hardware devices. The acceptance tests that is stated in section K.2 and applicable to a mobile application have all been satisfied.

7.4 iOS application

The iOS application has been implemented using Objective C. The decision to use Objective C was made largely because it is the standard language used for writing iOS applications. Libraries and framework that has been used is mostly Apple's standard framework for iOS applications called UIKit. A few custom userinterface objects has been created to better suit our and our customers needs.

Communication with the server is done with a HTTP REST protocol and JSON objects are sent and received.

7.4.1 Testing

The logic in the application has been tested with XCTest which exists in the integrated development environment Xcode which has been user during the development. The graphical interface has both been tested throughout development both by the developers and by the customers. Furthermore a systemized testscript has been implemented using UI Automation which tests the application in the form of a user and runs through a set of scenarios. UI Automation is an instrument in Xcode which uses scripts written in JavaScript to simulate user interaction with the app.

7.5 Server

In this section the server and its different subsystem are displayed. Information about how the software design was realised in code will be provided.

7.5.1 Communication

This section explains implementation details of certain bits of the communication/control part of the system.

7.5.1.1 Doorman

The doorman is a class which greets all incoming connections and requests. The doorman creates a RequestHandler and a server context so that the requests can be correctly handled.

7.5.1.2 RequestHandler

The RequestHandler class checks the server context to determine what Command is to be created and what action to take. If for instance the HTTP header is **GET /experiment** a *GetExperimentCommand* is created. The command is afterwards validated and executed. Uploading and downloading is handled differently. Instead a downloadHandler/uploadHandler is used to handle the exchange.

7.5.1.3 Authorization

The communication between a client and the server is authorized by a user-unique token which is created when the user sends a login request. A token is created when a user has logged in successfully and the token is sent back to the user so that the user can thereafter use this in future requests. The token created when a user sends a login is stored in the server memory until the user sends a logout request.

7.5.1.4 Removing inactive tokens

The server has a function which removes inactive tokens after a set limit of time. This is done because a client sometimes skips sending a logout request when shutting down the client program. The InactiveUuidsRemover class is used to achieve this goal. In a thread it sleeps for one hour before checking all clients. If any client hasn't sent a request for 24 hours, the client token is removed from the server memory.
This feature may be turned of with the flag "-nri".

7.5.1.5 Command object

The command object represent a specific command. It is created from the RESTful header and/or the `JSON`/ body sent from the client. The `JSON`/ API provides methods for automatic parsing of the `JSON`/ body into an object. The fields in the command object created must match the attributes in the `JSON`/ body. This match is case sensitive

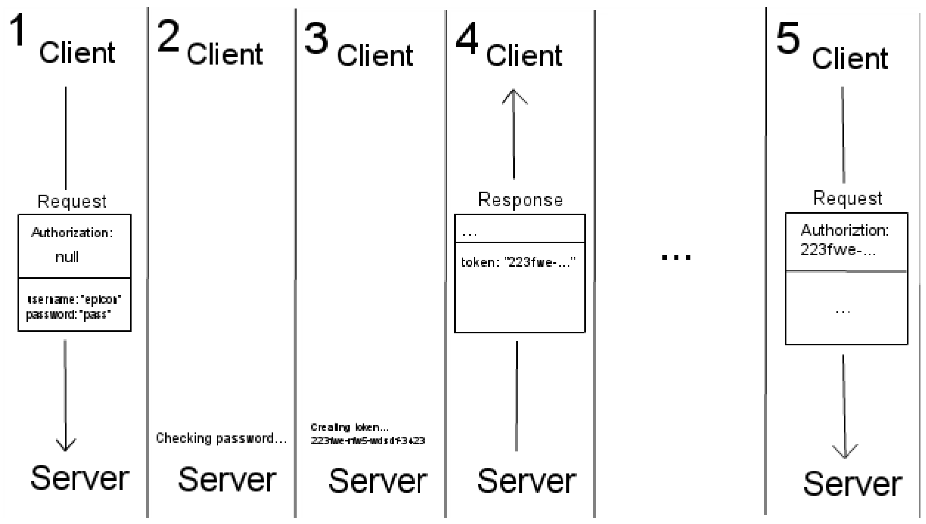


Figure 7.1: 1. The user sends a login request without any authorization token. 2. The server checks the given password. 3. The server creates a unique token for the user. 4. Server sends the token back to the user in a response. 5. Now that the user has a unique token, the token is placed in the header whenever the user sends another request.

7.5.1.5.1 Execute

Every command object must implement a execute method. This method is the part of the command which uses the system interface to perform the task that corresponds to the command.

The execute method returns a response object which is sent up to the RequestHandler which then sends the response to the client.

7.5.1.5.2 Validation

Every command must implement a validate method. This method is run after the command is created but before the command is executed.

The validate method throws ValidateExceptions if any information given by the client is in an incorrect format. The validation is used to prevent unnecessary communication.

7.5.1.6 Processing handling

For processing of data for instance multiple instances of a raw to profile convertion a queue is used. A client sends a list of processes to run by the command PutProcessCommands. The list of processes is placed in a ProcessPool which can start and cancel processing. All lists of processes that are placed in the queue are executed one at a time in the order first in first out. If a certain list of processes seems to take too long and is not prioritized the list can be cancelled through the CancelProcessCommand. Note that one single process in the list can not be individually cancelled but the entire list inputed through that particular PutProcessCommands can only be cancelled.

7.5.1.7 Response object

There are different response objects for different kind of responses since the form of the response to the client depends on the command the client initially sent.

The response object contains all the data necessary to create a RESTful header and a `JSON/` body for the response.

7.5.1.8 Testing

Testing has been done in multiple steps. The first step is unit testing, where individual methods are tested. This is often difficult due to the fact that the responsibility of handling client requests is shared by multiple classes. To catch these test cases a client dummy has been frequently used, which is the next step. It simulates a client by sending HTTP requests and examines the response from the server. It is used manually to test a particular use case, and to see that the server behaves as intended for that request. After a feature has passed the client dummy it is pushed to a test server, where it is open for other clients to test and debug. If no bugs are found the feature is declared complete and can be released.

7.5.1.9 Travis

To make sure of continous integration Travis is used with the GitHub project. Whenever a change is pushed to a branch of the project Travis will try to build the project and run the JUnit tests. No branches are allowed to be merged with others unless the Travis tests succeeded.

7.5.2 Conversion

This section will explain the implementation of the *SmoothingAndStep* subroutine used in the conversion of files from *raw* to *profile*. The basic algorithm is a dynamic arrayList which carries the rows that are relevant at a given time, smoothing on the first row is performed. The newly smoothed value is shifted out and replaced with a fresh row. This becomes a dynamic window that traverses the entire file one row at a time.

7.5.2.1 Methods

- smoothing : The one public method of the class. It controls the whole process and calls the other methods. It takes in the following parameters:
  - int[ ] params: An array with 5 integers representing parameters. params[0]: Window Size, the number of signal values that the smoothing should be calculated on.
    params[1]: Whether the smoothing should be trimmed mean (0) or median (1)
    params[2]: Minimum numbers to smooth. A number that says how many signal values the program at least need in order to smooth one row. This number must be smaller than windowSize.
    params[3]: Can either be 1 or 0. If 1 the program will calculate the total mean value for all rows and print those.
    params[4]: Print zeroes. If the program should print rows where the signal value is 0 the flag should be (1), if (0) the program will not print the zeroes.

  - String inPath: A filepath to the source file.
  - String outPath: A filepath to either an existing file to be overwritten or of a location and name that will become the path to a newly created file.
  - int stepSize: An integer larger than 0 that tells if there should be stepping. No stepping will be done if the number is 1.

  The method will also return the total mean of every row in the file if that flag is set properly.

- smoothOneRow: Checks whether smoothing should be trimmed mean or median and calls the corresponding method, after this is done it calls the method that writes to the new file.
- smoothTrimmedMean: Extracts the first position from the data array and initiates it's value to min and max values. We do this because trimmed mean means that we should remove the largest and smallest number from the mean value in order to get a more reliable/stable result. We then check that we have more numbers in the data array than the minimum numbers to smooth number. In order to avoid doing unnecessary calculations.
- smoothMedian: This method tries to fill an array with window size number of signal values and then pass this array to a method that finds which number is the median.
- writeToFile: This method does three different things. It check whether we should print zeroes in the outfile. It also check whether the current position is divisible with stepSize to determine if the row should be written to the outfile or skipped. After these two checks it either writes the row to the new file or not.

  It also check whether we want to print the total mean of the whole file and/or if we should then it counts up the proper variables.
- shiftLeft: Removes the first row from the data array and adds one row to the end of it. It then checks whether the new row is of a different chromosome than the others, if so it calls the special method chromosomeChange.
- chromosomeChange: This method knows that the last element in the data array is a new chromosome. It then reads and smooths as many rows as it can before hitting the cutoff number (minimum number of rows to smooth). It then writes and removes these values from the data array as well. It's important to note that so far it doesn't add new values to the array. Afterwards the method tries to refill the array with the new chromosome until it has window size number of rows.

7.5.3 Data Storage

The following text describes the different classes the Genomizer server uses to communicate with the database and the file system. All the communication with the database happens through the `DatabaseAccessor`-class, which uses several helper classes that contain methods with the actual logic. These relationships are visualized in Figure 7.2.

7.5.3.1 DatabaseAccessor

`DatabaseAccessor` is a class that serves as an API for the *Genomizer* database - it handles all connections and queries to the database. To counteract data inconsistency the isolation level of the connection is set to *TRANSACTION_REPEATABLE_READ*. The methods simplify queries to the database by removing the need to write *SQL* in any other packages. For more details see the class diagram, Figure G.1, in Appendix G.

7.5.3.2 Containers

The classes in the `database.containers` package represents domain specific objects. The `Experiment` class contains an experiment's annotations and a list of files. The `Annotation`

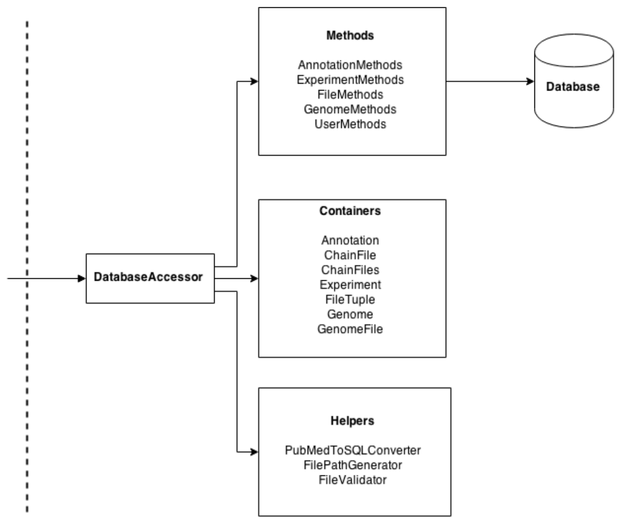


Figure 7.2: Conceptual overview - The implementation of data access and storage

class contains information about the annotation label, whether the label is required or not, the default value of the label and annotation choices for drop down values. The files are contained in `FileTuple` objects which stores filename, filetype, filesize, etc. More information can be found in the class diagram Figure G.2 in Appendix G.

7.5.3.3 Methods

The `DatabaseAccessor` uses helper classes to execute *SQL* queries. These helper classes are grouped by their area of responsibility:

- Annotation methods
- Experiment methods
- File methods
- Genome methods
- User methods

These classes execute the basic *SQL* functions create, read, update and delete (CRUD). The use of *prepared statements* in these classes (also known as parameterized queries) safeguards against *SQL* injection. When modifications to more than one relation are made from a single method, transactions are used to not put the database in an inconsistent state.

7.5.3.4 Helpers

Folders are created for a new experiment when the experiment is added to the database, including subfolders in preparation of files to be uploaded. The creation of folders is handled by the `FilePathGenerator` class. All files are then sorted into folders corresponding to experiment ID, except for genome release files that are categorized by species.

The `PubMedToSQLConverter` class converts a *PubMed* string to a *SQL* query. The simplest format for a *PubMed* string is *value[Label]* but the user can enter the annotation labels and values in combination with the the logical operators `AND`, `OR` and `NOT`. Parentheses are used for disambiguation.

> **Example 8** *To search for all raw files that Per created the PubMed string should be:*
>
> *raw[FileType] AND Per[Author]*

Searches can be made on labels corresponding to file attributes as well as experiment ID. Searching for empty string will return all experiments in the database.

7.5.3.5 Testing

WARNING! The unit tests in the `database.test` package have a database dependency. Do not run any of the tests found in this package on a database that is in use. All tuples are removed from the database upon test completion. Every instance of unit testing should start with an empty database and finish with an empty database to avoid test interdependency.

All the unit tests use the JUnit package and utilize the `TestInitializer` class. This simplifies the process of connecting to the test database, filling it with test tuples and clearing the test database and closing the connection when the test class is finished. The individual unit tests can be found in the `database.test.unittests` package. The scripts for adding the test tuples and clearing the test database tables can be found in the `sql` package.

7.6 Limitations

The *Genomizer* system has some limitations and known problems that needs to be mitigated. In Appendix M a detailed description of known problems for each of the *Genomizer* subsystems are listed.

Bibliography

[1] Langmead, Ben and Trapnell, Cole and Pop, Mihai and Salzberg, Steven L and others. *Ultrafast and memory-efficient alignment of short DNA sequences to the human genome.* Genome Biol 10(3). 2009.

[2] Zhan, Xiaowei. ”LiftOver”. *Center For Statistical Genetics.* February 14, 2014. Web. May 30, 2014. <`http://genome.sph.umich.edu/wiki/LiftOver`>

[3] Norris, David. ”Integrated Genome Browser”. *BioViz.* Web. May 31, 2014. <`http://bioviz.org/igb/`>

[4] *technoweenie.* ”Release Your Software”. *GitHub.* July 2, 2013. Web. May 29, 2014. <`https://github.com/blog/1547-release-your-software`>

[5] Preston-Werner, Tom. ”Semantic Versioning 2.0.0”. Web. May 29, 2014. <`http://semver.org/`>

[6] *National Center for Biotechnology Information.* ”PubMed Advanced Search Builder”. *U.S. National Library of Medicine.* October 28, 2009. Web. May 29, 2014. <`http://www.ncbi.nlm.nih.gov/pubmed/advanced`>

[7] *Genome Bioinformatics Group.* ”Frequently Asked Questions: Data File Formats”. *Genome Bioinformatics Group, UC Santa Cruz.* Web. June 2, 2015. <`http://genome.ucsc.edu/FAQ/FAQformat.html`>

[8] ”Authentication and Authorization”. *The Apache Software Foundation.* Web. May 21, 2014. <`http://httpd.apache.org/docs/2.2/howto/auth.html`>

[9] Dudler, Roger. ”git - the simple guide”. Web. May 29, 2014. <`http://rogerdudler.github.io/git-guide/`>

[10] Davis, Adam. ”Git for beginners: The definitive practical guide”. *Stackoverflow.* May 21, 2012. Web. May 29, 2014. <`http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide`>

[11] ”Generating SSH Keys”. *Github.* May 16, 2014. Web. May 29, 2014. <`https://help.github.com/articles/generating-ssh-keys`>

[12] Backbone.js documentation: `http://backbonejs.org/` Retrieved 8/5 -14

[13] Bootstrap documentation: `http://getbootstrap.com/` Retrieved 8/5 -14

[14] AJAX on wiki: `http://en.wikipedia.org/wiki/Ajax_(programming)` Retrieved 8/5 -14

[15] JSON on wiki: `http://en.wikipedia.org/wiki/JSON` Retrieved 8/5 -14

[16] RequireJS documentation: `http://requirejs.org/` Retrieved 8/5 -14

[17] JQuery documentation: `http://jquery.com/` Retrieved 8/5 -14

[18] Chai documentation: `http://chaijs.com/` Retrieved 9/5 -14

[19] Mocha documentation: `http://visionmedia.github.io/mocha/` Retrieved 9/5 -14

[20] Sinon documentation: `http://sinonjs.org/` Retrieved 9/5 -14

[21] ”List of UCSC genome releases”. *UCSC.* Web. May 30, 2014. <`https://genome.ucsc.edu/FAQ/FAQreleases.html`>

Nomenclature

Bowtie	Program the preforms parsing of the raw data and counts base pairs
Chain Files	Genome release files with small alterations to previous genome releases.
Genome Releases	Constant research gives more understanding, new genome versions are often found.
Genomizer	Collective name for the project.
GEO	Centralized database where article data can be found.
LiftOver	Converts genome release versions.
Profile	Data converted to a human readable file for analysis.
Raw	Collection word for files that are the result from a *DNA*-sequencing machine.
Region	Region data is small parts of the profile data.

A User manual

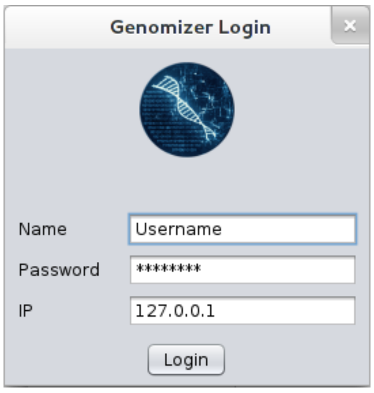
This chapter explains how you use each of the *Genomizer* clients. First instructions on how to use the desktop and the web clients are presented. These are the clients which provide the most functionality. The mobile clients are more lightweight and offer a subset of the functionality presented by the desktop client. Instructions on using the smartphone applications for *Android* and *iOS* are presented in their own sections at the end of the chapter.

A.1 Desktop application

This is a user manual for the desktop client. It will provide guides on how to use the client and the different functionalities it holds. The screen shots shown in this document are made from a Linux machine, but the application also runs on Windows or Mac, and will follow the design principles thereafter. Because of this, some details of the look of the client may vary, but the functionality is the same.

A.1.1 Login and startup

When you start this application the first thing that's displayed is a login screen, as illustrated in Figure A.1. In this screen you enter your username, password and the IP-Address for the server and then press the login button to enter the *Genomizer* Desktop.


Genomizer Login

Name: Username
Password: ********
IP: 127.0.0.1

Figure A.1: Screenshot of the login screen.

The application is built with tabs, as illustrated below in Figure A.2. Each tab contains separate features of the application. There are seven tabs: Search, Upload, Process, Workspace, Administration, Convert and Settings.

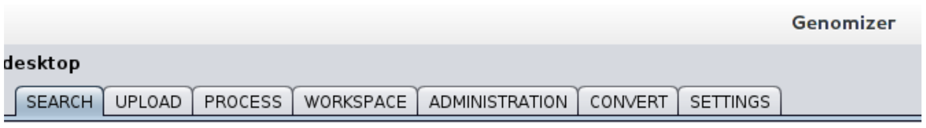


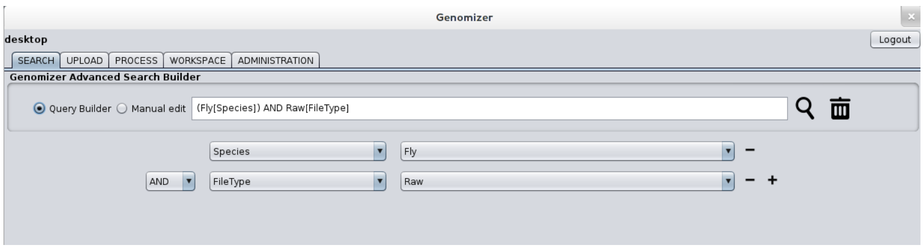
Figure A.2: Illustration of the different tabs of *Genomizer*Desktop and displaying the Search tab.

A.1.2 Search

The first tab you see after logging in is the *Search tab*, illustrated in Figure A.3. The Search tab uses the same query building technique as the *Pubmed Advanced Search Builder*[6]. It has one text field where you either can type in the query yourself or you can use the query builder to build the query.

To switch between manually editing the query and using the query builder there are two radio buttons to the left of the text field. Each row in the query builder has at most five components. These are a logical expression, an annotation name field, a free text field or a drop down menu to insert search words, a minus button and a plus button. The plus button is only available in the last row and it adda another row to the query. The minus button is used to remove a row and it exists on every row except if there is only one row in the query. The logical expressions combines the annotations, so they are available in every row but the first.

By writing in the annotation text field or selecting a value in the drop down menu you can specify the query the row will produce. Together each row builds a full query. As illustrated in Figure A.3 below.



[✓ Query Builder] Manual edit (Fly[Species]) AND Raw[FileType]

	Species	Fly
AND	FileType	Raw

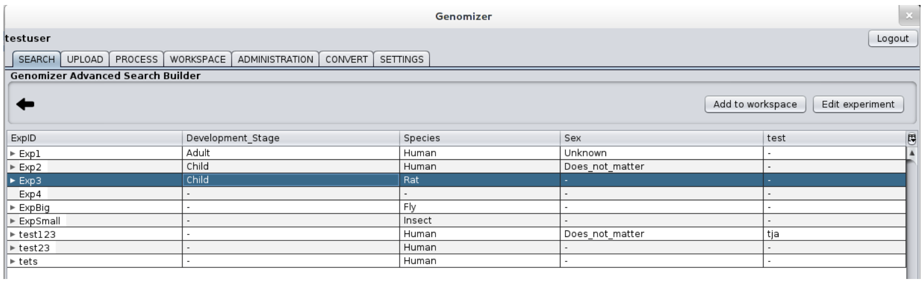
Figure A.3: Illustration of a query, made by the query builder.

A.1.2.1 Search results

When you press the search button the search tab will change it's view to display the search results as illustrated in Figure A.4. The results are displayed as experiments in a tree table. Each row is an experiment that can be expanded to show more information and the files associated with the experiment.

The tree table can be sorted both vertically or horizontally by clicking the headings or by dragging and dropping the columns. You can choose which columns to display by using the menu in the upper right corner of the table. In the same menu there are also buttons for expanding and collapsing all experiments in the search results.

To go back to the previous view, you can click the *Back* button. There is also a button called *Add to workspace* for adding the selected files or experiments to the workspace. The last button, *Edit experiment* is used to upload more files to an experiment or to edit information in the marked experiment.



ExpID	Development_Stage	Species	Sex	test
Exp1	Adult	Human	Unknown	-
Exp2	Child	Human	Does_not_matter	-
Exp3	Child	Rat	-	-
Exp4	-	-	-	-
ExpBig	-	Fly	-	-
ExpSmall	-	Insect	-	-
test123	-	Human	Does_not_matter	tja
test23	-	Human	-	-
tets	-	Human	-	-

Figure A.4: Illustration of search results.

A.1.3 Upload

If you need to upload files to the database it can be done through the upload tab. When the tab is pressed you get presented with a button to create a new experiment shown in figure A.5.

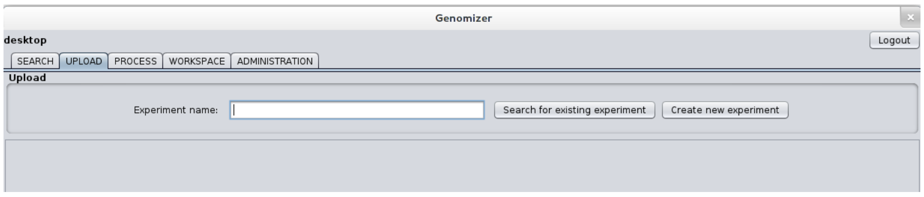


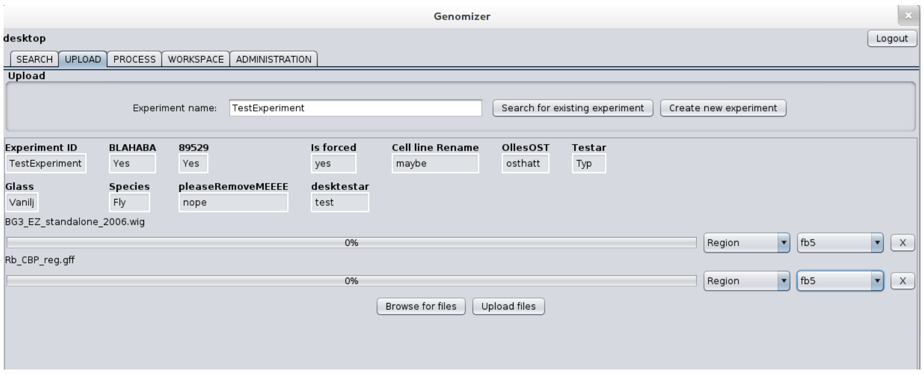
Figure A.5: Illustration of the starting view of the upload tab.

A.1.3.1 Existing experiment

In order to edit files or upload files to an existing experiment you need to search for the experiment in the search tab and then press the *Edit experiment*-button. When this is done the experiment information get retrieved from the server and presented to you.

To edit the experiment, change the annotation values and press the *Save changes*-button. To add files to the experiment you can press the button labeled *Browse files.* A file browser window will appear, it is illustrated in figure A.8. Here you can select the files you want to add to the experiment. The files will be added to the upload tab and there will be some new choices available for you. Each file will be associated with one file row, this is also shown in A.6.

The new choices are whether the new files are either raw, region or profile files. And if it is region or profile there is another choice for which genome release. There is also the possiblity to delete the file row, by clicking the *X*-button, in case this file is not suppose to be added to the experiment. When all fields have been filled and the files are correct, you simply click the button labeled *Upload files.* The progress bar will start to progress and if all goes well it will reach 100% and the files will be added to the existing experiment.



Experiment name: TestExperiment

Experiment ID	BLAHABA	89529	Is forced	Cell line Rename	OllesOST	Testar
TestExperiment	Yes	Yes	yes	maybe	osthatt	Typ

Glass	Species	pleaseRemoveMEEEE	desktestar
Vanilj	Fly	nope	test

BG3_EZ_standalone_2006.wig 0% Region fb5 X

Rb_CBP_reg.gff 0% Region fb5 X

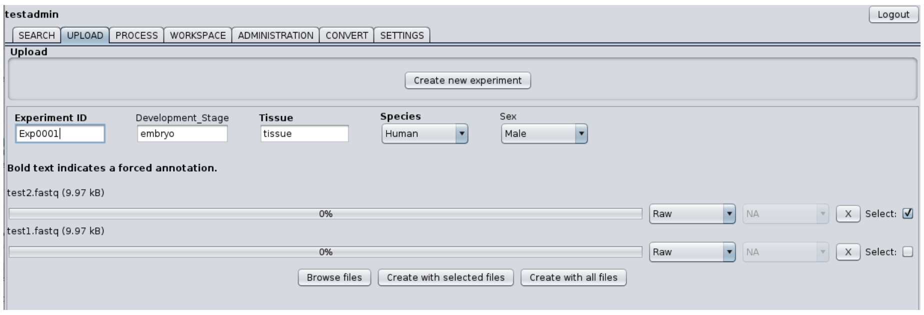
Figure A.6: Illustration of the add to existing experiment part of the upload tab.

A.1.3.2 New experiment

The first thing you need to do when creating a new experiment (A.7) is pressing the button labeled *Create new experiment* in the upload tab. After pressing this button all the different annotations get retrieved from the server. If the annotation is of freetext type there is a textfield to be filled out. If it is a multiple choice annotation there is a drop-down list of the different choices. The annotations who have bold text are forced and needs to be filled out in order to create the experiment.

In order to add files to this experiment you need to press the *Browse files*-button. A filebrowser will appear and you can choose which files to add. You can also drag and drop files from a folder on your machine onto the upload tab of the client. When the files are added they each get displayed in a file row. The file row consists of the file name and a progress bar.

Apart from the add experiments there are also three buttons and a checkbox. The checkbox will be explained in section A.1.3.3 below. The other three buttons are used in the same manner as in section A.1.3.1 above. When all the annotations that are needed is filled in and the associated files are added you press the button labeled *Create with all files* to create the experiment.



Experiment ID	Development_Stage	Tissue	Species	Sex
Exp0001	embryo	tissue	Human	Male

Bold text indicates a forced annotation.

test2.fastq (9.97 kB) 0% Raw NA X Select: ✓

test1.fastq (9.97 kB) 0% Raw NA X Select: ☐

Figure A.7: Illustration of the create new experiment part of the upload tab.

A.1.3.3 Batch upload experiments

The Genomizer desktop client has support for batch uploading of experiments. The following steps describes how to batch upload experiments:

1. Create a new experiment by pressing the button labeled *Create new experiment*.
2. Press the button labeled *Browse files* and select all the files to be included in the batch upload.
3. Fill in the experiment ID and annotations of the first experiment
4. Select one or more files to upload to the first experiment by ticking the check-boxes next to the files.
5. Select filetype and genome release if needed.
6. Press the button labeled *Create with selected fiels* and wait for the file(s) to be uploaded. When the upload is complete, the file row will disappear.
7. When the files have been uploaded, repeat steps 3-6 with the desired changes to annotations and experiment ID's.

Using this method let's the user make small changes to annotations while creating multiple experiments.

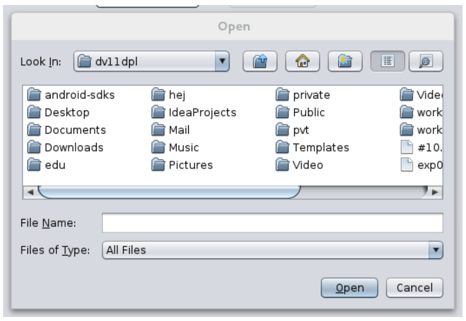


Figure A.8: Illustration of the file browsing window.

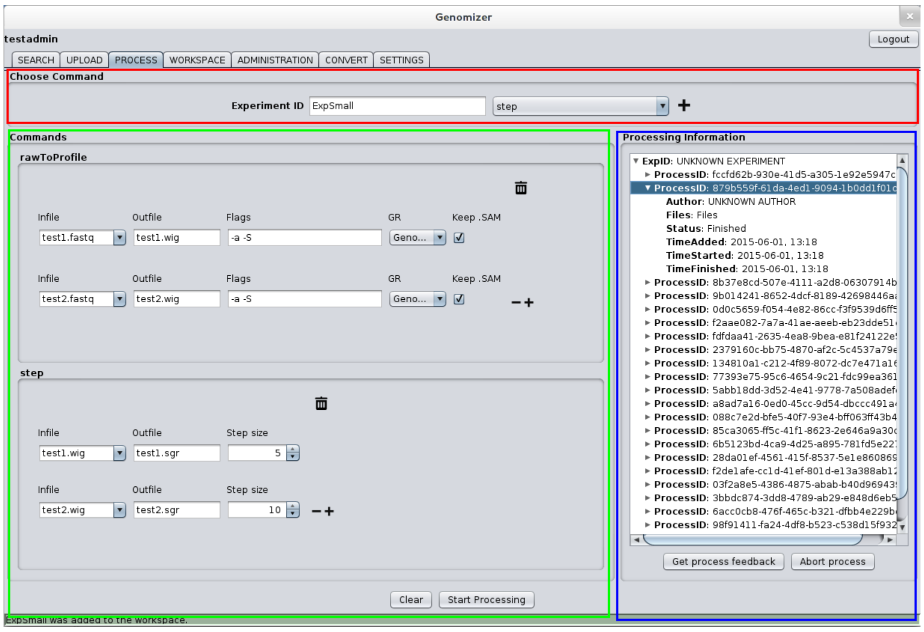
A.1.4 Process

From the *process view* (see Figure A.9) the user can process files uploaded to experiments. The view consists of three panels; one panel at the top with a drop-down list with process commands, one panel to the left that lists *command components* (see example in Figure A.10), and one to the right where feedback from processing tasks can be seen. The top-panel also has a button for adding commands to the list on the left panel.

To prepare an experiment for processing the following steps must be performed:

- Search for the desired experiment and add it to the workspace, as described under the search section.
- Select the experiment from the worksapce view and press the button labeled *Process*.

After performing these steps, the user can process the files of the selected experiment. The user can perform several process steps in a sequence, and also perform each step on several files. A command component in the list on the left panel represents a step in the processing sequence. Each command component has a row for input fields, where the user can select files to apply the command on, specify names of resulting files, and define other parameters.



Experiment ID: ExpSmall — step

rawToProfile

Infile	Outfile	Flags	GR	Keep .SAM
test1.fastq	test1.wig	-a -S	Geno...	✓
test2.fastq	test2.wig	-a -S	Geno...	✓

step

Infile	Outfile	Step size
test1.wig	test1.sgr	5
test2.wig	test2.sgr	10

```
▼ ExpID: UNKNOWN EXPERIMENT
    ▸ ProcessID: fccfd62b-930e-41d5-a305-1e92e5947c
    ▼ ProcessID: 879b559f-61da-4ed1-9094-1b0dd1f01c
        Author: UNKNOWN AUTHOR
        Files: Files
        Status: Finished
        TimeAdded: 2015-06-01, 13:18
        TimeStarted: 2015-06-01, 13:18
        TimeFinished: 2015-06-01, 13:18
    ▸ ProcessID: 8b37e8cd-507e-4111-a2d8-06307914b
    ▸ ProcessID: 9b014241-8652-4dcf-8189-42698446a
    ▸ ProcessID: 0d0c5659-f054-4e82-86cc-f3f9539d6ff5
    ▸ ProcessID: f2aae082-7a7a-41ae-aeeb-eb23dde51
    ▸ ProcessID: fdfdaa41-2635-4ea8-9bea-e81f24122e
    ▸ ProcessID: 2379160c-bb75-4870-af2c-5c4537a79e
    ▸ ProcessID: 134810a1-c212-4f89-8072-dc7e471a1
    ▸ ProcessID: 77393e75-95c6-4654-9c21-fdc99ea361
    ▸ ProcessID: 5abb18dd-3d52-4e41-9778-7a508adef
    ▸ ProcessID: a8ad7a16-0ed0-45cc-9d54-dbccc491a
    ▸ ProcessID: 088c7e2d-bfe5-40f7-93e4-bff063ff43b4
    ▸ ProcessID: 85ca3065-ff5c-41f1-8623-2e646a9a30
    ▸ ProcessID: 6b5123bd-4ca9-4d25-a895-781fd5e22
    ▸ ProcessID: 28da01ef-4561-415f-8537-5e1e86086
    ▸ ProcessID: f2de1afe-cc1d-41ef-801d-e13a388ab1
    ▸ ProcessID: 03f2a8e5-4386-4875-abab-b40d96943
    ▸ ProcessID: 3bbdc874-3dd8-4789-ab29-e848d6eb5
    ▸ ProcessID: 6acc0cb8-476f-465c-b321-dfbb4e229b
    ▸ ProcessID: 98f91411-fa24-4df8-b523-c538d15f932
```

ExpSmall was added to the workspace.

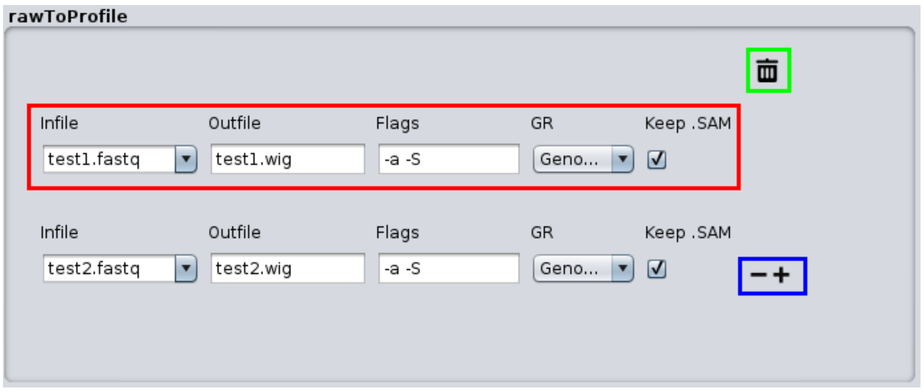
Figure A.9: The process view of the Genomizer desktop client. The top-panel (red) has a drop-down list and an add-button. The left panel (green) lists process commands to be performed in the processing sequence. The right panel (blue) shows the status of different processing tasks.

The command component also have buttons for adding and removing extra input-rows, and a button for removing the whole component.

The following describes how to process one or more files:

- Select a process command from the drop-down list located on the top-panel of the tab.
- Click the plus-button next to the drop-down list to add the command to the command list located below.
- In the command component in the list, fill in the needed parameters. Click the plus-button on the command component to add more files to process, or remove unwanted files by pressing the minus-button.
- To add another command to the sequence, repeat the above steps.
- To remove undesired commands, press the trash can icon in the upper-right corner of the command component. To remove all commands press the button labeled *Clear*.
- When the desired process commands have been added and parameters have been filled in, press the button labeled *Start Processing* to start processing.

The server will now start processing the specified files with the specified commands and parameters, in the same order the commands were added by the user. To get feedback from the processing task, the user can press the button labeled *Get process feedback*. To abort ongoing process tasks, the user can press the button labeled *Abort process*. The feedback panel lists experiments, which can be expanded to show the status of processing tasks, associated with the expanded experiment.



Infile	Outfile	Flags	GR	Keep .SAM
test1.fastq	test1.wig	-a -S	Geno...	✓
test2.fastq	test2.wig	-a -S	Geno...	✓

Figure A.10: A command component representing a raw-to-profile-process. The component has rows for input parameters (red), buttons to add and remove input rows (blue), and a button to remove the entire component (green).

A.1.5 Workspace

The workspace Tab seen in Figure A.11 is a tab where you can temporarily store experiments and their files, and choose different options for action. Results from various searches can be stored here, and the contents of the workspace is saved as long as the program is running. Files and/or experiments are chosen by clicking them, multiple files by using either Shift-click, Ctrl-click or simply holding down the mouse button and dragging the cursor over multiple files. By choosing an experiment, all of the containing files are selected.

A.1.5.1 Delete from workspace

Items can be deleted from the Workspace by pressing *Remove from workspace*.

A.1.5.2 Delete from database

To delete the selected data from the database the *Delete from database* button should be used instead. When pressing the delete button a small popup window with a progress bar will be displayed. By closing this window the deletion of data can be aborted.

A.1.5.3 Upload to

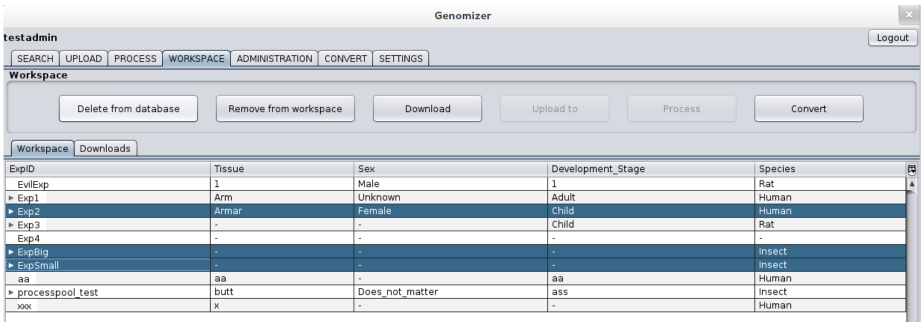
If you want to upload files to an experiment you have in the workspace, you can simply click the *Upload to* button to switch to the upload tab and upload to the experiment they have selected. If multiple experiments have been selected, only the first one will be uploaded to.

A.1.5.4 Process

If you want to add files to the process tab there is a *Process* button which transfers the selected experiment to the process tab.

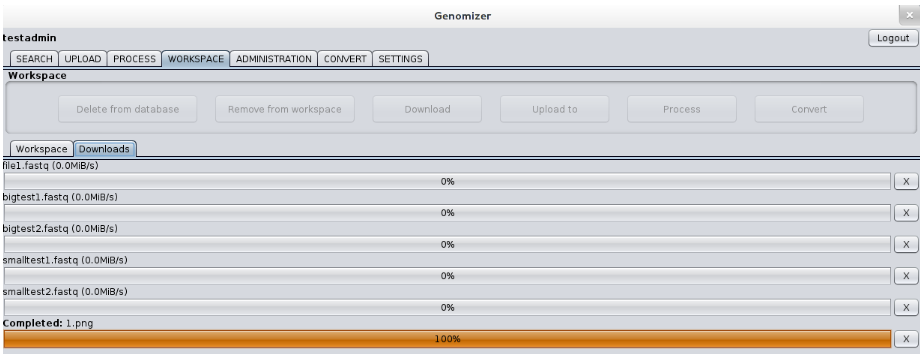
A.1.5.5 Download

You can make the choice to download files to their local computer. If you press the *Download* button seen in Figure A.11, you get to choose a directory where you want to save the files. When a directory has been chosen, the files get downloaded and all current and completed download can be seen in the tab *downloads*, see Figure A.12.



ExpID	Tissue	Sex	Development_Stage	Species
EvilExp	1	Male	1	Rat
Exp1	Arm	Unknown	Adult	Human
Exp2	Armar	Female	Child	Human
Exp3	-	-	Child	Rat
Exp4	-	-	-	-
ExpBig	-	-	-	Insect
ExpSmall	-	-	-	Insect
aa	aa	-	aa	Human
processpool_test	butt	Does_not_matter	ass	Insect
xxx	x	-	-	Human

Figure A.11: Screenshot of the workspace tab in the program.



```
file1.fastq (0.0MiB/s)
0%  X
bigtest1.fastq (0.0MiB/s)
0%  X
bigtest2.fastq (0.0MiB/s)
0%  X
smalltest1.fastq (0.0MiB/s)
0%  X
smalltest2.fastq (0.0MiB/s)
0%  X
Completed: 1.png
100%  X
```

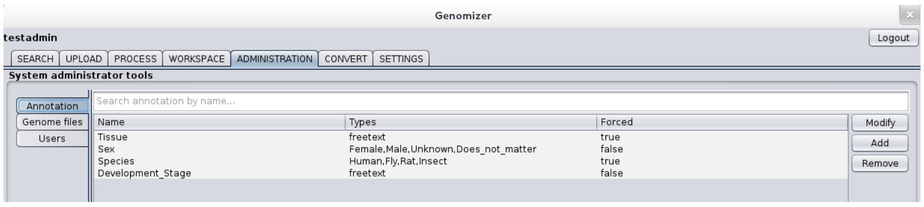
Figure A.12: The downloads tab of the workspace

A.1.6 Administration

A.1.6.1 Annotation

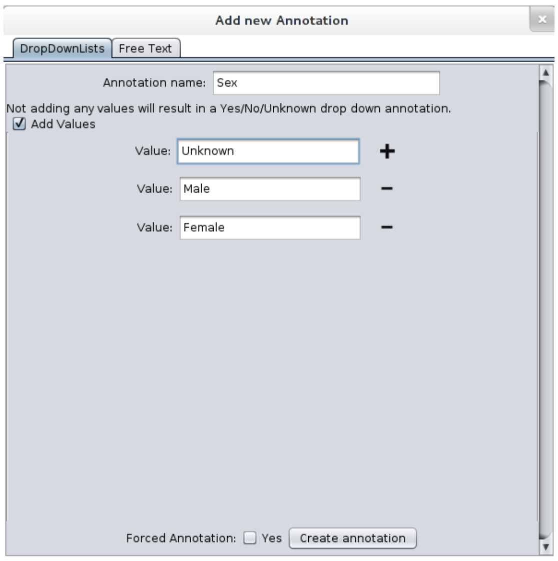
The system administration tools for the desktop client is available under the Administration tab. There are two different tools: Annotation and Genome files. The annotation tab is the first sub tab in the Administration tab. Annotations are used for specifying properties of uploaded data. For example, if new data from an experiment done with rat tissue is uploaded, the data shuld have an annotation called "species" with the value "rat". The Annotations sub tab in the Administration tab gives you the tools to create, edit and remove annotations and annotation values.

In the annotations tab, when you press the button labeled *Add* in the sidepanel a new popup window appears. It is possible to write the name of the new annotation and the new values in this popup, as well as check a "forced annotation"-box. The "forced" value determines if the annotation will have to be present in all future file uploads. See Figure A.14



Name	Types	Forced
Tissue	freetext	true
Sex	Female,Male,Unknown,Does_not_matter	false
Species	Human,Fly,Rat,Insect	true
Development_Stage	freetext	false

Figure A.13: The annotation view



Annotation name: Sex

Not adding any values will result in a Yes/No/Unknown drop down annotation.

☑ Add Values

Value: Unknown

Value: Male

Value: Female

Forced Annotation: ☐ Yes

Figure A.14: The add annotation popup

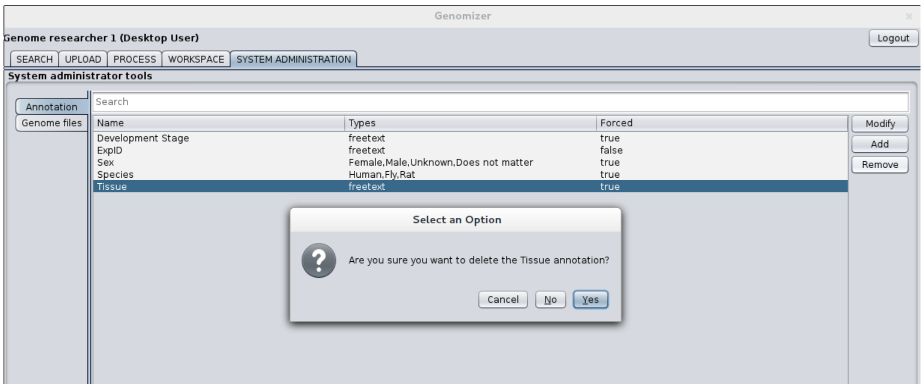
If you want to have free text as a value, for example if the annotation is pubmedID, the value of that annotation will not be able to be chosen from a drop-down menu, since the number available values is enormous. You might then want to use a freetext annotation, which allows you to type any value you want. To create a freetext annotation you click on the freetext tab on the "add" popup.

To remove an annotation, you select an annotation from the table in the center of the view, and click on the remove button on the right side. You then have to confirm this deletion. After this the annotation is completely removed and cannot be brought back to life, see Figure A.15. Some annotations cannot be removed for security reasons, 'Species' is such an annotation. Trying to remove it will generate an error message.

A.1.6.2 Genome files

The genome files tab shown in Figure A.16 contains a table with information about which genome release versions are stored on the server. If you click on one of the entries, a smaller frame is displayed at the bottom of the table showing which files are included in the selected genome release. To the right of the genome release tab are the tools for adding new genome releases. You can name the new genome release in the text field and you are then able to upload the files associated with that genome release.

When the desired files are selected, progress bars representing the upload of those files appear

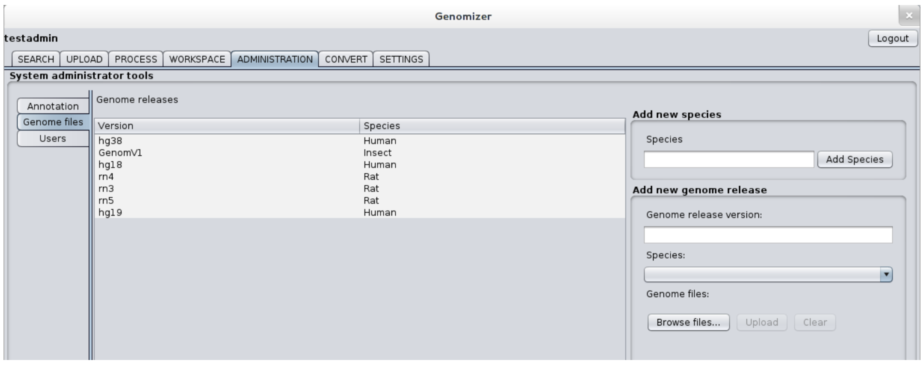


Name	Types	Forced
Development Stage	freetext	true
ExpID	freetext	false
Sex	Female,Male,Unknown,Does not matter	true
Species	Human,Fly,Rat	true
Tissue	freetext	true

Are you sure you want to delete the Tissue annotation?

Figure A.15: The remove annotation popup.

at the bottom of the *Add Genome Release*-frame. When you press the button labeled *Upload*, the upload of the selected files will start and you can follow the upload progress from the progress bars. After the upload is finished, you will be notified of its success or failure with a message dialog. Genome releases can also be removed by selecting the release version from the table and pressing the *Remove genome release*-button which appears at the bottom of the table when a release version is selected. This will remove the genome release and all associated files.



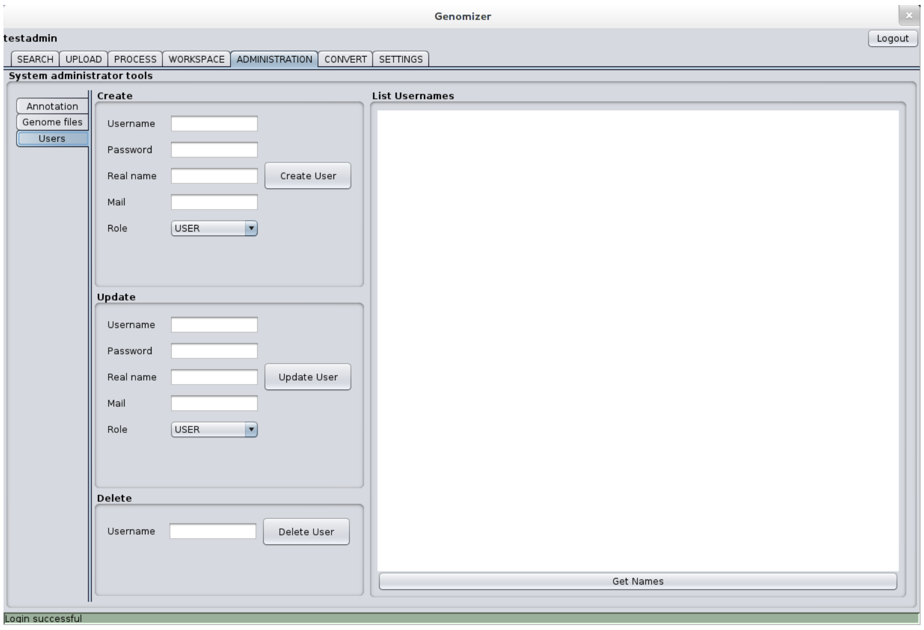
Version	Species
hg38	Human
GenomV1	Insect
hg18	Human
rn4	Rat
rn3	Rat
rn5	Rat
hg19	Human

Figure A.16: The genome release view.

If you want to add a new species to add or remove genome releases for, this can be done in the top right corner of the genome release tab. You simply writes the name of the new species and presses the *add*-button and the species will be added to the "Species" annotation.

A.1.6.3 Users

The Users tab shown in Figure A.17 contains four different panels, one to create a new user, one to update an existing user, one to delete an existing user and one to show all the users. Just fill in all information in the text fields and press one of the buttons to do that command. There is three different kind of users, the Guest, User and the admin. The Admin is the only one to have access to this User tab.



Login successful

Figure A.17: The Users admin view.

A.1.7 Convert

The Convert tab shown in Figure A.18 contains buttons to convert and remove selected files, and buttons to choose what to convert into. The panel *Convert from* shows the files sent into the convert tab, and the panel *Converted files* shows the finished files that have been converted.

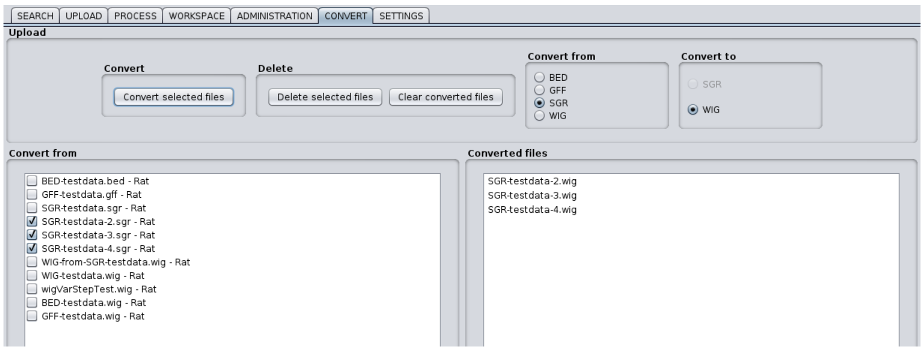
To convert files, begin with selecting the wished files to convert in the Figure A.11 and press the convert button. You will be directed to the convert tab with the files, check all the files you want to convert (only possible to convert one filetype at a time) and press the *Convert selected files*-button. If the files managed to convert successfully they will appear in the *Converted files*-panel to the right.

Notice that you need to choose which filetype you want to convert into in the *Convert to*-panel if the selected files can be converted into many different filetypes. The convert from panel is only in the panel for show, to show which filetypes you have selected.

Remove files from the list by pressing the *Remove selected files*-button, and clear the list with finished conversions by clicking on the *Clear converted files*- button.

A.1.8 Settings

In the settings tab Figure A.19 you can change password and information about your user. To change information, all fields must be filled. When the fields are filled, press the update settings.



Convert from: ( ) BED ( ) GFF (●) SGR ( ) WIG

Convert to: ( ) SGR (●) WIG

Convert from

- [ ] BED-testdata.bed - Rat
- [ ] GFF-testdata.gff - Rat
- [ ] SGR-testdata.sgr - Rat
- [x] SGR-testdata-2.sgr - Rat
- [x] SGR-testdata-3.sgr - Rat
- [x] SGR-testdata-4.sgr - Rat
- [ ] WIG-from-SGR-testdata.wig - Rat
- [ ] WIG-testdata.wig - Rat
- [ ] wigVarStepTest.wig - Rat
- [ ] BED-testdata.wig - Rat
- [ ] GFF-testdata.wig - Rat

Converted files

SGR-testdata-2.wig
SGR-testdata-3.wig
SGR-testdata-4.wig

Figure A.18: The Convert tab.

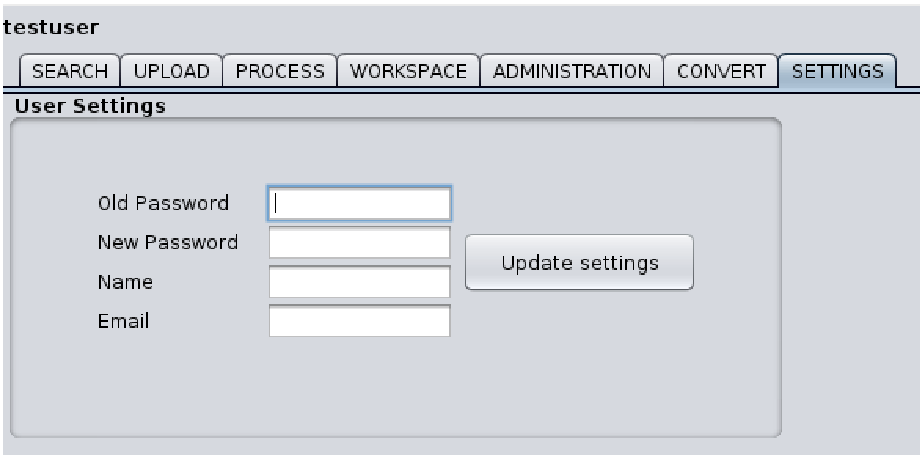


Figure A.19: The Settings tab.

A.2 Web application

To access the web application, navigate to a domain and directory that publicly serves the web page. An example of this could be: `https://scratchy.cs.umu.se:8000/app/`. All functionality of the web application is (or rather should be) fairly self-explanatory and intuitive. A short description and explanation will be given for each component that has been implemented so far.

A.2.1 Using the interface

This section describes how to use the interface and how to interact with it.

A.2.1.1 Start view

When first entering the web page, the login pop-up window in Figure A.20 is shown and the user will have to enter their username and password to gain access to the application.

When the user has logged in, the user is taken to the search page as shown in Figure A.21.

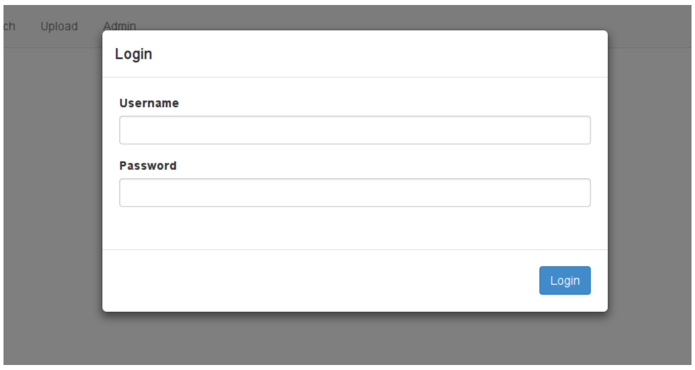


Figure A.20: The login pop-up window.

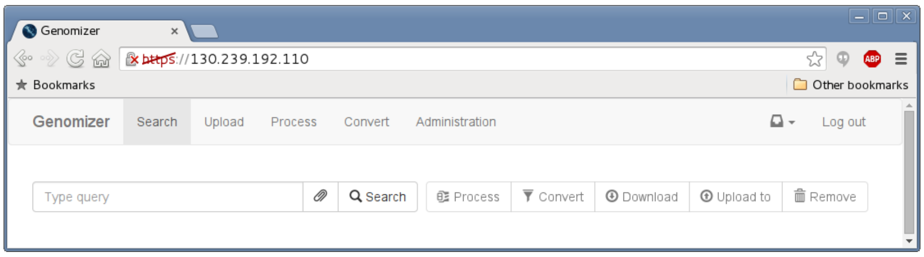


Figure A.21: The start view of the web page.

The navigation bar at the top has a number of buttons to the left and two buttons to the right with the following functionality:

- Clicking the "Genomizer" logo takes the user right back to the start view.
- The "Search" button will bring up the search view where the user can enter search strings to be sent to the server, and view search results.
- The "Upload" button will bring up the upload view where the user can select files to be uploaded and input annotation to a new experiment.
- The "Process" button will bring up the process view where the user can select an experiment to process.
- The "Administration" button will bring up the admin view where the user can handle genome releases and annotations.
- The inbox icon on the left side opens a dropdown list which displays the statuses of files currently being processed.
- The "Log out" button will log out the user.

This navigation bar is persistent through all sub pages and can easily be accessed.

A.2.1.2 Search view

In the search view, below the navigation bar, a "search-and-functionality" bar is visible. There is a search field and there are seven buttons that are explained below, starting with the left-most button:

- "Query builder", represented by a paperclip, brings up a query builder, shown in Figure A.22, that helps unexperienced users construct a valid query used for searching experiments. Just select a value in the three fields and press add. The correct pubmed-styled query will be shown in the search field and the three query fields will be reset so the user can add more things to search for in their query.

- “Search” searches for the query in the search field.
- “Process” processes the selected files.
- “Convert” converts the selected files. This feature is demonstrated further in section A.2.1.4.
- “Upload to” opens the upload view with the selected experiments selected where the user can upload new files to an already existing experiment.
- “Remove” opens a new view where the files which are going to be deleted are presented along with a confirmation dialog that the user really wants to delete those files and experiments.

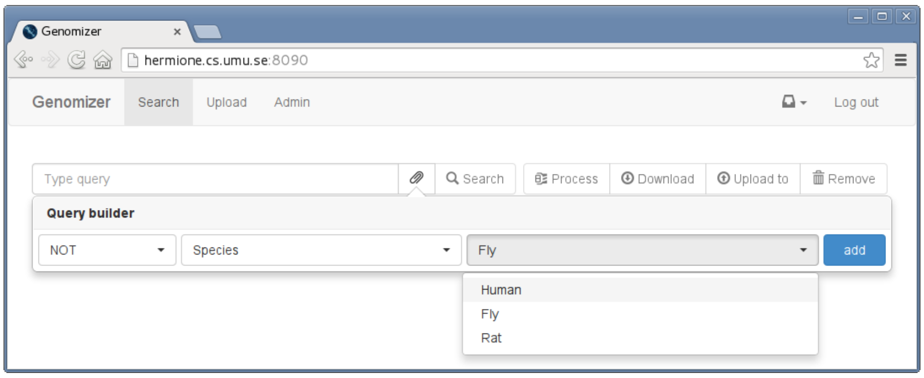
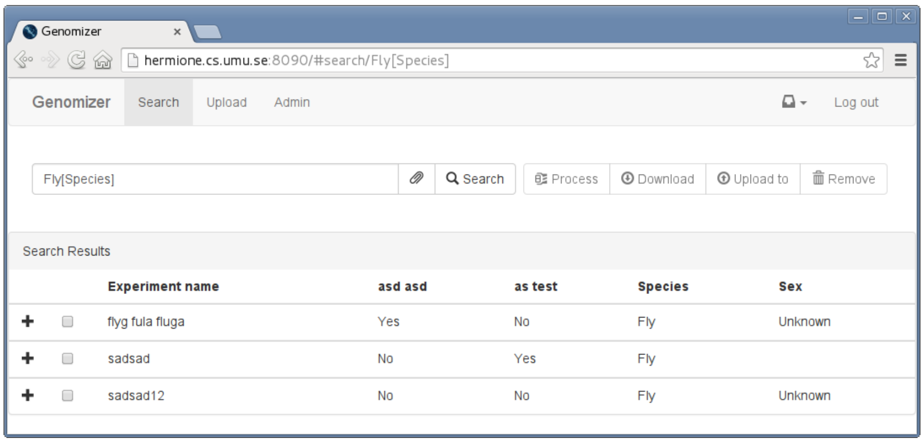


Figure A.22: The query builder.

When first entering the search view, only the query builder button and the search button are clickable. The rest of the buttons become clickable once the user selects experiments or files. To search, the user can either write a pubmed style query (for example: *”Fly[Species]”* to search for every experiment with ”Fly” as value of the annotation ”Species”) or use the query builder. When clicking the search button, a loading screen is shown. The experiment data is displayed once it has been retrieved from the server.

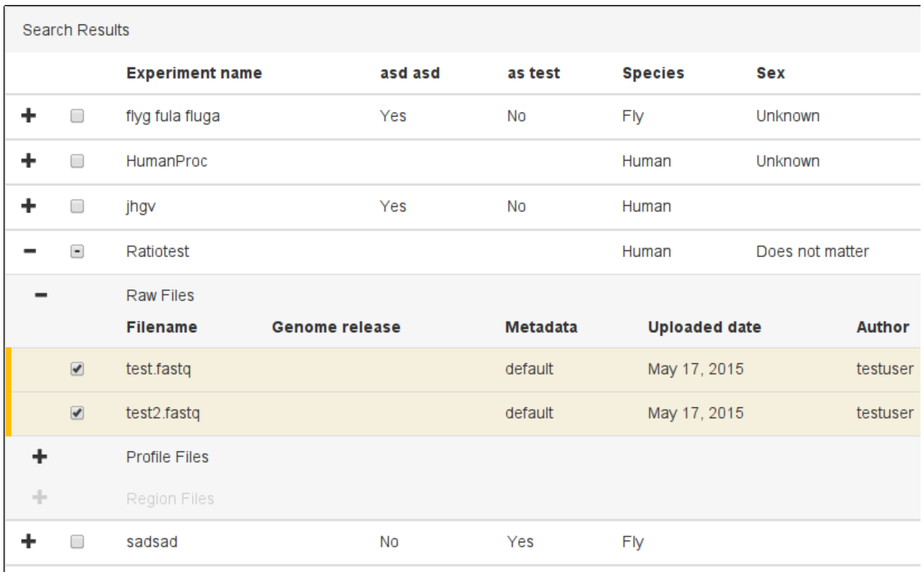


Search Results

		Experiment name	asd asd	as test	Species	Sex
✚	☐	flyg fula fluga	Yes	No	Fly	Unknown
✚	☐	sadsad	No	Yes	Fly	
✚	☐	sadsad12	No	No	Fly	Unknown

Figure A.23: The search tab after searching for *“Fly[Species]”*.

The view in Figure A.23 is shown when the user has searched for the query *”Fly[Species]”*. The displayed list contains all experiments returned from the search and a header on top with all annotation types. Every experiment can be expanded by clicking it to show the file types it contains. Each file type can be further expanded to show all files of that type in the experiment. Every file and experiment has a checkbox next to it that is used to select it. In Figure A.24, an experiment called `Ratiotest` and its contained collection of `raw` files have been expanded. Furthermore, the files `test.fastq` and `test2.fastq` have been selected. These files can now, for example, be processed or removed by using the buttons in the "search-and-functionality" bar.



Search Results

		Experiment name	asd asd	as test	Species	Sex
+	☐	flyg fula fluga	Yes	No	Fly	Unknown
+	☐	HumanProc			Human	Unknown
+	☐	jhgv	Yes	No	Human	
−	☐	Ratiotest			Human	Does not matter

− Raw Files

	Filename	Genome release	Metadata	Uploaded date	Author
☑	test.fastq		default	May 17, 2015	testuser
☑	test2.fastq		default	May 17, 2015	testuser

+ Profile Files

+ Region Files

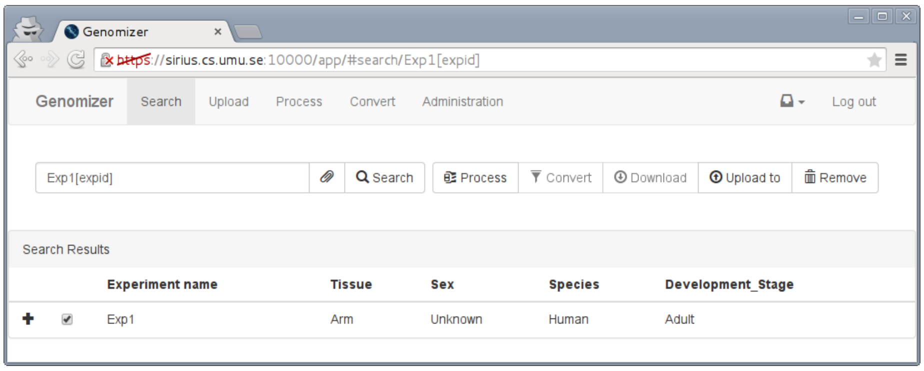
+	☐	sadsad	No	Yes	Fly	

Figure A.24: The search results table zoomed in, displaying the information of a `raw` file after having expanded an experiment.

If no experiments match the search query, the Search Results table will be empty stating "No search results found".

A.2.1.3 The processing view

If the user wants to process files from an experiment they first have to enter the search view and search for the experiment in which they wish to process files. Check the box for the experiment and then click the process button (Figure A.25). When the process view is



		Experiment name	Tissue	Sex	Species	Development_Stage
✚	✓	Exp1	Arm	Unknown	Human	Adult

Figure A.25: Selected Exp1.

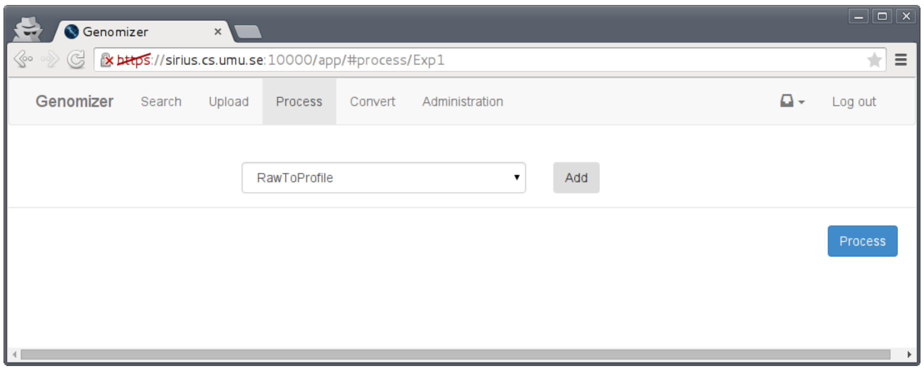


Figure A.26: The process view opened with Exp1 selected.

opened choose which process step you want to do in the dropdown list and then press the add button (Figure A.26). To add a new file to the selected process step press the ”+” button

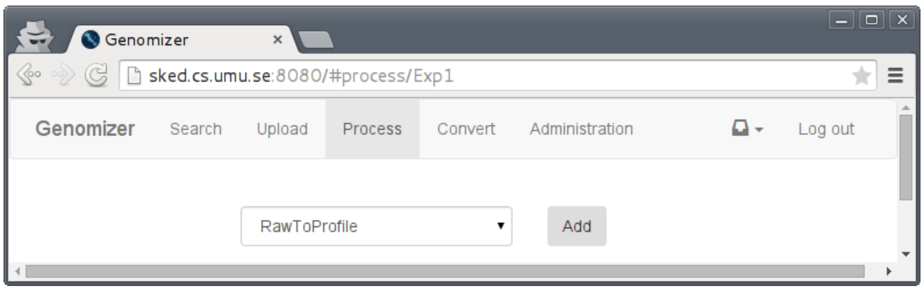


Figure A.27: Scrollbar which shows the different process steps.

(Figure A.27). After pressing the ”add” button the view will now show a new set of processing

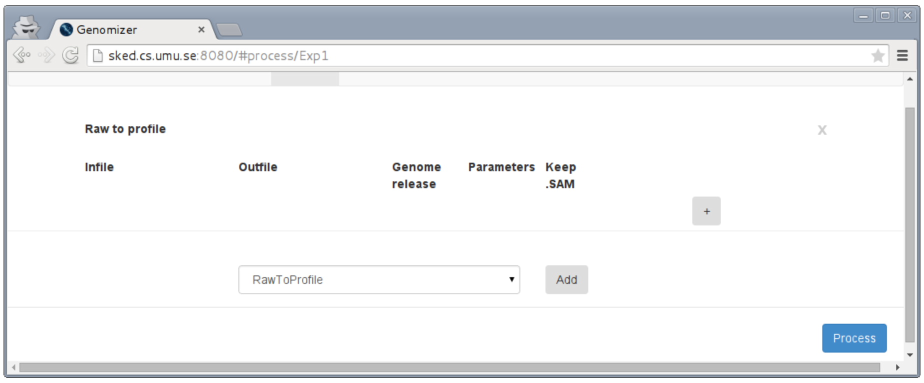


Figure A.28: Process view after a process step has been selected.

options. The ”Infile” dropdown list chooses which file is going to be processed. ”Outfile” is used to set the name for the new processed file. ”Genome release” sets which genome release to use for the process and ”Parameters” sets which parameters to be used. If the user wants the `.sam` file to be saved the ”Keep .SAM” box has to be checked (Figure A.28). To start

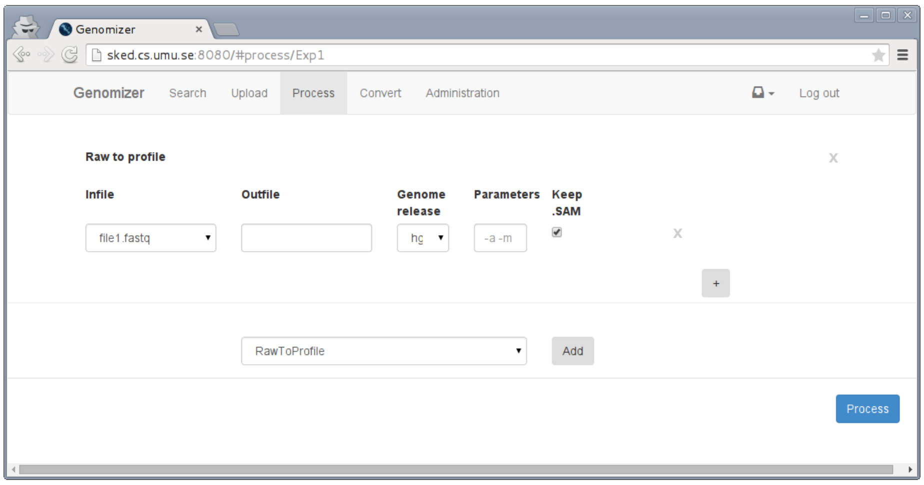
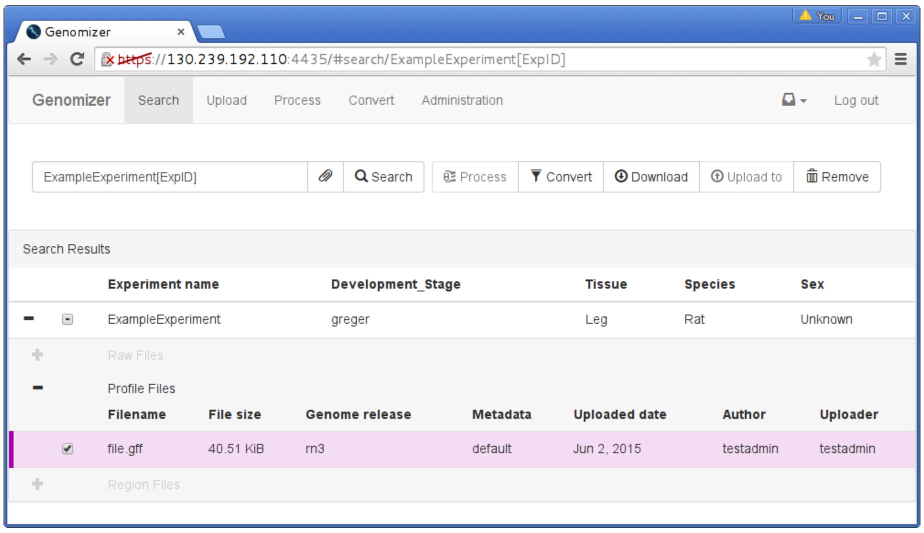


Figure A.29: Process view after adding a file for a process step.

processing simply press the ”Process” button (Figure A.29).

A.2.1.4 The convert view

The web application allows conversion between a number of file formats which both are of profile type. More specifically, the user may convert any of the file formats `.sgr`, `.wig`, `.bed` and `.gff` to either `.wig` or `.sgr`, with the exception that a file cannot be converted to the same file format.



ExampleExperiment[ExpID]

	Experiment name	Development_Stage	Tissue	Species	Sex
☐	ExampleExperiment	greger	Leg	Rat	Unknown

	Filename	File size	Genome release	Metadata	Uploaded date	Author	Uploader
✓	file.gff	40.51 KiB	rn3	default	Jun 2, 2015	testadmin	testadmin

Figure A.30: Selecting a file to convert.

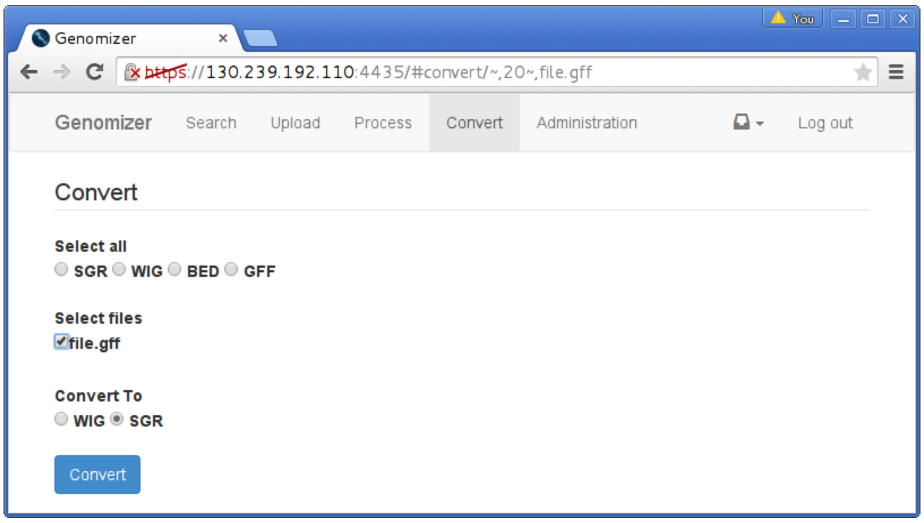
Assume that there is an experiment **ExampleExperiment** which contains a profile file **file.gff**. Then the experiment will show up in the search view, see Figure A.30, when typing "Example-Experiment[ExpID]" as search query and then clicking the search button. If the user wants to convert **file.gff** to a new file of format `.wig` called **file.wig**, the following steps can be taken:

- From the view in Figure A.30, select **file.gff** by clicking in its checkbox.
- Click the "Convert" button next to the search text field.
- The user will now be taken to the convert view shown in Figure A.31.
- Select **file.gff** by clicking on it.
- Mark the `.wig` checkbox in "Convert to" and click the "Convert" button.

The file conversion will now start on the server. Once the conversion is done, the user will be able to see **file.wig** listed together with the old file **file.gff** when searching for the experiment.

If multiple files are selected for conversion, all of them will appear as a list in the convert view. If the user quickly wants to select all files of a specific file format, for example all `.wig` files, the GFF option can be marked in "Select all". Then all files in the list of the format `.wig` will automatically be selected.

A.2.1.5 The remove pop-up window



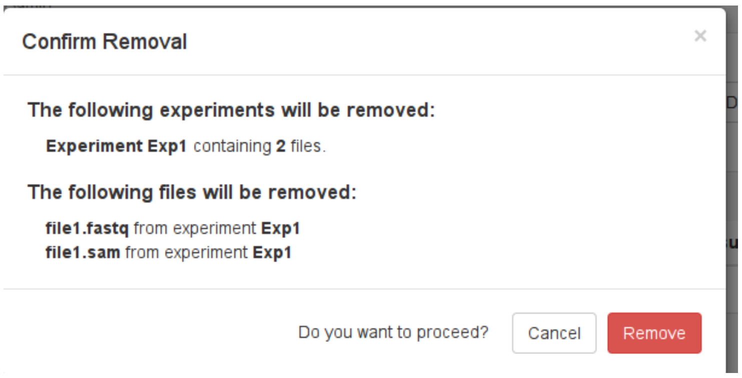
Convert

Select all
SGR WIG BED GFF

Select files
[✓ file.gff]

Convert To
WIG [✓ SGR]

Figure A.31: The convert view.



Confirm Removal

The following experiments will be removed:

Experiment Exp1 containing **2** files.

The following files will be removed:

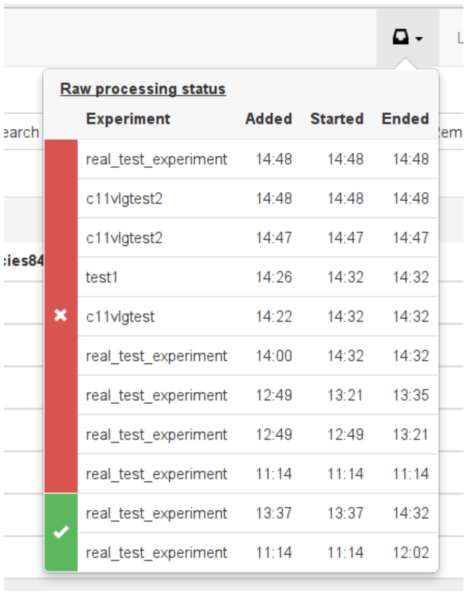
file1.fastq from experiment **Exp1**
file1.sam from experiment **Exp1**

Do you want to proceed?

Figure A.32: The remove pop-up window.

When the remove button is pressed the pop-up window in Figure A.32 is shown displaying which files and experiments will be removed when the remove button is pressed.

A.2.1.6 The process status dropdown



Raw processing status

	Experiment	Added	Started	Ended
🟥	real_test_experiment	14:48	14:48	14:48
🟥	c11vlgtest2	14:48	14:48	14:48
🟥	c11vlgtest2	14:47	14:47	14:47
🟥	test1	14:26	14:32	14:32
🟥	c11vlgtest	14:22	14:32	14:32
🟥	real_test_experiment	14:00	14:32	14:32
🟥	real_test_experiment	12:49	13:21	13:35
🟥	real_test_experiment	12:49	12:49	13:21
🟥	real_test_experiment	11:14	11:14	11:14
🟩	real_test_experiment	13:37	13:37	14:32
🟩	real_test_experiment	11:14	11:14	12:02

Figure A.33: The process status dropdown.

When pressing the inbox icon, a dropdown is shown as in Figure A.33 displaying the status of experiments currently being processed. There are four different statuses a processing can have, all grouped into colors: Waiting (yellow), Running (blue), Complete (green) and Failed (red). For example, in the figure, the two bottom experiments are complete and the rest have failed. If there are no experiments being processed, the dropdown will simply display "No process status available".

A.2.1.7 The upload view

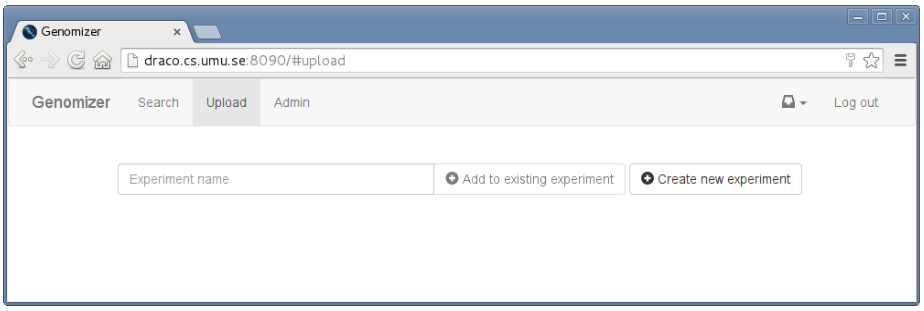


Figure A.34: The upload view.

When the user clicks the upload tab in the navigation bar, the view in Figure A.34 will appear. The user has the option to create a new and fresh experiment or to load an existing experiment by entering its experiment name.

After clicking the "Create new experiment" button, the view in Figure A.35 will appear. Here the user can input the annotations for the experiment through either freetext fields or dropdown lists. If a freetext field has a red border around it, that annotation is required and the experiment cannot be uploaded before all required fields have been filled in and at least

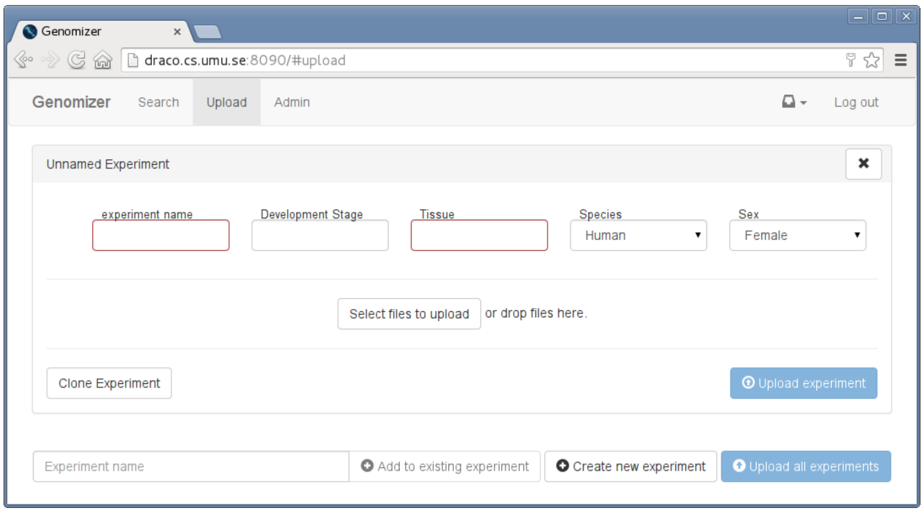


Figure A.35: Creating a new experiment.

one file has been added.

The user can create more experiments by clicking the “Create new experiment” button and a new empty experiment will be placed below the first experiment. The user can also clone an experiment by clicking the “Clone Experiment” button. What happens in this case, is that the every filled-in annotations gets copied to the new experiment.

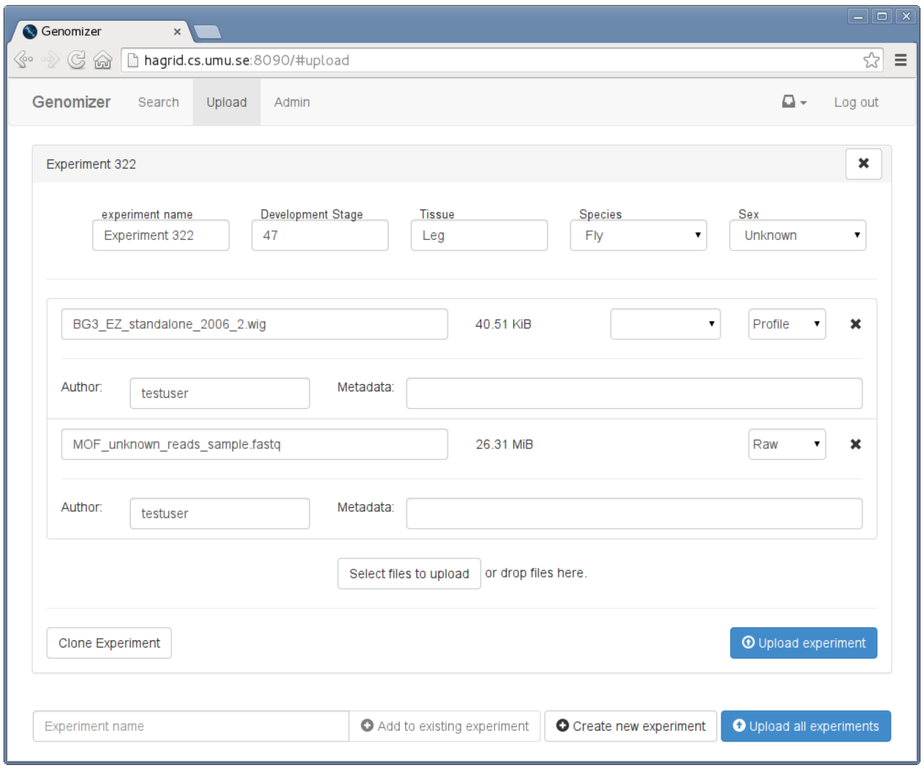
To add files to the experiments the user can browse for local files and upload them by clicking the “Select files to upload” button. The user will only see file types that have to do with experiments but have the ability to search for all file types. There is also a way of adding files to the experiment by dragging them from a file browser and dropping them onto the experiment “drag and drop”.

An experiment can only contain two `raw` files and if the user tries to upload more a message with this information will appear and the experiment cannot be uploaded before the extra `raw` file/s is removed.

To add files to a existing experiment the user types the name of the experiment in the field next to the “Upload to existing experiment” and clicks the button. If the experiment exists on the server it will appear in the experiment view the same way that a new experiment is shown. The annotations of an existing experiment cannot be changed from this view and if there are files already in this experiment they cannot be manipulated. Adding new files to existing experiments works the same way as to a new experiment.

When the user selects files, they will appear below the annotations as in Figure A.36. The file name is displayed in a text field on the left side of the file view. Next to the file name is a box that shows the size of the selected file. On the right side there is an option to select what type of file is being uploaded and an option to remove the file from the experiment. If the file type is either `profile` or `region`, there is an option to select what genome release the file is mapped to. The file type option will automatically be filled in with a guessed value depending on the file ending as follows: `.fastq` files are considered `raw` and all other formats (`.sgr, .wig, .gff`) are interpreted as `profile`.

When the user is done selecting files, filling in annotations and clicks the “Upload experiment” button the experiment view will be minimized showing only the name of the experiment and the progress bar of the files being uploaded. When the progress bar is done it turns green and now the experiment with all the files have been uploaded to the server. The user also has a way of uploading several experiments at the same time by clicking “Upload all experiments”.



Genomizer

hagrid.cs.umu.se:8090/#upload

Genomizer Search Upload Admin Log out

Experiment 322

experiment name	Development Stage	Tissue	Species	Sex
Experiment 322	47	Leg	Fly	Unknown

BG3_EZ_standalone_2006_2.wig 40.51 KiB Profile

Author: testuser Metadata:

MOF_unknown_reads_sample.fastq 26.31 MiB Raw

Author: testuser Metadata:

Select files to upload or drop files here.

Clone Experiment Upload experiment

Experiment name Add to existing experiment Create new experiment Upload all experiments

Figure A.36: Files selected for upload.

A.2.1.8 System administration view

This part of the web application is only accessible if the user has administrator rights. It is integrated with the rest of the web user interface and accessible through the “Administration” tab. The administrator can through this site see all annotations, add new annotations and edit existing ones.

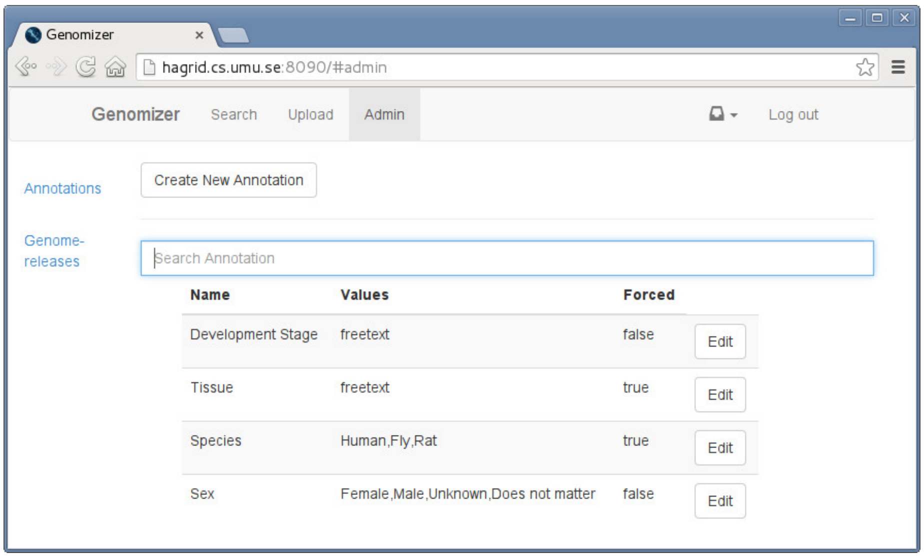
The start page of this section has a “Create New Annotation” button, a list of existing annotations in the database and an edit button per existing annotation. The view looks like in Figure A.37.

For each annotation in the annotations list, an “Edit” button is available. When pressed, it will take the user to a page in which they can edit the selected annotation to change its name and what values the dropdown list will have if it is not a freetext field (see Figure A.38).

In the edit page, the admin can see the attributes of the chosen annotation and is able to delete the chosen annotation or change the information of it. The “Delete Annotation” button will delete the whole annotation, and for that reason two pop-up windows will appear to confirm that the administrator is sure of the action.

The administrator can change the list of annotation values. The site will automatically check whether something is added, removed or both and sends a request to change the annotation values to the server when the “Update Annotation” button is clicked.

If the admin clicks on the “Create new annotation” button from the admin start page, another view will open with the following structure:



Name	Values	Forced	
Development Stage	freetext	false	Edit
Tissue	freetext	true	Edit
Species	Human,Fly,Rat	true	Edit
Sex	Female,Male,Unknown,Does not matter	false	Edit

Figure A.37: The start page for the administrator in the web client.

- Annotation Name
  Admin can enter a name for the annotation.
- Annotation Types
  Yes/No/Unknown - creates a dropdown list with those three options.
  freetext - creates an annotation where the users will be able to enter anything.
  Dropdown list - will enable a fourth field enabling the admin to enter which items that this list will contain.
- Forced Annotation
  Admin can choose if the new annotation should be required by users to enter.

A Create Annotation will, if all necessary information has been entered, result in a popup (see Figure A.39) showing the resulting annotation and if confirmed, the annotation is added to the database. If canceled the administrator can keep making changes or go back to exit this view. If not all values is entered the admin will be alerted of the mistake and nothing will be created.

The example in Figure A.39 will result in a drop-down annotation with the name Number of toes and possible values: 0, 1, 2, 3, 4, 5 with 0 as default and is not forced.

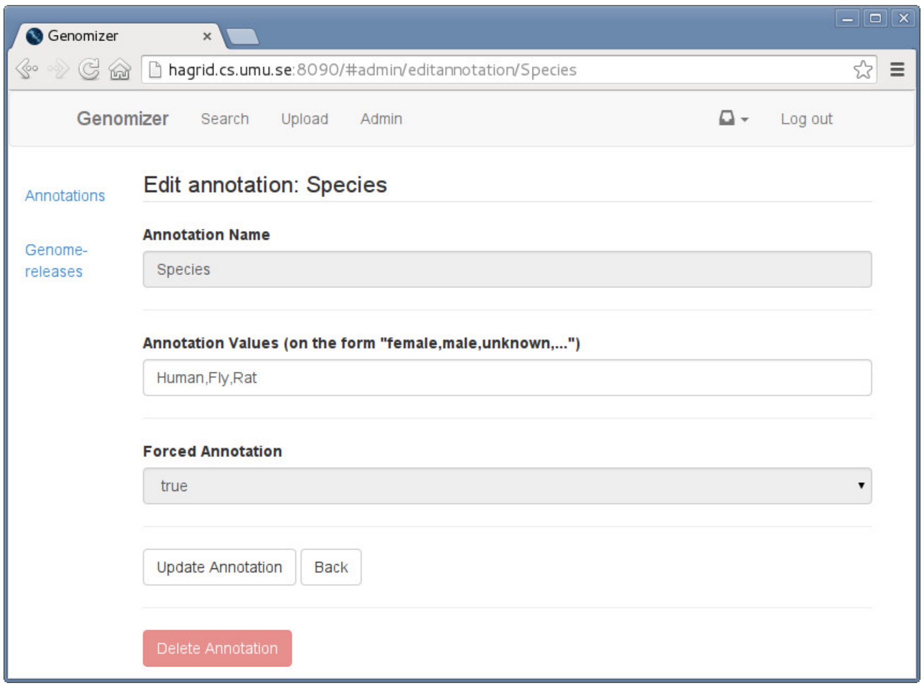
A back button which takes the user back to the annotations start page is also available in this view. In Figure A.40 the create annotation view can be seen.

The “Genome-releases” link on the sidebar takes the administrator to a page where it is possible to add and remove genome releases to and from the server (see Figure A.41.

The button “Select files to upload” opens the native file explorer where the user can select one ore multiple files and click on “OK”. This will open a popup-window, seen in Figure A.42, showing what files that where chosen and asks for species and genome version before uploading.

When the upload begins the popup closes and a progress-bar appears showing the progress, showing ”Upload completed” when done. The user can at this stage move between pages without disturbing the upload but should not close or refresh the web browser.

Every genome release in the table can be deleted by clicking on the “Delete” button next to the release. This will prompt a small popup asking for user confirmation and if given a positive response, deletes the genome release from the server and updates the view.



Edit annotation: Species

Annotation Name: Species

Annotation Values (on the form "female,male,unknown,..."): Human,Fly,Rat

Forced Annotation: true

Figure A.38: The edit annotation view.

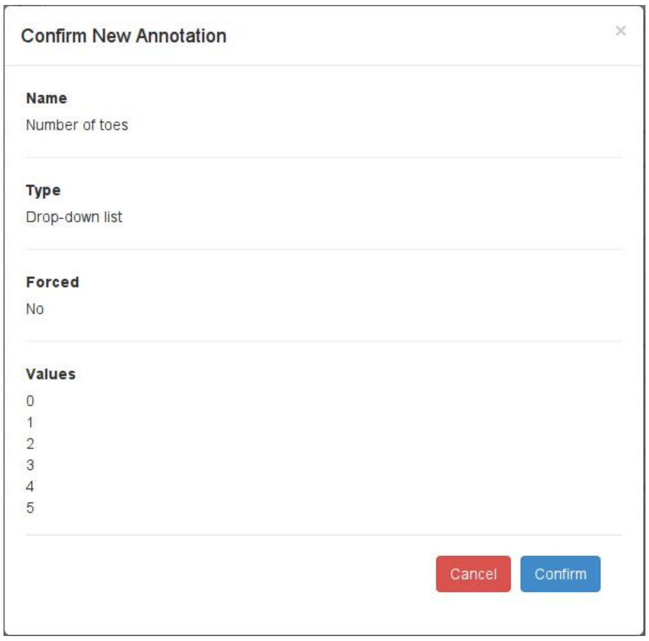
If any genome release is used by an experiment already an error will appear telling the user exactly that.

A.2.2 Setting up the application

To setup the application, move the content of the folder `genomizer-web/app/` to the desired location from where the application should be run. To run the web page, open a web browser and enter the url to the folder which contains the `index.html` file (where the content of app was placed). For example, given that the `genomizer-web` folder is placed in the home folder of the Umeå university CS user `c11abc` and that user wants to put the web app in a folder called `public_html/` which is also in the home folder of the user. In Linux, do the following steps:

1. Navigate to the app folder: `"cd ~/genomizer-web/app/"`
2. Move the contents of app to the folder `public_html`: `"mv * ~/public_html/"`
3. Given that the url to `public_html` is: `"www8.cs.umu.se/~c11abc/"`
4. To run the application start a web browser and type `"www8.cs.umu.se/~c11abc/"`

This will open the web page in the browser.


Confirm New Annotation

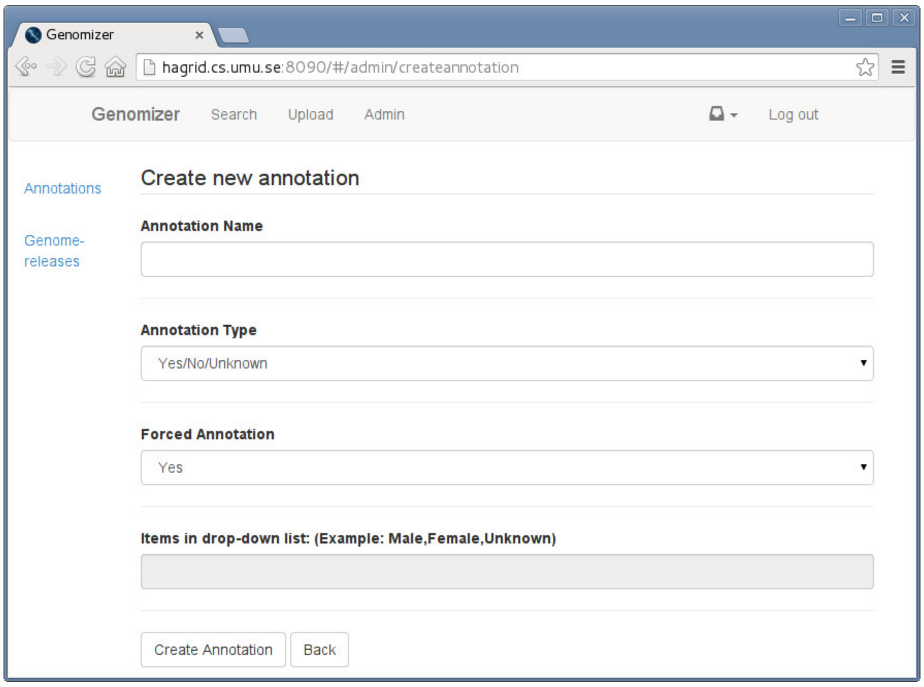
Name
Number of toes

Type
Drop-down list

Forced
No

Values
0
1
2
3
4
5

Figure A.39: The confirm annotation pop-up.



Create new annotation

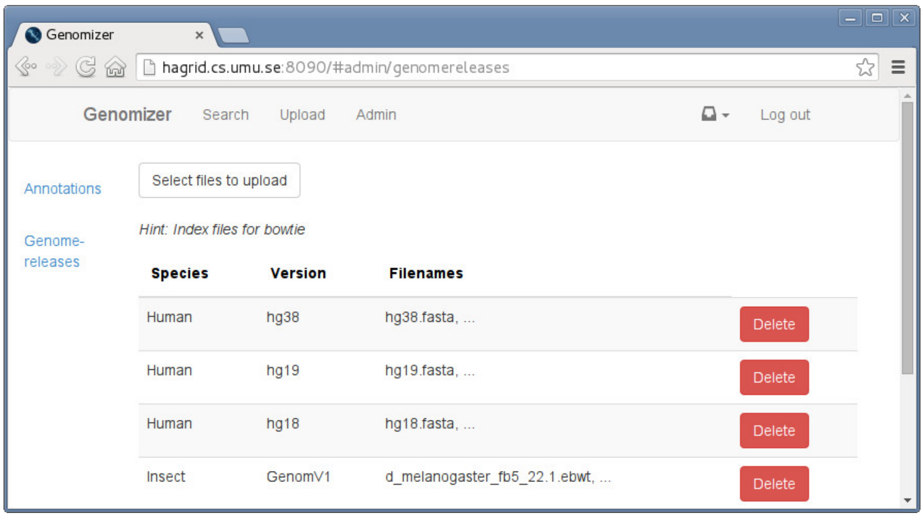
Annotation Name

Annotation Type: Yes/No/Unknown

Forced Annotation: Yes

Items in drop-down list: (Example: Male,Female,Unknown)

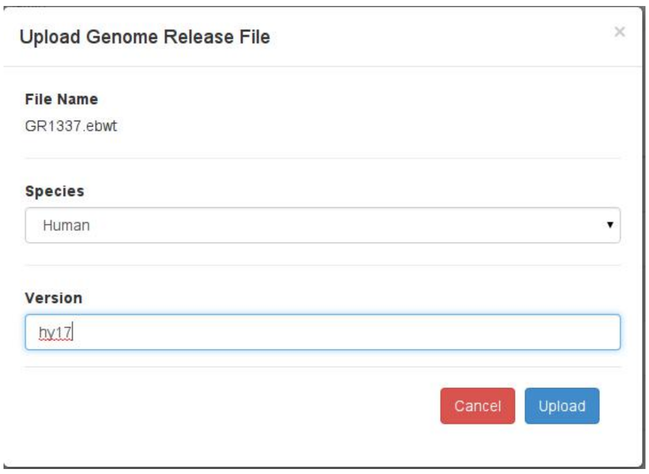
Figure A.40: The view for administrators where new annotations can be created.



Hint: Index files for bowtie

Species	Version	Filenames
Human	hg38	hg38.fasta, ...
Human	hg19	hg19.fasta, ...
Human	hg18	hg18.fasta, ...
Insect	GenomV1	d_melanogaster_fb5_22.1.ebwt, ...

Figure A.41: The genome-release view.


Upload Genome Release File

File Name	GR1337.ebwt
Species	Human
Version	hy17

Figure A.42: Popup for uploading genome releases.

A.3 Android application

In this section instructions for the usage of the *Genomizer* Android application is presented. In subsection A.3.1 there is a description on how to set up the server URL, in subsection A.3.2 you get descriptions on how to login and subsection A.3.4 gives instructions on how to search for experiments.

A.3.1 Setting up the server URL

Start with selecting a server that the application will connect to by pressing the cogwheel in the top right corner seen in Figure A.43. This will take you to a view where you can do any of the following:

- Select one of previously used server URLs
  - See Figure A.44a.
- Add a server URL
  - The + sign in the topright corner of Figure A.44a, will take you to the view displayed in Figure A.44 B where you can add a server.
- Remove a server URL
  - The - sign in the topright corner of Figure A.44a, will show a dialog where you can remove the selected server URL
- Edit an existing server URL
  - The Edit URL button seen in Figure A.44a, will show a dialog where you can edit the selected server URL

When you are satisfied with the server URL simply go back to the *login screen* by pressing the genomizer icon seen in the top left corner of Figure A.44a.

A.3.2 Logging in

In order to login to a server first make sure that the correct server is displayed. If it is, simply insert your username and password in the corresponding boxes and press the *sign in.* If the server is incorrect see subsection A.3.1 on how to fix it. If you don't have a username or password, the system administrator should be contacted to help with the creation of an account.

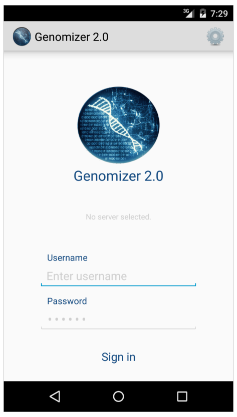
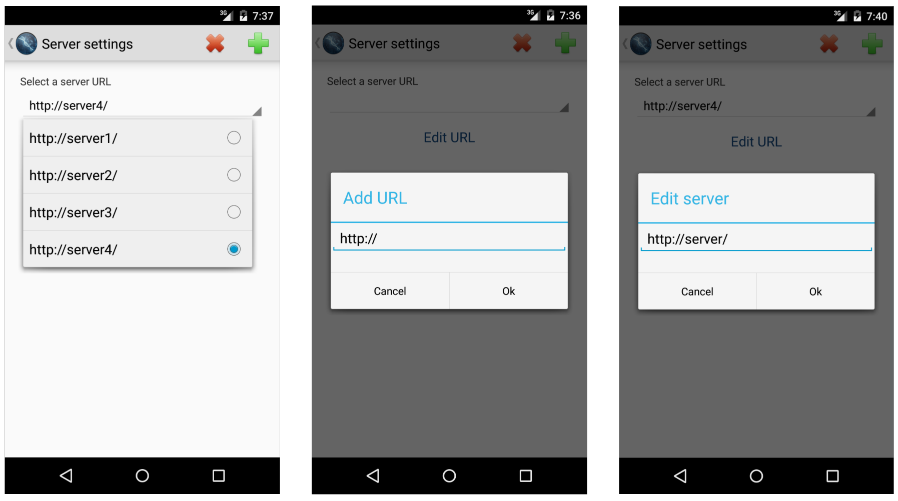


Figure A.43: Login View



a b c

Figure A.44: Settings View

A.3.3 Navigation

To navigate in the application you can open the navigation menu shown in Figure A.45 by pressing the top left corner of any base view (Search and process or Active processes). In this menu you can logout or navigate to the diffent base views of the application which are described below.

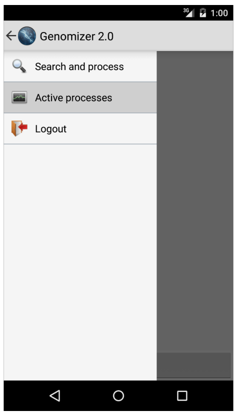
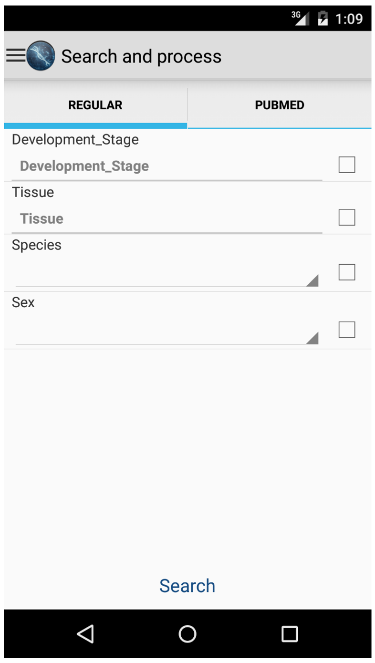


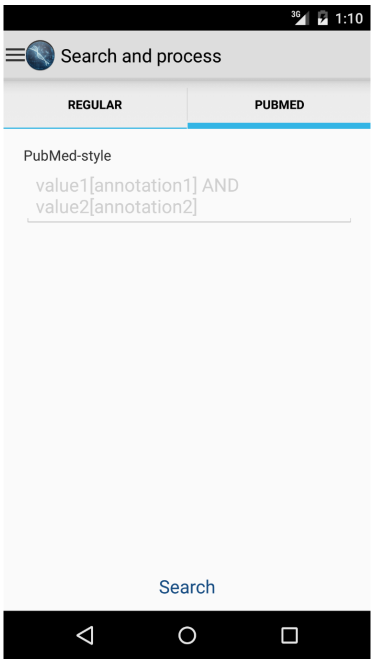
Figure A.45: The Navigation Menu

A.3.4 Search and process

The search view has two tabs, one for regular searches and one for pubmed searches.



a



b

Figure A.46: The Search View

A.3.4.1 Regular search

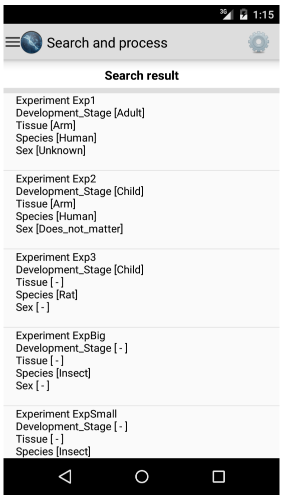
When entering the regular search view illustrated in Figure A.46a, you will be represented with a list of all annotations that experiments in the database have. Chose which value you want for the annotation and press the checkmark next to it in order to add it to the search. Start the search of experiments with the checked annotations by pressing the *Search* button at the bottom of the view.

A.3.4.2 Pubmed search

When entering the pubmed search view illustrated in Figure A.46b, you will be represented with a text field that you manually can fill with a pubmed query if you want to. The query can use perentheses and the logical connectives AND, OR and NOT. Start the search of experiments with the pubmed query by pressing the *Search* button at the bottom of the view.

A.3.4.3 Search results

After a search have been performed you will come to the search results. Here you will see a list of all experiments that have annotations with values corresponding to the search (see Figure A.47). To receive more information about data files that are available for each experiment you can click on each experiment in the list. This will take you to the view seen in Figure A.49a and is described further in subsubsection A.3.4.5.


1:15

Search and process

Search result

Experiment Exp1
Development_Stage [Adult]
Tissue [Arm]
Species [Human]
Sex [Unknown]

Experiment Exp2
Development_Stage [Child]
Tissue [Arm]
Species [Human]
Sex [Does_not_matter]

Experiment Exp3
Development_Stage [Child]
Tissue [-]
Species [Rat]
Sex [-]

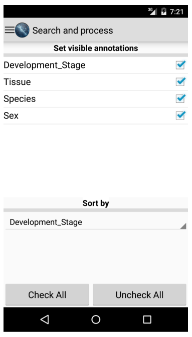
Experiment ExpBig
Development_Stage [-]
Tissue [-]
Species [Insect]
Sex [-]

Experiment ExpSmall
Development_Stage [-]
Tissue [-]
Species [Insect]

Figure A.47: The Search Results View

A.3.4.4 Search result settings

Clicking on the cogwheel button in the top right corner of the view seen in Figure A.47 will take you to the search result settings (see Figure A.48). Here you can Modify which annotations are presented on the experiments of the search result by marking them in the list. You can also order the experiments by one annotation in alphabetical order by chosing that annotation under the *Sort by* header.


7:21

Search and process

Set visible annotations

Development_Stage ✓
Tissue ✓
Species ✓
Sex ✓

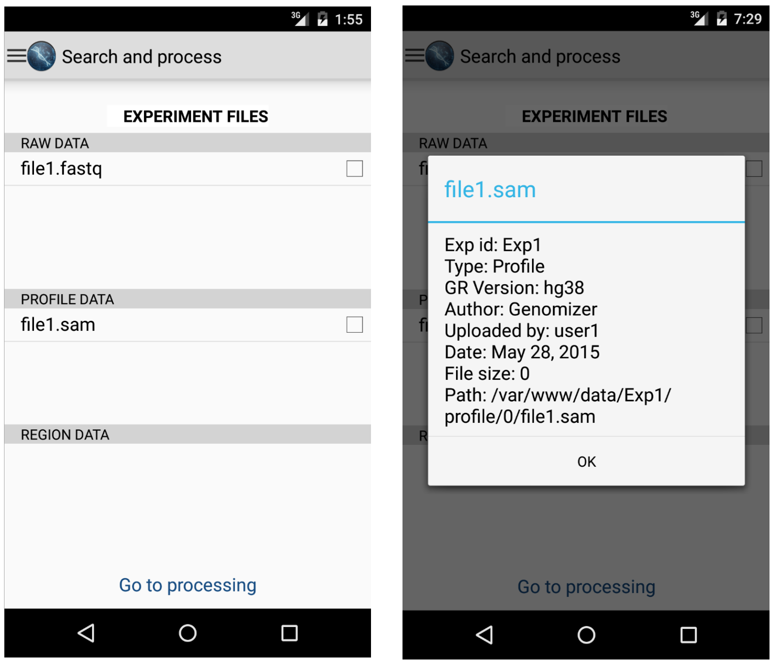
Sort by

Development_Stage

Figure A.48: Search Settings View

A.3.4.5 Experiment file view

Pressing on an experiment in the list of search result seen in Figure A.47 will take you to the experiment file view (see Figure A.49a). You are represented with three lists of files associated to the experiment, one for each type of data file (raw, profile and region). To receive more information about a file, simply click on that file and the information seen in Figure A.49 will be shown. In the bottom there's a button *Go to processing* which will take you to the raw to profile processing stage described in subsubsection A.3.4.6.



file1.sam

Exp id: Exp1
Type: Profile
GR Version: hg38
Author: Genomizer
Uploaded by: user1
Date: May 28, 2015
File size: 0
Path: /var/www/data/Exp1/profile/0/file1.sam

a b

Figure A.49: The Experiment File View

A.3.4.6 Processing

After pressing the *Go to processing* button seen in the bottom of Figure A.49a which will take you to the view seen in Figure A.50. Here you can do a few things before starting the process.

- *Input file* - Chose which raw-file you want to run the processing on.
- *Genome release* - Chose which genome release you want to use in the processing.
- *Output file* - Chose the name of the region file that will be created by the processing.
- *Parameters* - Open a dialoge where you can chose the parameters for the raw to profile processing stage called *bowtie*.
- *Keep sam* - Check this box to keep the sam files that are created during the processing.
- *green plus in top right corner* - add a file to process.
- *red cross next to input file* - remove this file from beeing processed.
- *Process* - starts all processing of the files that are in the list.

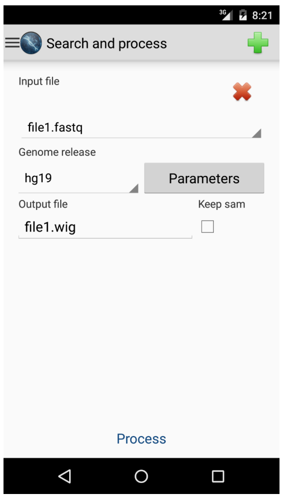


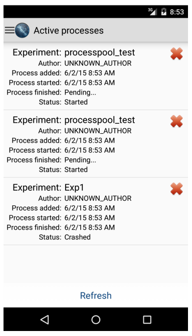
Figure A.50: The Processing View

A.3.5 Active processes

The active process view, illustrated in Figure A.51, is where you can see the current workload on the server. The view contains a list of tasks that has been assigned to the server. Each task contains:

- The name of the experiment the process is running in.
- The author of the processing.
- The time when the process was added.
- The time when the process was started.
- The the time when the process was finished.
- Information about the process current state:
  - Waiting - The task is awaiting processing by the server
  - Started - The task is currently being processed by the server
  - Finished - The task has been completed
  - Crashed - The task was not successfully completed

To abort a running or remove a finished process you can press the red cross seen on each process. This will open a dialog where you have to confirm the removal.



Experiment: processpool_test
Author: UNKNOWN_AUTHOR
Process added: 6/2/15 8:53 AM
Process started: 6/2/15 8:53 AM
Process finished: Pending...
Status: Started

Experiment: processpool_test
Author: UNKNOWN_AUTHOR
Process added: 6/2/15 8:53 AM
Process started: 6/2/15 8:53 AM
Process finished: Pending...
Status: Started

Experiment: Exp1
Author: UNKNOWN_AUTHOR
Process added: 6/2/15 8:53 AM
Process started: 6/2/15 8:53 AM
Process finished: 6/2/15 8:53 AM
Status: Crashed

Figure A.51: The Process View

A.4 iOS application

A.4.1 How to run the app in Xcode

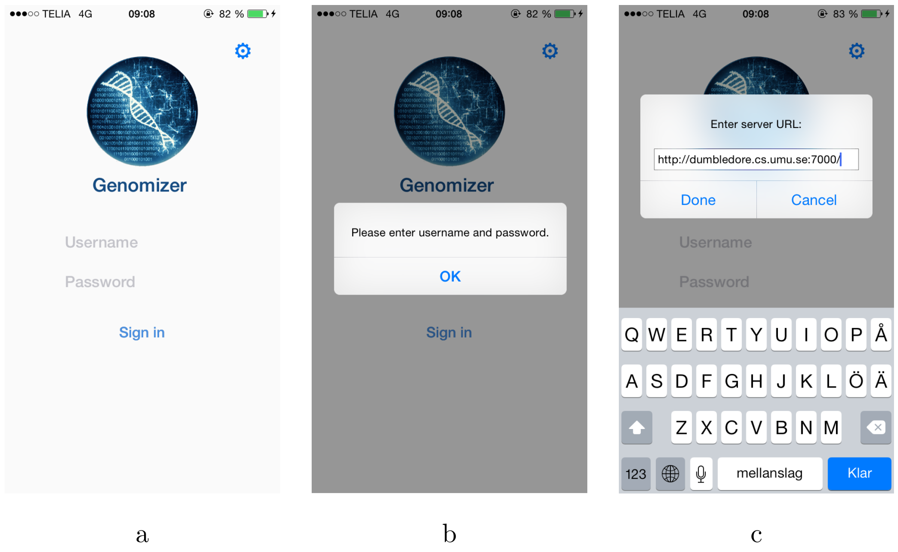
In order to use the program, import the project from github into Xcode from the following repository: `https://github.com/genomizer/genomizer-iOS.git`

To compile and run the program, press ***cmd+R***. A simulator will start and the login screen will be shown as seen in Figure A.52a.

A.4.2 How to login

1. Tap the ***Gear*** in the upper right corner and enter the url and port for the server you want to use and press ***Done***. See Figure A.52a,c.
2. Tap on the ***Username*** textfield and enter your username.
3. Tap on the ***Password*** textfield and enter your password.
4. Tap on ***Sign in*** to sign in.

A user gets logged in when accepted credentials are entered in the ‘username’ and ‘password’ fields and the ‘Sign in’ button is pressed. If incorrect credentials are entered, a popup message is shown, informing the user that the username or password is incorrect.



Please enter username and password.

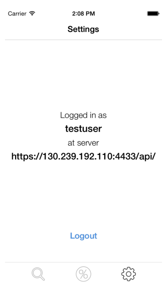
Enter server URL:

http://dumbledore.cs.umu.se:7000/

Figure A.52: The login screen.

A.4.3 How to logout

1. Tap ***Gear***-symbol on the tab bar at the bottom of the screen and a *Setting*-screen will appear. See Figure A.53
2. Tap ***Logout*** to logout.



Logged in as
testuser
at server
https://130.239.192.110:4433/api/

Figure A.53: The settings screen.

A.4.4 How to search for experiments

1. Tap on leftmost button(magnifying glass) on the tab bar which you can find on the bottom of the screen to get to the *Search*-view.
2. Tap on the annotation you want to search for and a spinning wheel with options will appear from the bottom of the screen. See Figure A.54a.
3. Drag the wheel up or down to select the option you want or enter the value with the keyboard depending on the type of annotations is tapped.
4. Enable the annotation to use when searching by toggling the switch to the right of the annotation. See Figure A.54b.
5. Do steps (2)-(5) for more search criteria.
6. Tap ***Search*** to search.

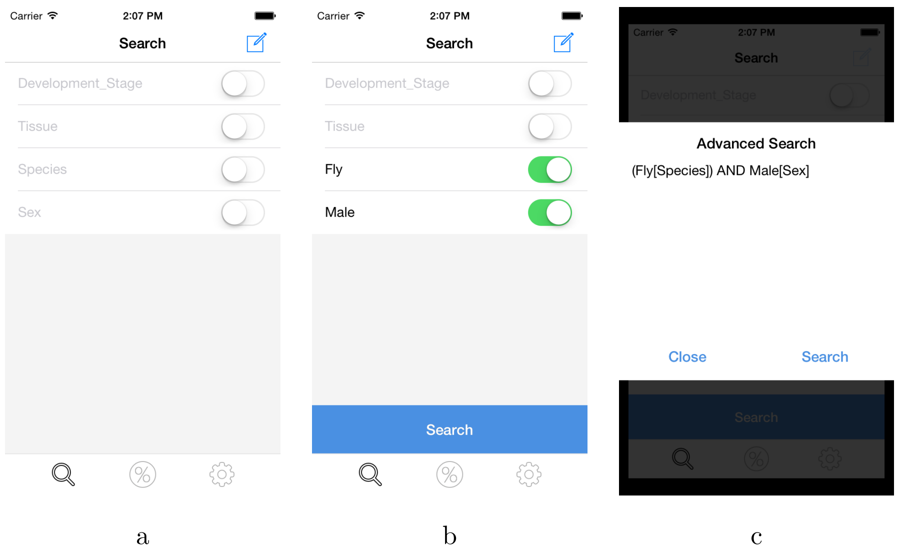


Figure A.54: The search screen.

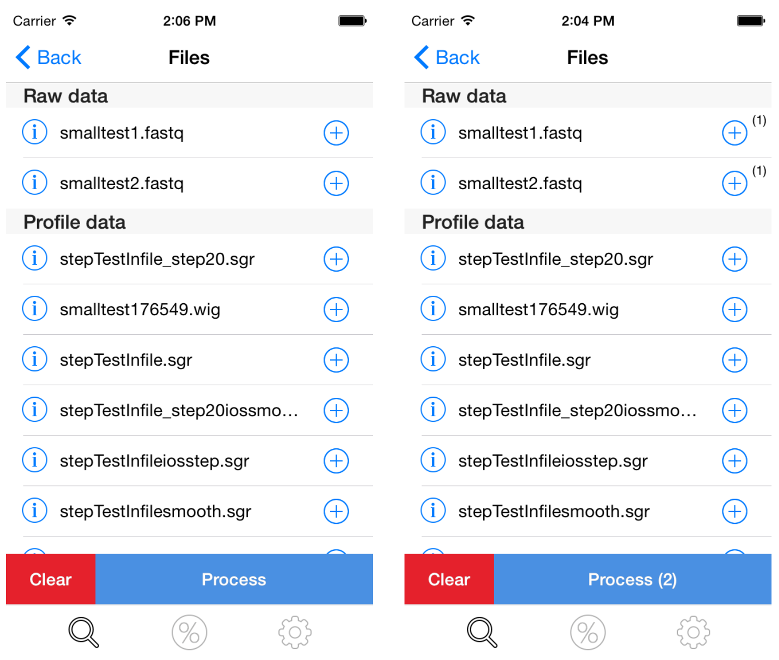
A.4.5 How to use advanced search

1. Tap on leftmost button(magnifying glass) on the tab bar which you can find on the bottom of the screen to get to the *Search*-view.
2. Tap on the symbol to top right of the screen and a new view will appear with the title *Advanced Search*. See Figure A.54c.
3. Write your search criteria in PubMed-style and tap ***Search***

The annotations you select on the search-screen will also show in advanced search.

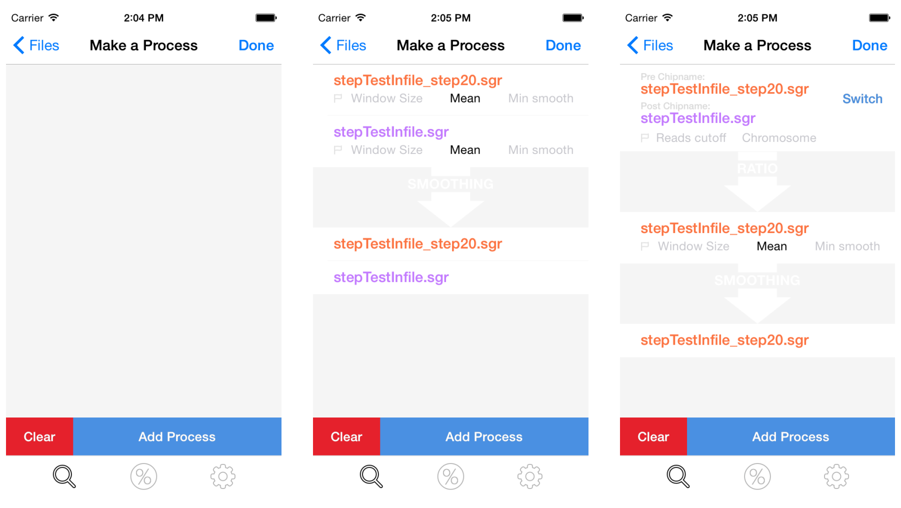
A.4.6 How to process files

1. Search for experiments
2. In *Search Results*-screen tap on an experiment and the *Files*-screen will appear showing the files which belongs to the experiment. See Figure A.55a
3. Tap the ***Plus***-symbol next to the file you want to process.
4. Do step (3) for every file you want to process. Same file can be used more than once. A counter will appear next to the Plus-symbol to keep track on how many times the file will be used in the process-step, see Figure A.55b.
5. Tap the ***Process***-button and *Make a process*-view will appear with every file you have selected. See Figure A.56a
6. Tap ***Add Process*** and select what kind of process you want to do on the selected files.
7. After you have selected a process the input files and output files will appear with the selected process separating them. You can add parameter values by tapping the parameter fields below the input files. The output files can be renamed by tapping on the filename. See Figure A.56b
8. Do step (6) to create a sequence of processes to be made on the selected files. See Figure A.56c for an example of a process sequence.
9. Tap ***Done***-button to send the sequence of processes to the server.



a b

Figure A.55: Files view.



a b c

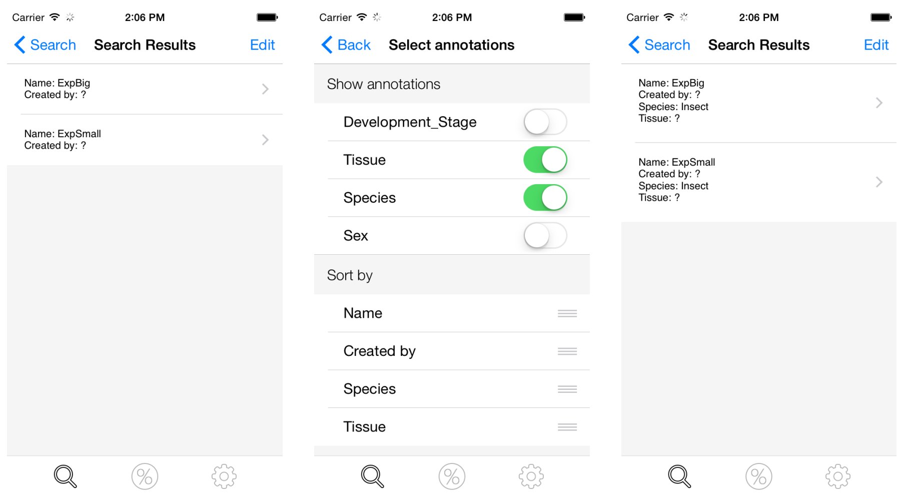
Figure A.56: Create processes.

A.4.7 How to set which annotation to be visible on Search Results

1. In Search Results view, tap ***Edit*** and *Select Annotations*-screen will appear
2. Select which annotations to show by toggle the switch next to each annotation.
3. Tap ***Back*** to go back to *Search Results*. See Figure A.57a-c

A.4.8 How to change the order which search results appear in

1. In Search Results view, tap ***Edit*** and *Select Annotations*-screen will appear
2. Under the *Sort by* header, drag-and-drop the annotation-names in the order you wish to sort the search result


Carrier 2:06 PM
Search Search Results Edit
Name: ExpBig
Created by: ?
Name: ExpSmall
Created by: ?

Carrier 2:06 PM
Back Select annotations
Show annotations
Development_Stage
Tissue
Species
Sex
Sort by
Name
Created by
Species
Tissue

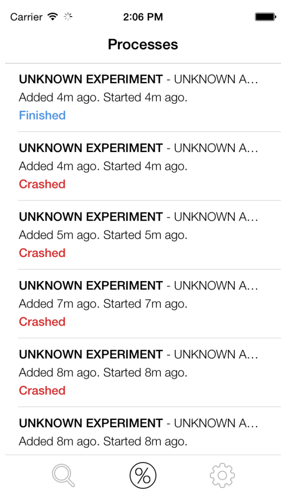
Carrier 2:06 PM
Search Search Results Edit
Name: ExpBig
Created by: ?
Species: Insect
Tissue: ?
Name: ExpSmall
Created by: ?
Species: Insect
Tissue: ?

a b c

Figure A.57: Select annotation

A.4.9 How to view process status on the server

1. Tap ***Process***-symbol (the percentage symbol) to view the processes on the server.
2. To refresh the view, drag the view down until an activity indicator icon is visible below the title of the screen and release. See Figure A.58



Processes

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 4m ago. Started 4m ago.
Finished

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 4m ago. Started 4m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 5m ago. Started 5m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 7m ago. Started 7m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 8m ago. Started 8m ago.
Crashed

UNKNOWN EXPERIMENT - UNKNOWN A...
Added 8m ago. Started 8m ago.

Figure A.58: The process screen.

B Deployment and maintenance

This chapter is directed towards administrators and developers that wants to set up a server and install the software needed to get a fully functional system. It also gives instructions on how to maintain the system in case of problems that can arise.

B.1 Configure server

This chapter is directed towards administrators and developers who want to set up a server and install the software needed to get a fully functional system. It also contains instructions on how to maintain the system in case problems arise.

B.2 A brief introduction to vagrant

B.2.1 Basic usage

Vagrant is configured by a `Vagrant` file, which defines how the virtual machine is constructed. This should not be modified unless you understand what you are doing.

To start a vagrant instance, run `vagrant up`. The vagrant instance will now either (a) or (b):

a) Create a new virtualmachine with correct configuration
b) Start an already built virtual machine

If the instance is stale or needs to be rebuilt, it can be achieved with

```
vagrant destroy
vagrant up && vagrant ssh -c 'bash startup.sh'
```

If you simply run the server using `vagrant up`, then `vagrant ssh -c 'bash startup.sh'` starts the genomizer server in the virtual machine, inside a screen session labeled `server`.

B.2.2 Modifying the configuration

To make vagrant set up a specific branch, modify the corresponding script, such as `scripts/provision/install_genomizer_server` to checkout `yourBranch` instead of `develop`.

If your feature requires changes to the settings.cfg it is located at `scripts/provision/config/settings.cfg` and is installed along with a fresh instance of the virtual machine.

B.2.3 Entering the vagrant virtual machine

If you need to reach the environment using SSH, simply go to the environment you require with

```
cd development-1
vagrant ssh
```

and you will be logged in using ssh to the virtual machine.

B.3 Systems overview of production

The production server runs currently on `130.239.192.110`. The production server is constructed of several virtual machines, configured by vagrant[1].

The system is entirely scripted, that is, everything can be set up using the scripts located in the vagrant git[2]. The scripts located under `scripts/provision` deal with the virtual environments, while the scripts under `scripts/environment` deal with the server, such as setting up the firewall and external directories used by the virtual machines.

B.3.1 Using the toolchain

This is a short introduction to the tools used in the environments.

B.3.1.1 Using gnu screen

To ensure processes run even when nobody has a terminal open to the server `screen` is used. `screen` is a terminal multiplexer, which could be viewed as a in-terminal window manager. To start a new screen you run the command

```
$ screen
```

which then starts a `screen` and attaches your current terminal to it. `screen` is controlled by prefixing a command with a modifier. For example, to detach from a `screen` window, you use

[1]Vagrant is a virtual machine configuration manager. The manual is available at https://docs.vagrantup.com/v2/

[2]git.cs.umu.se, access required.

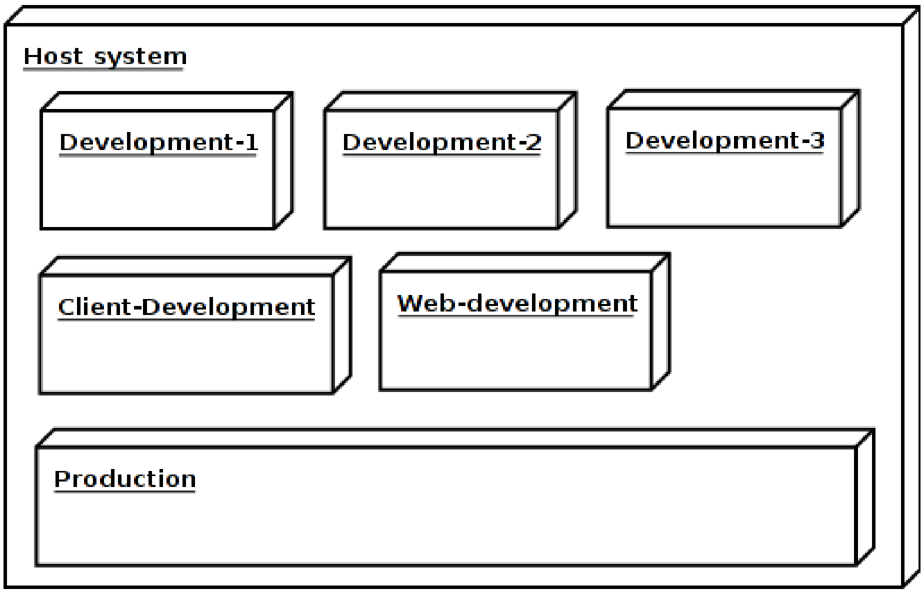


Figure B.1: Illustration of system setup

`<C-a>` which you then release, indicating you wish to send a command to `screen`. You then press `d`, which is the hot-key for *detach*. You are now detached from the screen.

You might not be surprised when I tell you that several `screen` windows can be running at the same time. To list the currently running `screen` sessions, you run the command

```
$ screen -list
```

If there is only one session running, you can attach to it using simply the command

```
$ screen -r
```

to attach. If there are several running `screens`, you must specify which session to attach to. The following is an example of this. It starts two sessions, `first` and `second` in a detached state. The `-dmS` flag means they are started in a detached state. We then list the sessions and see that several are running, as expected.

```
[vagrant@localhost ~]$ screen -dmS "first"
[vagrant@localhost ~]$ screen -dmS "second"
[vagrant@localhost ~]$ screen -list
There are screens on:
    15073.second    (Detached)
    15053.first (Detached)
2 Sockets in /var/run/screen/S-vagrant.
```

We can now attach to these sessions by running the command, replacing `<session-name` with the name of your session.

```
$ screen -S <session name>
```

To terminate a `screen`, attach to it and run `exit`.

B.3.2 Configured environments

The environments as currently configured are described below.

B.3.2.1 Production

This environment runs the production code. Hands off if you are unsure of or have not communicated your change clearly to everyone else.

- Virtual hardware
  - 45000 MB RAM
  - 8 CPU Cores
- Ports
  - Genomizer-server: 7000
  - Http: 80
  - Https: 443
- Local IP: 192.168.33.10
- Directories
  - Temporary files
    - External directory: `/Data/tmp`
    - Internal directory: `/tmp`
  - Data files
    - External directory: `/Data/production-data`
    - Internal directory: `/data`

B.3.2.2 Development-1

This environment is assigned to the `database` group.

- Virtual hardware
  - 4000 MB RAM
  - 1 CPU Core
- Ports
  - Genomizer-server: 7001
  - Http: 8081
  - Https: 4431
- Local IP: 192.168.33.11
- Directories
  - Temporary files
    - External directory: `/Data/development-1-tmp`
    - Internal directory: `/tmp`
  - Data files
    - External directory: `/Data/development-1-data`
    - Internal directory: `/data`

B.3.2.3 Development-2

This environment is assigned to the `business-logic` group.

- Virtual hardware
  - 4000 MB RAM
  - 1 CPU Core
- Ports
  - Genomizer-server: 7002
  - Http: 8082
  - Https: 4432
- Local IP: 192.168.33.12
- Directories
  - Temporary files
    - External directory: `/Data/development-2-tmp`
    - Internal directory: `/tmp`
  - Data files
    - External directory: `/Data/development-2-data`
    - Internal directory: `/data`

B.3.2.4 Development-3

This environment is assigned to the `processing` group.

- Virtual hardware
  - 4000 MB RAM
  - 1 CPU Core
- Ports
  - Genomizer-server: 7003
  - Http: 8083
  - Https: 4433
- Local IP: 192.168.33.13
- Directories
  - Temporary files
    - External directory: `/Data/development-3-tmp`
    - Internal directory: `/tmp`
  - Data files
    - External directory: `/Data/development-3-data`
    - Internal directory: `/data`

B.3.2.5 Client-development

This environment is assigned to the `desktop` group.

- Virtual hardware
  - 4000 MB RAM

  - 1 CPU Core

- Ports
  - Genomizer-server: 7004
  - Http: 8084
  - Https: 4434
- Local IP: 192.168.33.14
- Directories
  - Temporary files
    - External directory: `/Data/development-client-tmp`
    - Internal directory: `/tmp`
  - Data files
    - External directory: `/Data/development-client-data`
    - Internal directory: `/data`

B.3.2.6 Web-development

This environment is assigned to the `web` and `app` groups.

- Virtual hardware
  - 4000 MB RAM
  - 1 CPU Core
- Ports
  - Genomizer-server: 7005
  - Http: 8085
  - Https: 4435
- Local IP: 192.168.33.15
- Directories
  - Temporary files
    - External directory: `/Data/development-web-tmp`
    - Internal directory: `/tmp`
  - Data files
    - External directory: `/Data/development-web-data`
    - Internal directory: `/data`

B.3.3 The important scripts

The machines are configured using a myriad of scripts with various responsibilities. The `Vagrantfile` defines how the virtual machine is built. This includes which provisioning files are to be executed, and how much memory and resources to give the machine. It also configures the port forwarding into the machine, and which shared data folders are available to the machine. Messing with these configurations is inadvisable.

The `Vagrantfile` runs, as previously said, configuration scripts. Their names and basic responsibilities are listed below. Unless otherwise specified, they are located under `scripts/provision`.

1. `install_apache.sh` - Installs and configures the apache server. It installs the config files `httpd.conf`, `ssl.conf`, and `proxy.conf` located under `scripts/provision/config`, along with running the `install_certificates` script.

2. `install_postgresql.sh` - While the name may seem confusing, this installs `puppet` and the `puppet/postgresql` module. This allows the `Vagrantfile` to run puppet provisioning on the machine to install postgresql with the settings provided in `manifests/build.pp`.
3. `install_certificates.sh` - This is an expect script, which is run by the `install_apache` script. This is introduced to deal with the fact that you need to respond to questions to generate a SSL certificate. It generates a SSL certificate, and moves it to the correct location.
4. `install_utils.sh` - There are some tools and utilities which were requested or needed by the scripts or persons working on the project. They are installed from this script.
5. `install_startup_scripts.sh` - Installs various scripts that are used for server administration.
6. `install_genomizer_server.sh` - Installs the genomizer-server
7. `install_genomizer_webclient.sh` - Installs the genomizer-web client to the apache server.

If you are confused by what a script does, reading it can give you a clearer picture of what it does.

B.3.4 Creating a new environment

B.3.4.1 Considerations

Before setting up a new environment, you should consider the memory and CPU footprint inherent in running a virtual machine. The host machine as currently configured can run 6 environments with no noticeable slowdown, with five machines at 1 cpu core and 4GB of RAM, and a production environment that is assigned 8 CPU cores and 45GB of RAM. It is not recommended to exceed this configuration, as unexpected side-effects could occur.

B.3.4.2 Checking out the correct files

When building a new environment, you may wish to configure which versions of the server software and web client software the script installs. By default, the scripts are configured to fetch the `master` branch of the `genomizer-server` and the `master` branch of the `genomizer-web` application. These settings are modified by editing the `scripts/provision/install_genomizer_server.sh` and `scripts/provision/install_genomizer_webclient.sh` respectively.

In order to change the version of the `genomizer-server`, edit the `scripts/ provision/install_genomizer_server.sh` script. Locate the line(s)

```
# We checkout the master version by default
git checkout master
```

and replace them with

```
# We checkout the <desired branch> version by default
git checkout <desired branch>
```

To change the version of the `genomizer-web` client, edit the `scripts/provision/ install_genomizer_webclient.sh` script. Locate the line(s)

```
# We run the master branch by default
cd genomizer-web
git checkout master
```

and replace them with

```
# We run the <desired branch> branch by default
cd genomizer-web
git checkout <desired branch>
```

B.3.4.3 Setting up the files

When setting up the new environment, you may (or may not) want to run it with the same settings as the production environment. In this section all relevant settings are explained and motivated.

B.3.4.3.1 Editing the Vagrantfile

In the vagrant file, the primarily interesting settings are concerning *port forwarding*, *local ip*, *synced folders*, and *memory*. Given that you wish to customize the environment, the *provision* settings might be of use as well. The `Vagrantfile` is written in a Ruby DSL.

The port forwarding settings looks as follows.

```
config.vm.network "forwarded_port", guest: 80, host:8080
config.vm.network "forwarded_port", guest: 7000, host:7000
config.vm.network "forwarded_port", guest: 443, host:4443
```

The `host` field defines which port the virtual machine binds on in the host machine, while the `guest` defines the port the virtual machine it binds on internally.

To define which **local IP** the virtual machine runs on, you modify the line below to suit your needs. Note that the IP must be unique, and it is recommended to stay inside the `192.168.33.x` subnet.

```
config.vm.network "private_network", ip: "192.168.33.10"
```

You may also need to mount a folder from the host machine as a drive inside the virtual machine. This is achieved by editing or adding to the lines shown below. The first argument is the hosts path to the folder to share, and the second is where the virtual machine should mount it.

```
config.vm.synced_folder "/Data/production-data", "/data"
config.vm.synced_folder "/Data/production-tmp", "/tmp"
```

The actual hardware specifications given to the virtual machine is defined by these lines:

```
config.vm.provider "virtualbox" do |vb|
    vb.memory = "45000"
    vb.cpus = 8
end
```

These lines define that the machine should run with 45000 MB of RAM, and run on 8 CPU cores. If other settings are desired, you modify these lines. For example, the development environments might look like

```
config.vm.provider "virtualbox" do |vb|
    vb.memory = "4000"
    vb.cpus = 1
end
```

B.3.4.3.2 Editing the settings.cfg

The `settings.cfg` file contains the settings that are installed to the genomizer-server. These should need no modification, except perhaps to edit the tunneling settings.

If you however do want to edit some settings, note that the database settings must correspond to the ones configured in `build.pp`, else the server will fail to connect to the database.

The settings available in the server settings are straightforward, and are commented such that no further explanation is required.

B.3.4.3.3 Editing the httpd.conf

The `httpd.conf` file is dead-standard, except for the very last few lines. These lines assures that SSL is forced, and that any traffic connecting to the non-SSL apache is told to reconnect with SSL.

```
<VirtualHost *:80>
    RedirectPermanent / https://130.239.192.110
</VirtualHost>
```

You may wish to edit where this reroutes to as needed.

B.3.4.3.4 Modifying the build.pp

The `build.pp` file details how the postgresql database is constructed, along with which sql files are automatically run on the server. It is not recommended to change how this file functions except to change the password and modify which sql files are run.

B.3.4.3.5 Creating the `tmp` and `data` folders

When constructing an environment, the machine requires a place to store temporary files, and a place to store large data files. These are by default placed in `/Data/<environment>-tmp` and `/Data/<environment>-data`. To create these folders, and give them the correct permissions, you should run the following commands. The `<your-user>` should be replaced with the user that runs the virtual machines, and `<environment>` with the name of your environment.

```
[user@your-environment]$ sudo mkdir /Data/<environment>-tmp
[user@your-environment]$ sudo mkdir /Data/<environment>-data
[user@your-environment]$ sudo chown -R <your-user> /data/<environment>-tmp
[user@your-environment]$ sudo chmod -R 1777 /data/<environment>-tmp
[user@your-environment]$ sudo chmod -R 777 /data/<environment>-data
```

B.3.4.4 Running the new environment

Once you have created an environment, you need to allow vagrant to construct the associated virtual machine. This is done by running `vagrant up` in the directory associated with your environment.

Vagrant should now start building the virtual machine, along with provisioning it. It will write a lot of information to the terminal you run it in, and this is perfectly normal. The build takes on average 5-10 minutes to complete.

Once it has finished, the environment is up and running, except for one detail: `genomizer-server` is not running. You can start it by typing the following into the terminal.

```
[user@environment]$ vagrant ssh
Last login: Tue May 12 12:4032 2015 from 10.0.2.2
[vagrant@localhost]$ bash startup.sh
```

If you wish to view the server process, this can be done by running

```
[vagrant@localhost]$ screen -r
```

which attaches to the screen created by `startup.sh`.

B.3.5 Modifying an existing environment

While it is very useful to build an environment from scratch, you do not always want to do everything from scratch. You want to *modify* an existing environment. Modifying an environment is fairly straight forward, you simply edit the configuration files, as shown in *Creating a new environment*. Once you have modified the files according to your preferences, you run

```
[user@your-environment]$ vagrant destroy && vagrant up
```

if you wish to *completely* rebuild the machine. If you simply want the machine to reload the `Vagrantfile` settings, you can run

```
[user@your-environment]$ vagrant reload
```

B.3.5.0.1 Checkout new branches

If you decide that you wish to run another branch in the environment, and do not wish to rebuild the environment, you can simply ssh into the vagrant machine and checkout using git as you normally would, compiling and running as you would normally.

```
[user@your-environment]$ vagrant ssh
[vagrant@localhost]$ cd genomizer-server
[vagrant@localhost]$ git stash
[vagrant@localhost]$ git checkout <branch>
[vagrant@localhost]$ git stash pop
[vagrant@localhost]$ ant clean build jar
[vagrant@localhost]$ java -jar server.jar
```

In this context, the `stash` ensures that our modified `settings.cfg` is preserved. If for some reason `settings.cfg` is overwritten, you can get a fresh copy from `/vagrant/scripts/provision/config/settings.cfg` inside the virtual machine.

B.3.6 Rebuilding an environment

Rebuilding an environment is dead simple, simply run

```
[user@your-environment]$ vagrant destroy && vagrant up
```

B.3.7 Deleting an environment

NOTE: The changes explained in this chapter are irreversible!

To delete an environment it is not sufficient to simply remove the folder containing the environment. The virtual machine must also be deleted. The following steps removes an environment cleanly.

1. Enter the environment folder.
2. Destroy the virtual machine with `vagrant destroy`.
3. Remove the `.vagrant` folder.
4. Remove the environment folder.
5. Remove the environments `tmp` and `data` folders.

The `tmp` and `data` folders are by default located under `/Data/<environment>-tmp` and `/Data/<environment>-data`.

B.3.8 Configuring the host system

Please note, all of these things can be achieved by running the script `scripts/ environment/setup_host_environment.sh`, provided a CentOS host system. **Never run this script without reading it through. Seriously.** Even if you want to configure manually, reading this script is useful to get examples of the commands that are relevant.

B.3.8.1 External folders and their permissions.

The host machine is required to make available storage space for the virtual machines. By default this is assumed to be `/Data`. In this folder, you should place `tmp` and `data` folders – one for each environment – with the permissions 1777 for `tmp`. The `data` folder should have the permissions 777. The `tmp` folder must be owned by the vagrant user.

B.3.8.2 Configuring the firewall

The firewall needs to allow *at least* the ports `80`, `443` and `7000`. There is a script which configures these settings under `scripts/environment/iptables_config.sh`. The only advanced setting here is a workaround to patch reserved ports into the virtual machines without requiring root permissions.

The firewall is instructed to reroute traffic to port `80` to the port `8080` and traffic to the port `443` to the port `4443` which then the virtual machine can bind on.

Specifics on how to run these commands are illustrated in `iptables_config.sh`.

B.3.8.3 Prerequisite software

The setup requires `vagrant` and `virtualbox` to be installed.

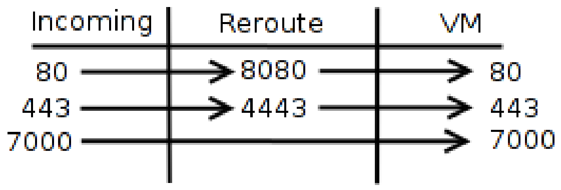


Figure B.2: Illustration of rerouting

B.3.8.3.1 Virtualbox

To install virtualbox, you can run the following commands. Replace `$VAGRANT_USER` with the vagrant user.

```
# Get the EPEL repo
rpm -Uvh https://download.fedoraproject.org/pub/epel/7/x86_64/e/\
    epel-release-7-5.noarch.rpm

# Install kernel headers and devel tools
yum -y install kernel-devel kernel-headers dkms
yum groupinstall "Development Tools"
yum update

# Install oracle public key
wget http://download.virtualbox.org/virtualbox/debian/\
    oracle_vbox.asc
rpm --import oracle_vbox.asc
rm -rf oracle_vbox.asc

# Add the virtualbox repo
wget http://download.virtualbox.org/virtualbox/rpm/el/\
    virtualbox.repo -O /etc/yum.repos.d/virtualbox.repo
yum update # Update repo info for safety

# Time to actually install virtualbox
yum -y install VirtualBox-4.3
service vboxdrv setup # Setup vbox driver
usermod -a -G vboxusers $VAGRANT_USER # Add VB user
```

B.3.8.3.2 Vagrant

To install vagrant, you can run the following commands.

```
wget https://dl.bintray.com/mitchellh/vagrant/\
    vagrant_1.7.2_x86_64.rpm
rpm -ivh vagrant_1.7.2_x86_64.rpm
rm -rf vagrant_1.7.2_x86_64.rpm
```

B.4 Administer the database

This chapter contains instructions for setting up the postgresql database and user accounts.

B.4.1 Set up postgresql account

This step is only required if you do not already have a `psql` username and password. If you have been assigned this from a sysadmin proceed to *Upload SQL Script to server*.

1. Log in to the server:

   ```
   > ssh <username>@<host>
   ```

2. Become sudo-user "postgres":

   ```
   > sudo su postgres
   ```

3. Add yourself as a postgresql user:

   ```
   > createuser <username>
   ```

4. Log into postgresql as root:

   ```
   > psql
   ```

5. Set your password:

   ```
   > \password <username>
   ```

6. Create database:

   ```
   > create database genomizer;
   ```

7. Grant yourself all permissions on the *Genomizer* database:

   ```
   > grant all on database genomizer to <username>;
   > \q
   ```

8. Navigate to postgresql configuration folder:

   ```
   > cd /
   > cd etc/postgresql/9.3/main
   ```

9. Navigate to postgresql configuration folder:

   ```
   > sudo nano postgresql.conf
   ```

10. Change connection settings:
    Locate line:

    ```
    #listen_adresses = '<settings>'    # what IP address(es) to listen on;
    ```

    Change to:

    ```
    listen_addresses = '*'    # what IP address(es) to listen on;
    ```

11. Write changes and exit:
    Hold down ctrl and press o
    Hold down ctrl and press x
12. Open configuration file:

    ```
    > sudo nano pg_hba.conf
    ```

13. Change Client Authentication Configuration:
    Locate the heading:

```
# IPv4 local connections:
```

Under the heading, add the line:

```
host    all    all    127.0.0.1/32    md5
```

14. Write changes and exit:
    Hold down ctrl and press o
    Hold down ctrl and press x
15. Restart postgresql:

```
> cd /
> sudo /etc/init.d/postgresql restart
```

B.4.2 Upload SQL Script to server

1. In a termainal window navigate to the folder where the `genomizer_database_tables.sql` script resides.
2. Establish secure ftp connection to the server:

```
> sftp <username>@<host>
```

3. Create a new folder on the server:

```
> mkdir SqlScripts
```

4. Upload `genomizer_database_tables.sql`:

```
> put genomizer_database_tables.sql SqlScripts/
```

5. Exit `sftp`:

```
> exit
```

B.4.3 Create the *Genomizer* Tables

1. Log in to the server:

```
> ssh <username>@<host>
```

2. Log in to the database:

```
> psql genomizer
```

3. Run `genomizer_database_tables.sql`

```
> \i SqlScripts/genomizer_database_tables.sql
```

The *Genomizer* database is now ready to use.

B.5 Install the server

Installing the server requires these three programs to exist on the server machine.

- *Java JDK*: To run the server.
- *Git*: To download the server source code.
- *Ant*: To build the server.

The source code for the server is hosted by *Github* and can be downloaded using *Git* or using the home page and get it as a *.zip*.

Downloads using *Git* is done with the help of `clone`.

```
git clone https://github.com/genomizer/genomizer-server.git
```

After the source code has been downloaded the server needs to be compiled and built into a runnable `jar`. This is done with the help of *Ant*. The repository has a build script in the directory, and has a command to build a jar.

```
ant jar
```

All that is left before starting the server is editing the settings file in the main directory for the server(same place as the `jar`). The settings file is named `settings.cfg`, it is designed as followed.

```
# Database settings.

databaseUser        = genomizer
databasePassword    = password
databaseHost        = localhost:5432
databaseName        = genomizer

# Port number on which the server listens for HTTP connections.
genomizerPort       = 7000

# Directory on the server where uploaded files are stored.

fileLocation        = /data/

# Max. number of threads in the processing work pool.

nrOfProcessThreads  = 5

# Paths to various programs that we use.

bowtieLocation      = resources/bowtie/bowtie
picardLocation      = resources/picard-tools/picard.jar
bowtie2Location     = resources/bowtie2/bowtie2
pyicosLocation      = resources/pyicoteo/pyicos

# Address and port of the publicly visible WWW server (if any)
# that is tunnelling requests to the genomizer server.
# 'wwwTunnelPath' is the path on the WWW server that
# requests to genomizer server are mapped to (usually '/api').

wwwTunnelHost       = https://130.239.192.110
wwwTunnelPort       = 4433
wwwTunnelPath       = /api

# Directory used for storing temporary files during uploading.
# Should be on the same partition as 'fileLocation'.
uploadTempDir       = /data/tmp/
```

All that is left after the settings is correct is to start the server.

```
java -jar server.jar [-debug|-p <port> | -f <alternativSettings.cfg> | -nri ]
```

As seen the server may be started with optional arguments.

- `debug`: Starts the server with a active debug logger. Printing everything to `stdout`.
- `p`: Alternativ port.
- `f`: Alternativ settings file other then `settings.cfg`
- `nri`: No remove inactive users. Halts the thread that removes inactive user tokens.

B.6 Set up processing

To be able to run the processes such as raw to profile convertion the right scripts and programs need to be in the folder resources. The scripts needed for converting will be there but bowtie need to be downloaded and extracted to resources, which need to be a folder in the servers root directory.

The programs and scripts needed are:

- `bowtie`: base pair processing. Processes *raw* files into *profile*.
- `picard`: generates the *profile* to a sam file.
- `bowtie2`: Updated version of bowtie.
- `pyicos`: Mesuring tool for *DNA* and *RNA*.

C User Stories

A *User Story* is a description of functionality in non technical terms. It describes the wishes of a certain user group and a motivation for why the function is needed.

C.1 Implemented user stories

Annotation
To structure the data files the researchers want to be able to annotate the data files.

Figure C.1: Annotation user story

Single download
To scrutinize a single data file the researchers want to be able to download a specific file.

Figure C.2: Download user story

Single upload
To store a single data file the researchers want to be able to upload a specific file.

Figure C.3: Upload user story

Search for data
To analyse data the researchers want to be able to search for specific types of data.

Figure C.4: Search for data user story

Raw to profile
To be able to analyse the researchers want to process raw data to profile data (using bowie and then Philge's code).

Figure C.5: Raw to profile user story

Delete data
To save space the researchers want to be able to delete data from the database.

Figure C.6: Delete data user story

Change annotation
To correct and update annotations the researchers want to be able to change data annotations.

Figure C.7: Change annotation user story

Backup
To prevent loss of data the researchers want the data to be backed up.

Figure C.8: Backup user story

Password protected
To protect the database from unauthorized use the researchers want the application to be password protected.

Figure C.9: Password protected user story

Add genome release / reference genome
To be able to annotate the data properly and extract genome reference the researchers want to be able to add genome releases and reference genome.

Figure C.10: Add genome release / reference genome user story

Batch download

To scrutinize several data files
the researchers
want to be able to download multiple files at once.

Figure C.11: Batch download user story

Convert common file formats

To get data in a certain convenient file format
the researchers
want to convert between common file formats (WIG, SGR, GFF3, BED).

Figure C.12: Convert common file formats user story

Work queue

To reduce server load
the researchers
want to queue time consuming work.

Figure C.13: Work queue user story

User rights

To allow invitation of guests (postgraduate students or other researchers etc.)
the researchers
what to have different users types with different rights.

Figure C.14: User rights user story

Batch upload

To analyse, share and have greater access to data
the researchers
want to be able to upload multiple files and batch annotate them to a shared location.

Figure C.15: Batch upload user story

Sort search results
To easier find specific result the researchers want to filter the search result.

Figure C.16: Sort search results user story

C.2 Product backlog

Add chain file

To be able to convert between genome releases
the researchers
want to upload chain files (LiftOver).

Figure C.17: Add chain file user story

File traceabillity

To be able to access the underlying raw data or profile data
the researchers
want the raw data files to be traceable from profile files and the profile files to be traceable from the region data (if available) when the files have been generated on the server.

Figure C.18: File traceabillity user story

Convert genome release

To easier handle files
the researchers
want to convert files between genome releases (LiftOver).

Figure C.19: Convert genome release user story

Extract genome reference sequence

To analyse the reference genome
the researchers
want extract the reference genome sequence for a given region data.

Figure C.20: Extract genome reference sequence user story

Advanced batch upload

To simplify mass upload 500 files
the researchers
want to batch annotate files to be uploaded in a spreadsheet.

Figure C.21: Advanced batch upload user story

Profile to region

To be able to find regions of interest
the researchers
want to process profile data to region data (Per's code).

Figure C.22: Profile to region user story

Workspace

To be able to save work in a convenient way
the researchers
want to have some sort of workspace view where all kind of results/data can be saved.

Figure C.23: Workspace user story

Unread results

To avoid missing results
the researchers
want to see which results are unread.

Figure C.24: Unread results user story

Preview of file

To correctly annotatate a file
the researchers
want to preview a portion of a file.

Figure C.25: Preview of file user story

Work scheduling

To strategically spread the servers workload over time
the researchers
want to be able to schedule the processing/analysis of data.

Figure C.26: Work scheduling user story

Time estimation

To warn for time consuming jobs
the researchers
want to have a time estimation for jobs.

Figure C.27: Time estimation user story

Plot overlap analysis

To see region overlap of genomes
the researchers
want to plot an overlap analysis (see separate user story).

Figure C.28: Plot overlap analysis user story

Plot average regions

To view data
the researchers
want to plot average of regions with the profile data.

Figure C.29: Plot average regions user story

IGB Session

To be able to make IGB analysis
the researchers
want to retrieve a IGB session file.

Figure C.30: IGB Session user story

Combine regions

To find interesting regions
the researches
want to select multiple files and combine their regions (union, intersect).

Figure C.31: Combine regions user story

Create region subset

To retrieve certain parts of regions
the researchers
want to create region subsets using reference points.

Figure C.32: Create region subset user story

Calculate average of region

To find the average protein binding value for a region
the researchers
want to calculate average of regions with the profile data. (Possibly split into a number of bins).

Figure C.33: Calculate average of region user story

Overlap analysis
To conduct overlap analysis the researchers want to divide regions bins, either by value or by order.

Figure C.34: Overlap analysis user story

Save analysis results
To be able to return to previous work the researchers want to save analysis results(in workspace).

Figure C.35: Save analysis results user story

D Android application: *UML*-diagrams

In this appendix the *UML*-Class-diagrams of the Android application will be presented.

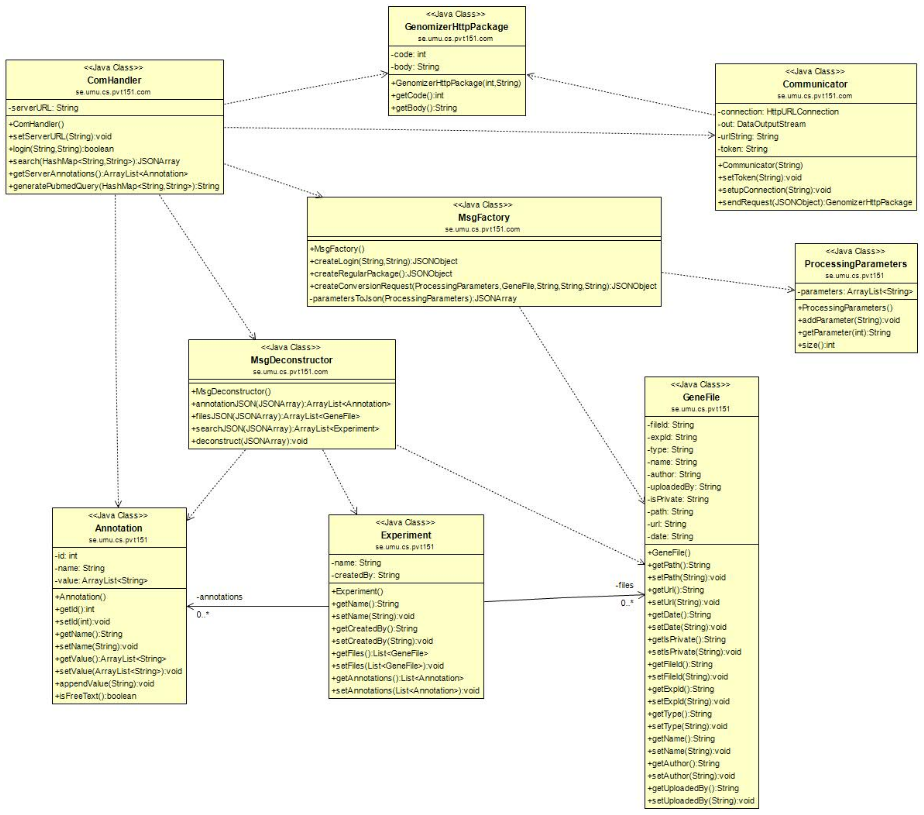


Figure D.1: Android UML of model

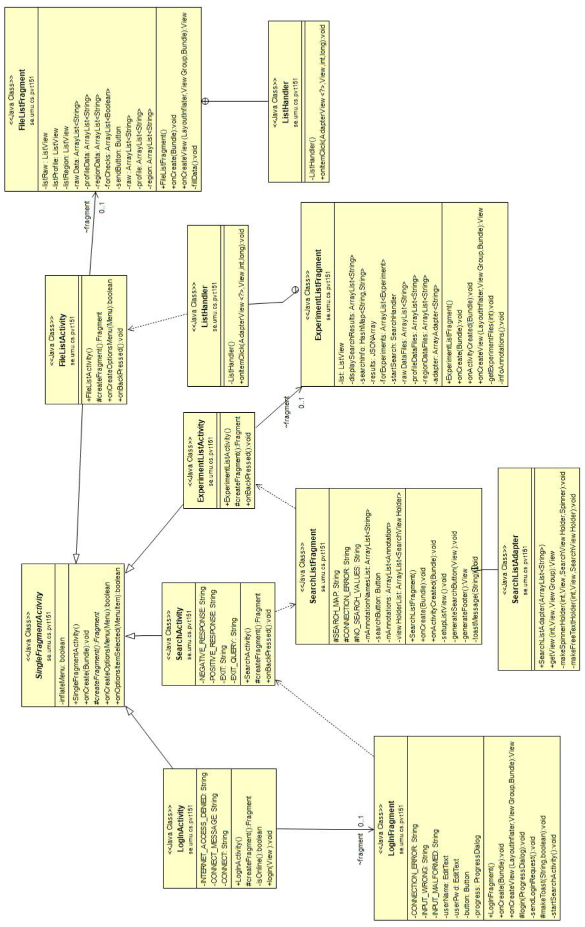


Figure D.2: Android UML without model

E iOS application: *UML*-diagrams

In this appendix the *UML*-Class-diagrams of the iOS application will be presented.

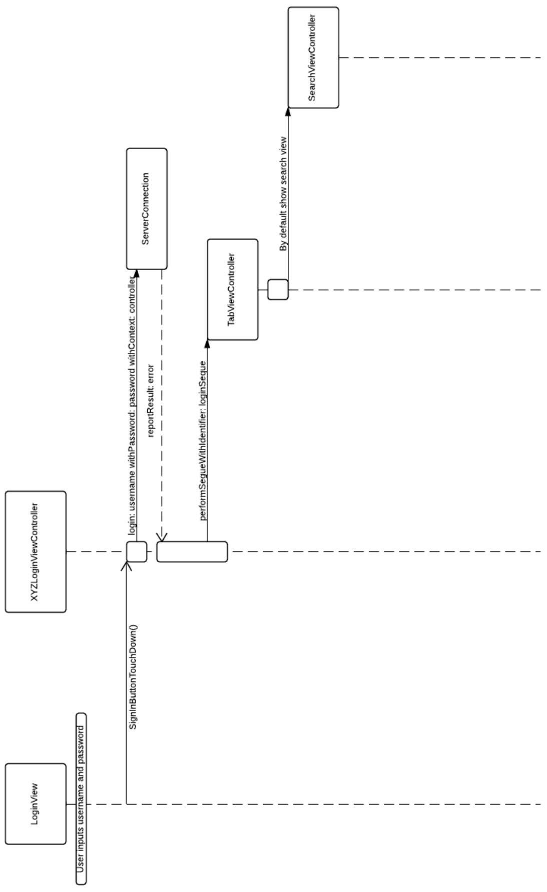

LoginView
User inputs username and password
XYZLoginViewController
SignInButtonTouchDown()
login: username withPassword: password withContext: controller
ServerConnection
reportResult: error
performSegueWithIdentifier: loginSegue
TabViewController
By default show search view
SearchViewController

Figure E.1: Login sequence diagram.

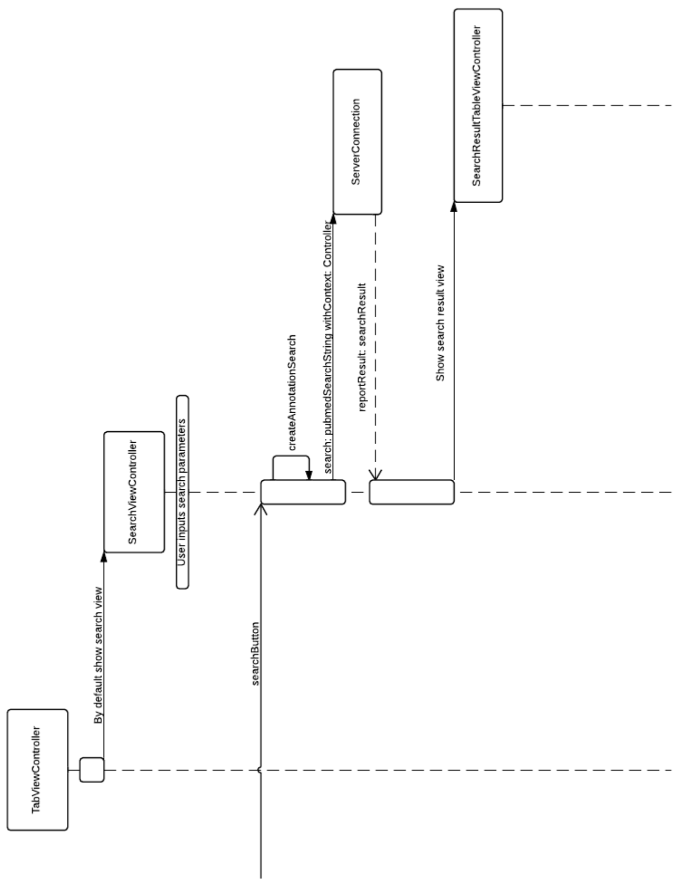

TabViewController
By default show search view
SearchViewController
User inputs search parameters
searchButton
createAnnotationSearch
search: pubmedSearchString withContext: Controller
ServerConnection
reportResult: searchResult
Show search result view
SearchResultTableViewController

Figure E.2: Search sequence diagram.

F Desktop application: *UML*-diagrams

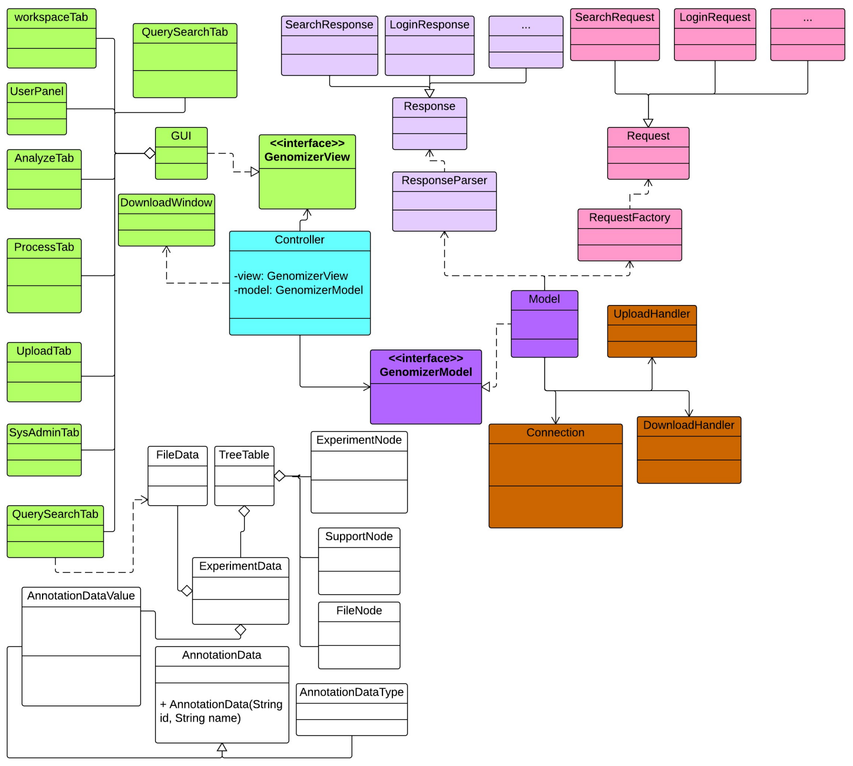


Figure F.1: `Desktop Overview` with major class relations.

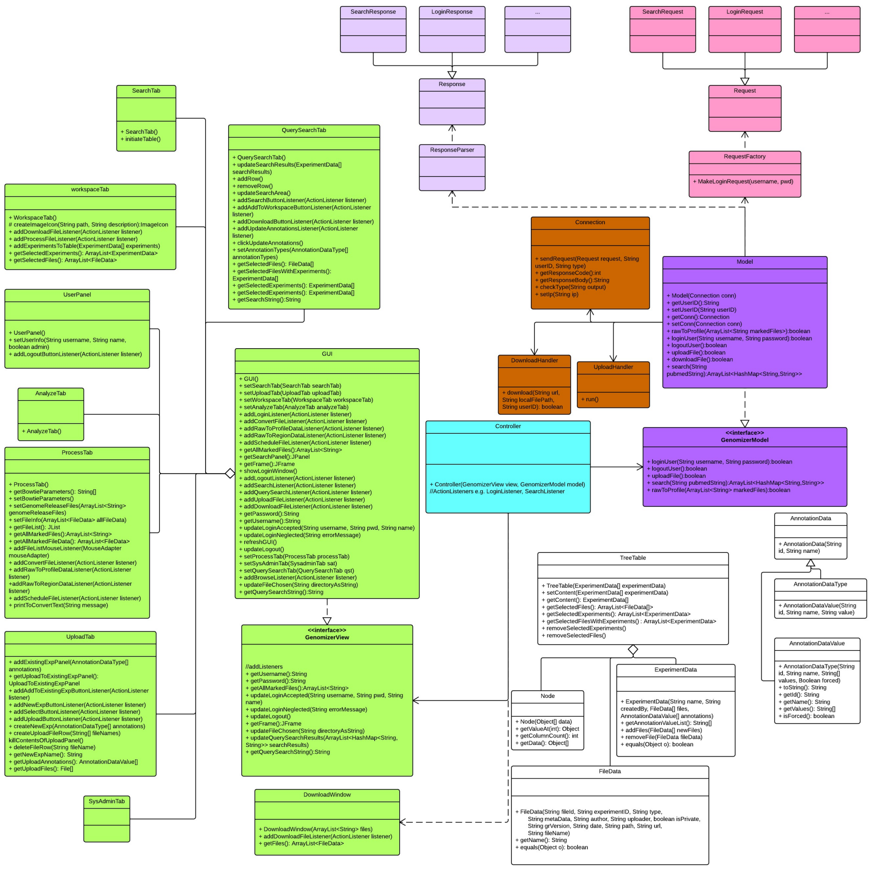

SearchResponse
LoginResponse
...
SearchRequest
LoginRequest
...
Response
Request
ResponseParser
RequestFactory
+ MakeLoginRequest(username, pwd)
SearchTab
+ SearchTab()
+ initiateTable()
QuerySearchTab
+ QuerySearchTab()
+ updateSearchResults(ExperimentData[] searchResults)
+ addRow()
+ removeRow()
+ updateSearchArea()
+ addSearchButtonListener(ActionListener listener)
+ addAddToWorkspaceButtonListener(ActionListener listener)
+ addDownloadButtonListener(ActionListener listener)
+ addUpdateAnnotationsListener(ActionListener listener)
+ clickUpdateAnnotations()
+ setAnnotationTypes(AnnotationDataType[] annotationTypes)
+ getSelectedFiles(): FileData[]
+ getSelectedFilesWithExperiments(): ExperimentData[]
+ getSelectedExperiments(): ExperimentData[]
+ getSelectedExperiments(): ExperimentData[]
+ getSearchString():String
workspaceTab
+ WorkspaceTab()
# createImageIcon(String path, String description):ImageIcon
+ addDownloadFileListener(ActionListener listener)
+ addProcessFileListener(ActionListener listener)
+ addExperimentsToTable(ExperimentData[] experiments)
+ getSelectedExperiments(): ArrayList<ExperimentData>
+ getSelectedFiles(): ArrayList<FileData>
Connection
+ sendRequest(Request request, String userID, String type)
+ getResponseCode():int
+ getResponseBody():String
+ checkType(String output)
+ setIp(String ip)
Model
+ Model(Connection conn)
+ getUserID():String
+ setUserID(String userID)
+ getConn():Connection
+ setConn(Connection conn)
+ rawToProfile(ArrayList<String markedFiles>):boolean
+ loginUser(String username, String password):boolean
+ logoutUser():boolean
+ uploadFile():boolean
+ downloadFile():boolean
+ search(String pubmedString):ArrayList<HashMap<String,String>>
UserPanel
+ UserPanel()
+ setUserInfo(String username, String name, boolean admin)
+ addLogoutButtonListener(ActionListener listener)
GUI
+ GUI()
+ setSearchTab(SearchTab searchTab)
+ setUploadTab(UploadTab uploadTab)
+ setWorkspaceTab(WorkspaceTab workspaceTab)
+ setAnalyzeTab(AnalyzeTab analyzeTab)
+ addLoginListener(ActionListener listener)
+ addConvertFileListener(ActionListener listener)
+ addRawToProfileDataListener(ActionListener listener)
+ addRawToRegionDataListener(ActionListener listener)
+ addScheduleFileListener(ActionListener listener)
+ getAllMarkedFiles():ArrayList<String>
+ getSearchPanel():JPanel
+ getFrame():JFrame
+ showLoginWindow()
+ addLogoutListener(ActionListener listener)
+ addSearchListener(ActionListener listener)
+ addQuerySearchListener(ActionListener listener)
+ addUploadFileListener(ActionListener listener)
+ addDownloadFileListener(ActionListener listener)
+ getPassword():String
+ getUsername():String
+ updateLoginAccepted(String username, String pwd, String name)
+ updateLoginNeglected(String errorMessage)
+ refreshGUI()
+ updateLogout()
+ setProcessTab(ProcessTab processTab)
+ setSysAdminTab(SysadminTab sat)
+ setQuerySearchTab(QuerySearchTab qst)
+ addBrowseListener(ActionListener listener)
+ updateFileChosen(String directoryAsString)
+ getQuerySearchString():String
DownloadHandler
+ download(String url, String localFilePath, String userID): boolean
UploadHandler
+ run()
AnalyzeTab
+ AnalyzeTab()
Controller
+ Controller(GenomizerView view, GenomizerModel model)
//ActionListeners e.g. LoginListener, SearchListener
<<interface>> GenomizerModel
+ loginUser(String username, String password):boolean
+ logoutUser():boolean
+ uploadFile():boolean
+ search(String pubmedString):ArrayList<HashMap<String,String>>
+ rawToProfile(ArrayList<String> markedFiles):boolean
ProcessTab
+ ProcessTab()
+ getBowtieParameters(): String[]
+ setBowtieParameters()
+ setGenomeReleaseFiles(ArrayList<String> genomeReleaseFiles)
+ setFileInfo(ArrayList<FileData> allFileData)
+ getFileList(): JList
+ getAllMarkedFiles():ArrayList<String>
+ getAllMarkedFileData(): ArrayList<FileData>
+ addFileListMouseListener(MouseAdapter mouseAdapter)
+ addConvertFileListener(ActionListener listener)
+ addRawToProfileDataListener(ActionListener listener)
+addRawToRegionDataListener(ActionListener listener)
+ addScheduleFileListener(ActionListener listener)
+ printToConvertText(String message)
AnnotationData
+ AnnotationData(String id, String name)
TreeTable
+ TreeTable(ExperimentData[] experimentData)
+ setContent(ExperimentData[] experimentData)
+ getContent(): ExperimentData[]
+ getSelectedFiles(): ArrayList<FileData[]>
+ getSelectedExperiments(): ArrayList<ExperimentData>
+ getSelectedFilesWithExperiments() : ArrayList<ExperimentData>
+ removeSelectedExperiments()
+ removeSelectedFiles()
AnnotationDataType
+ AnnotationDataValue(String id, String name, String value)
AnnotationDataValue
+ AnnotationDataType(String id, String name, String[] values, Boolean forced)
+ toString(): String
+ getId(): String
+ getName(): String
+ getValues(): String[]
+ isForced(): boolean
UploadTab
+ addExistingExpPanel(AnnotationDataType[] annotations)
+ getUploadToExistingExpPanel(): UploadToExistingExpPanel
+ addAddToExistingExpButtonListener(ActionListener listener)
+ addNewExpButtonListener(ActionListener listener)
+ addSelectButtonListener(ActionListener listener)
+ addUploadButtonListener(ActionListener listener)
+ createNewExp(AnnotationDataType[] annotations)
+ createUploadFileRow(String[] fileNames)
killContentsOfUploadPanel()
+ deleteFileRow(String fileName)
+ getNewExpName(): String
+ getUploadAnnotations(): AnnotationDataValue[]
+ getUploadFiles(): File[]
<<interface>> GenomizerView
//addListeners
+ getUsername():String
+ getPassword():String
+ getAllMarkedFiles():ArrayList<String>
+ updateLoginAccepted(String username, String pwd, String name)
+ updateLoginNeglected(String errorMessage)
+ updateLogout()
+ getFrame():JFrame
+ updateFileChosen(String directoryAsString)
+ updateQuerySearchResults(ArrayList<HashMap<String, String>> searchResults)
+ getQuerySearchString():String
Node
+ Node(Object[] data)
+ getValueAt(int): Object
+ getColumnCount(): int
+ getData(): Object[]
ExperimentData
+ ExperimentData(String name, String createdBy, FileData[] files, AnnotationDataValue[] annotations)
+ getAnnotationValueList(): String[]
+ addFiles(FileData[] newFiles)
+ removeFile(FileData fileData)
+ equals(Object o): boolean
SysAdminTab
DownloadWindow
+ DownloadWindow(ArrayList<String> files)
+ addDownloadFileListener(ActionListener listener)
+ getFiles(): ArrayList<FileData>
FileData
+ FileData(String fileId, String experimentID, String type, String metaData, String author, String uploader, boolean isPrivate, String grVersion, String date, String path, String url, String fileName)
+ getName(): String
+ equals(Object o): boolean

Figure F.2: `Desktop Overview` with major class relations.

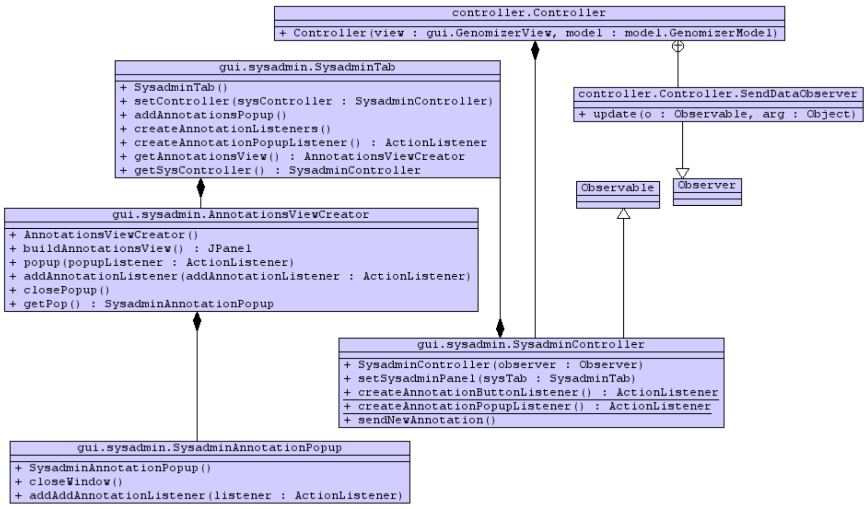

controller.Controller
+ Controller(view : gui.GenomizerView, model : model.GenomizerModel)
gui.sysadmin.SysadminTab
+ SysadminTab()
+ setController(sysController : SysadminController)
+ addAnnotationsPopup()
+ createAnnotationListeners()
+ createAnnotationPopupListener() : ActionListener
+ getAnnotationsView() : AnnotationsViewCreator
+ getSysController() : SysadminController
controller.Controller.SendDataObserver
+ update(o : Observable, arg : Object)
Observable
Observer
gui.sysadmin.AnnotationsViewCreator
+ AnnotationsViewCreator()
+ buildAnnotationsView() : JPanel
+ popup(popupListener : ActionListener)
+ addAnnotationListener(addAnnotationListener : ActionListener)
+ closePopup()
+ getPop() : SysadminAnnotationPopup
gui.sysadmin.SysadminController
+ SysadminController(observer : Observer)
+ setSysadminPanel(sysTab : SysadminTab)
+ createAnnotationButtonListener() : ActionListener
+ createAnnotationPopupListener() : ActionListener
+ sendNewAnnotation()
gui.sysadmin.SysadminAnnotationPopup
+ SysadminAnnotationPopup()
+ closeWindow()
+ addAddAnnotationListener(listener : ActionListener)

Figure F.3: The `Controller` and `Sysadmin` class relations.

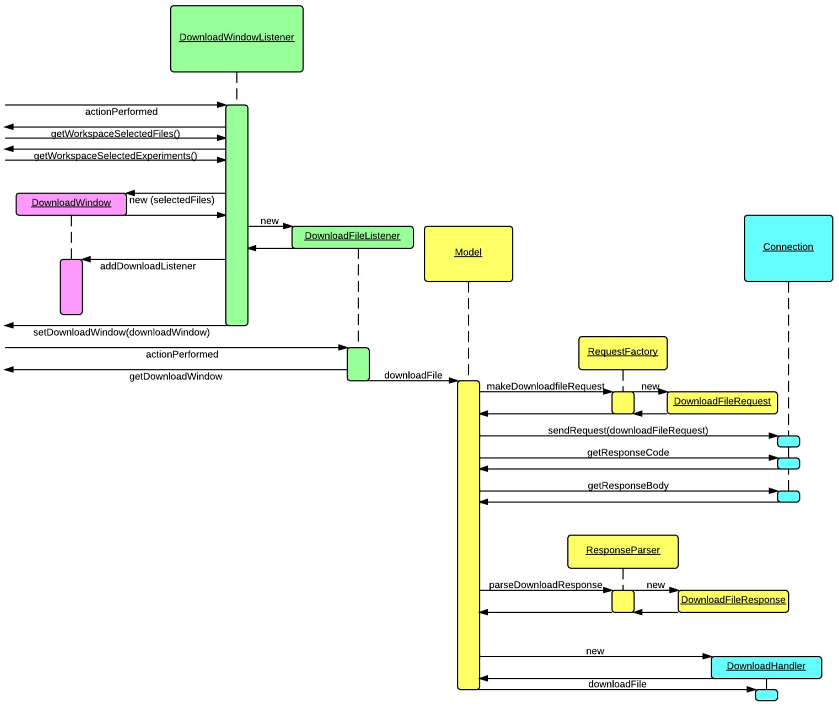


Figure F.4: The download sequence.

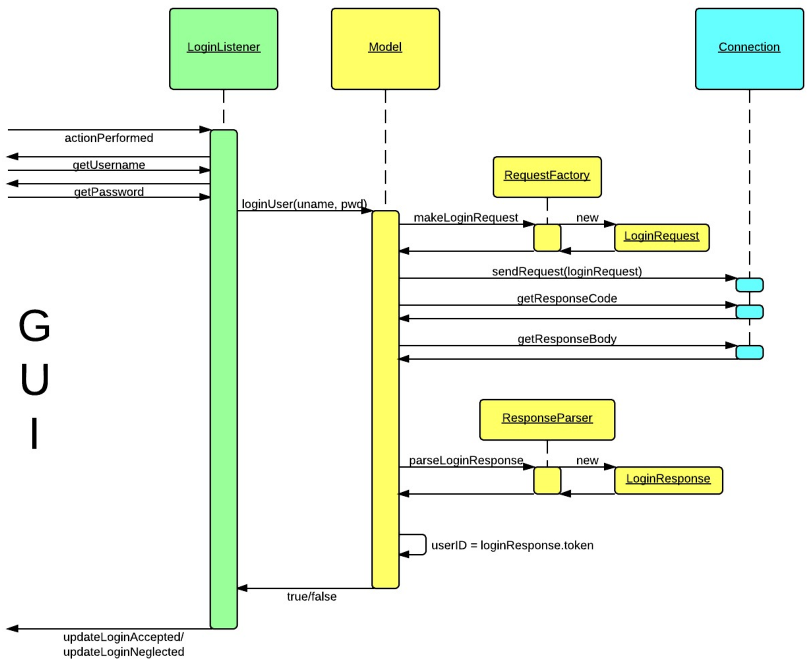


Figure F.5: The login sequence.

G Data Storage: *UML*-diagrams

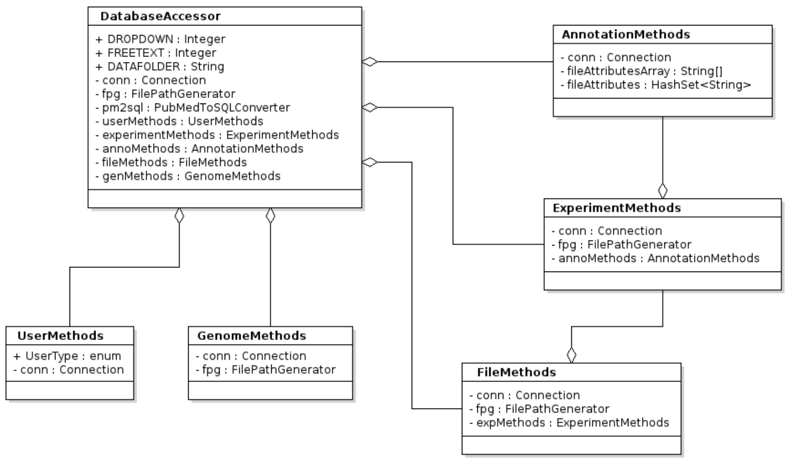

DatabaseAccessor
+ DROPDOWN : Integer
+ FREETEXT : Integer
+ DATAFOLDER : String
- conn : Connection
- fpg : FilePathGenerator
- pm2sql : PubMedToSQLConverter
- userMethods : UserMethods
- experimentMethods : ExperimentMethods
- annoMethods : AnnotationMethods
- fileMethods : FileMethods
- genMethods : GenomeMethods
AnnotationMethods
- conn : Connection
- fileAttributesArray : String[]
- fileAttributes : HashSet<String>
ExperimentMethods
- conn : Connection
- fpg : FilePathGenerator
- annoMethods : AnnotationMethods
UserMethods
+ UserType : enum
- conn : Connection
GenomeMethods
- conn : Connection
- fpg : FilePathGenerator
FileMethods
- conn : Connection
- fpg : FilePathGenerator
- expMethods : ExperimentMethods

Figure G.1: The wrapper class `DatabaseAccessor` and the method subclasses

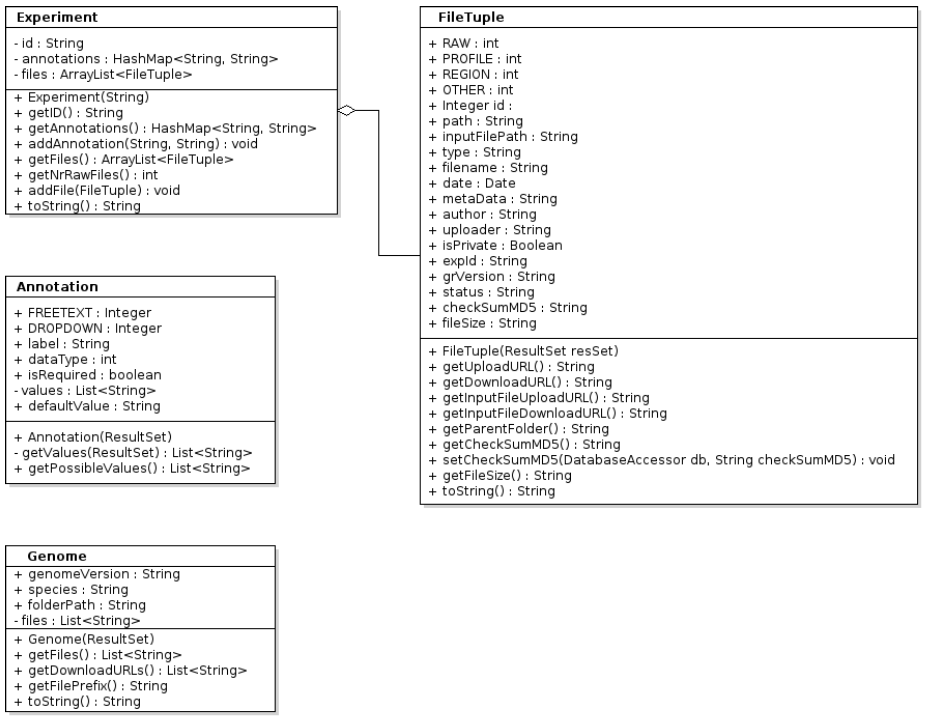


Figure G.2: The container classes used to return information

H Server: Communication UML

H.1 Overview

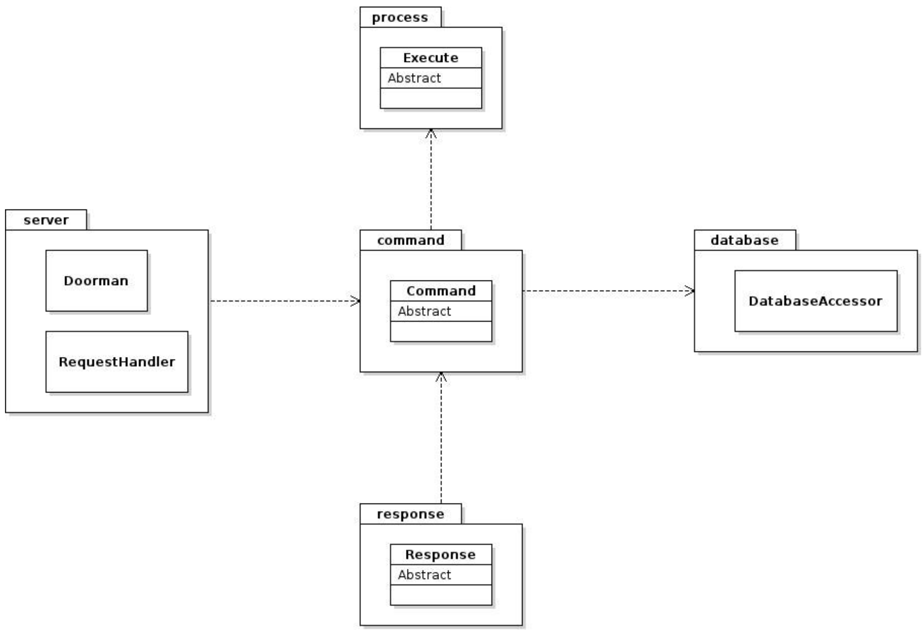


Figure H.1: Overview of the classes in communication parts

H.2 Request

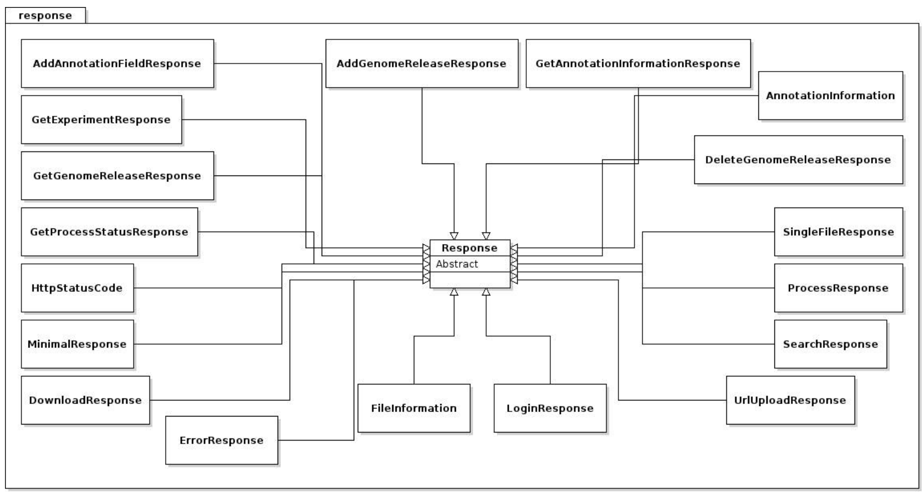


Figure H.2: Classes making the response package.

H.3 Command

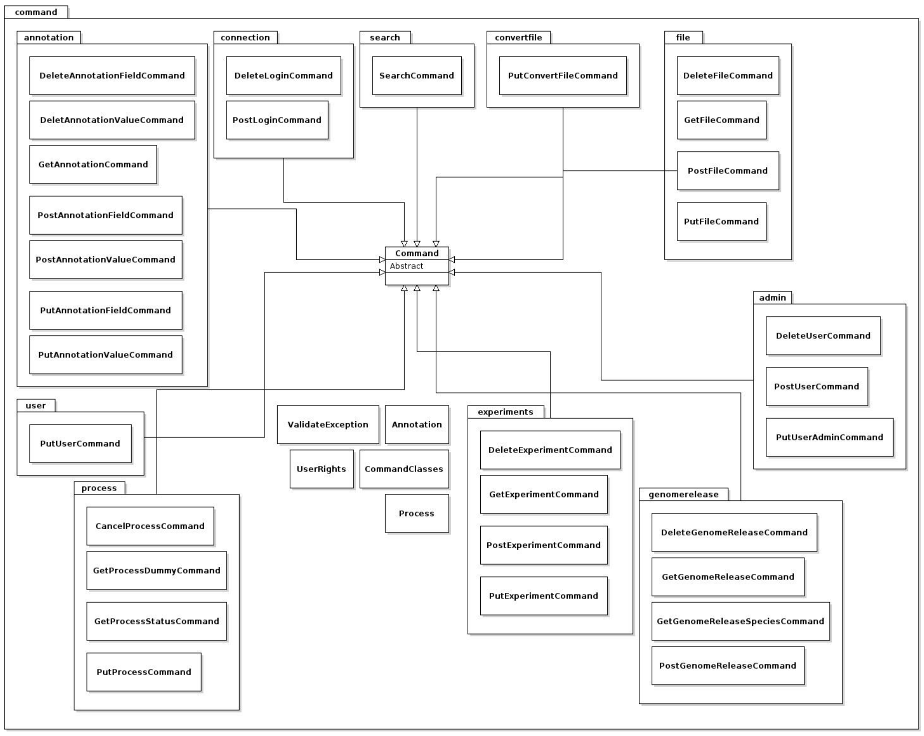


Figure H.3: Classes used for requests.

I Server API

The server relies on REST to handle communication between the server and the clients. There are a set of valid requests that can be made, this section encompasses all of them. They are divided into respective categories, those being Connection, Experiment, File, File conversion, Search, User, Admin, Processing, Annotation handling, Genome release handling, GEO and File upload/download.

All requests are briefly explained and example requests and responds are given.

Connection

Requests used to login/logout to and from the server.

POST /login

Logs in as a user.

Request:

```
POST /login HTTP/1.1
Content-Type: application/json
{
 "username": "uname",
 "password": "pw"
}
```

Response:

```
200 (OK)
Content-Type: application/json
{
 "token": "token",
 "role": "role"
}
```

DELETE /login

Logs out as a user.

Request:

```
DELETE /login HTTP/1.1
authorization: token
```

Response:

```
200 (OK)
```

Experiment

Requests used to add, retrieve, update and delete experiments.

POST /experiment

Creates a new experiment.

Request:

```
POST /experiment HTTP/1.1
Content-Type: application/json
Authorization: token
{
    "name": "experimentId",
    "annotations":
                 [
                 {
                  "name": "pubmedId",
                  "value": "abc123"
                 },
                 {
                  "name": "type",
                  "value": "raw"
                 },
                 {
                  "name": "specie",
                  "value": "human"
                 },
                 {
                  "name": "genome release",
                  "value": "v.123"
                 },
                 {
                  "name": "cell line",
                  "value": "yes"
                 },
                 {
                  "name": "development stage",
                  "value": "larva"
                 },
                 {
                  "name": "sex",
                  "value": "male"
                 },
                 {
                  "name": "tissue",
                  "value": "eye"
                 }
                 ]
}
```

Response:

```
200 (OK)
```

GET /experiment/<experiment-id>

Retrieves a specific experiment.

Request:

```
GET /experiment/<experiment-id> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
Authorization: token
{
    "name": "experimentId",
    "files": [
            {
             "id": "id",
             "path": "path",
             "url": "url",
             "type": "type",
             "filename": "filename",
             "date": "date",
             "metaData": "metaData",
             "author": "author",
             "uploader": "uploader",
             "expId": "expId",
             "grVersion": "realseNr",
             "fileSize": "fileSize"
            },
            {
             "id": "id",
             "path": "path",
             "url": "url",
             "type": "type",
             "filename": "filename",
             "date": "date",
             "metaData": "metaData",
             "author": "author",
             "uploader": "uploader",
             "expId": "expId",
             "grVersion": "realseNr",
             "fileSize": "fileSize"
            }
             ],
    "annotations":
                 [
                 {
                  "name": "pubmedId",
                  "value": "abc123"
                 },
                 {
                  "name": "type",
                  "value": "raw"
                 },
                 {
                  "name": "specie",
                  "value": "human"
                 },
                 {
                  "name": "genome release",
                  "value": "v.123"
                 },
                 {
                  "name": "cell line",
                  "value": "yes"
                 },
                 {
                  "name": "development stage",
                  "value": "larva"
                 },
                 {
                  "name": "sex",
                  "value": "male"
                 },
                 {
                  "name": "tissue",
                  "value": "eye"
                 }
                 ]

}
```

PUT /experiment/<experiment-id>

Edits an exisiting experiment.

Request:

```
PUT /experiment/<experiment-id> HTTP/1.1
Content-Type: application/json
Authorization: token
{
    "name": "experimentId",
    "annotations":
                 [
                 {
                  "name": "pubmedId",
                  "value": "abc123"
                 },
                 {
                  "name": "type",
                  "value": "raw"
                 },
                 {
                  "name": "specie",
                  "value": "human"
```

```
                },
                {
                 "name": "genome release",
                 "value": "v.123"
                },
                {
                 "name": "cell line",
                 "value": "yes"
                },
                {
                 "name": "development stage",
                 "value": "larva"
                },
                {
                 "name": "sex",
                 "value": "male"
                },
                {
                 "name": "tissue",
                 "value": "eye"
                }
                ]
    }
```

Response:

```
200 (OK)
```

DELETE /experiment/<experiment-id>

Deletes an existing experiment.

Request:

```
DELETE /experiment/<experiment-id> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
```

File

Requests ued to add, retrieve (file information), update and delete files.

POST /file

Adds a file to an experiment.

Request:

```
POST /file HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "experimentID": "id",
 "fileName": "name",
 "type": "raw",
 "metaData": "metameta",
 "author": "name",
 "grVersion": "releaseNr"
 "checkSumMD5": "checksum"
}
```

Response:

```
200 (OK)
```

GET /file/<file-id>

Retrieves file information about a specific file.

Request:

```
GET /file/<file-id> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
{
 "id": "id",
 "path": "path",
 "url": "url",
 "type": "type",
 "filename": "filename",
 "date": "date",
 "metaData": "metaData",
 "author": "author",
 "uploader": "uploader",
 "expId": "expId",
 "grVersion": "realseNr",
 "fileSize": "fileSize",
 "checkSumMD5": "checkSumMD5"
}
```

PUT /file/<file-id>

Edits file information for a specific file.

Request:

```
PUT /file/<file-id> HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "experimentID": "id",
 "filename": "name",
 "type": "raw",
 "metaData": "metameta",
 "author": "name",
 "grVersion": "releaseNr"
 "checkSumMD5": "checksum"
}
```

Response:

```
200 (OK)
```

DELETE /file/<file-id>

Deletes a specific file.

Request:

```
DELETE /file/<file-id> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
```

File conversion

Requests used to handle file conversion.

PUT /convertfile

Converts a file to a given format.

Request:

```
PUT /convertfile HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "fileid": "id",
 "toformat": "format"
}
```

Response:

```
200 (OK)
Content-Type: application/json
    {
     "id": "id",
     "path": "path",
     "url": "url",
     "type": "type",
     "filename": "filename",
     "date": "date",
     "metaData": "metaData",
     "author": "author",
     "uploader": "uploader",
     "expId": "expId",
     "grVersion": "realseNr",
     "fileSize": "fileSize",
     "checkSumMD5": "checkSumMD5"
    }
```

Search

Requests used to handle user information.

PUT /user

Edits the user information.

Request:

```
PUT /user HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "oldPassword": "oldPw"
 "newPassword": "newPw",
 "name": "John Johnson",
 "email": "john@mail.com"
}
```

Response:

```
200 (OK)
```

GET /user/<username>

Request:

```
GET /user/<username> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
    {
     "username": "myusername"
     "privileges": "ADMIN",
     "name": "John Johnson",
     "email": "john@mail.com"
    }
```

Search

Requests used to handle searching for experiments.

GET /search/?annotations=<pubmedStyleQuery>

Searches for an experiment using a pubmed query.

Request:

```
GET /search/?annotations=<pubmedStyleQuery> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
[
   {
        "name": "experimentId",
        "files": [
                {
                 "id": "id",
                 "path": "path",
                 "url": "url",
                 "type": "type",
                 "filename": "filename",
                 "date": "date",
                 "metaData": "metaData",
                 "author": "author",
                 "uploader": "uploader",
                 "expId": "expId",
                 "grVersion": "realseNr",
                 "fileSize": "fileSize"
                },
                {
                 "id": "id",
                 "path": "path",
                 "url": "url",
                 "type": "type",
                 "filename": "filename",
                 "date": "date",
                 "metaData": "metaData",
                 "author": "author",
                 "uploader": "uploader",
                 "expId": "expId",
                 "grVersion": "realseNr",
                 "fileSize": "fileSize"
                }
                 ],
        "annotations":
                     [
                     {
                      "name": "pubmedId",
                      "value": "abc123"
                     },
                     {
                      "name": "type",
                      "value": "raw"
                     },
                     {
                      "name": "specie",
                      "value": "human"
                     },
                     {
                      "name": "genome release",
                      "value": "v.123"
                     },
                     {
                      "name": "cell line",
                      "value": "yes"
                     },
                     {
                      "name": "development stage",
                      "value": "larva"
                     },
                     {
                      "name": "sex",
                      "value": "male"
                     },
                     {
                      "name": "tissue",
                      "value": "eye"
                     }
                     ]

    }
]
```

Processing

Requests used to process experiment data.

GET /process

Gets the statuses of all processes.

Request:

```
GET /process HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
[
  {
    "experimentName": "Exp1",
    "PID": "123",
    "status": "Finished",
    "outputFiles": [
      "file1",
      "file2"
    ],
    "author": "yuri",
    "timeAdded": 1400245668744,
    "timeStarted": 1400245668756,
    "timeFinished": 1400245669756
  },
  {
    "experimentName": "Exp2",
    "PID": "124",
    "status": "Finished",
    "outputFiles": [
      "file1",
      "file2"
    ],
    "author": "janne",
    "timeAdded": 1400245668746,
    "timeStarted": 1400245669756,
    "timeFinished": 1400245670756
  },
  {
    "experimentName": "Exp43",
    "PID": "125",
    "status": "Crashed",
    "outputFiles": [
      "file1",
      "file2"
    ],
    "author": "philge",
    "timeAdded": 1400245668748,
    "timeStarted": 1400245670756,
    "timeFinished": 1400245671757
  },
  {
    "experimentName": "Exp234",
    "PID": "126",
    "status": "Started",
    "outputFiles": [
      "file1",
      "file2"
    ],
    "author": "per",
    "timeAdded": 1400245668753,
    "timeStarted": 1400245671757,
    "timeFinished": 0
  },
  {
    "experimentName": "Exp6",
    "PID": "127",
    "status": "Waiting",
    "outputFiles": [],
    "author": "yuri",
    "timeAdded": 1400245668755,
    "timeStarted": 0,
    "timeFinished": 0
  }
]
```

DELETE /process

Cancels a running process.

Request:

```
DELETE /process HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "PID": "PID"
}
```

Response:

```
200 (OK)
```

PUT /process/rawtoprofile

Starts a raw to profile processing.

Request:

```
PUT /process/rawtoprofile HTTP/1.1
Content-Type: application/json
Authorization: token
{
```

```
"expid": "expID",
"parameters":
            [
             "Bowtie parameters",
             "",
             "y",
             "",
             "10 1 5 0 0",
             "y 10",
             "single 4 0",
             "150 1 7 0 0"
            ],
"metadata": "metaData",
"genomeVersion": "hg38"
}
```

Response:

```
200 (OK)
```

PUT /process/processCommands

Lists files and processes to run on them.

Request:

```
PUT /process/processCommands HTTP/1.1
Content-Type: application/json
Authorization: token
{
    "expId": ...,
    "processCommands": [
        {
            "type": "bowtie",
            "files": [
                {
                    "infile": ...,
                    "outfile": ...,
                    "params": ...,
                    "genomeVersion": ...,
                    "keepSam": true/false
                },
                ...
            ]
        },
        {
            "type": "ratio",
            ...
            (not supported)
        },
        ...
    ]
}
```

Response:

```
200 (OK)
```

Annotation handling

Requests used to handle annotations.

GET /annotation

Gets all existing annotations.

Request:

```
GET /annotation HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
[
 {
  "name": "pubmedId",
  "values": ["freetext"],
  "forced": true
 },
 {
  "name": "type",
  "values": ["freetext"],
  "forced": true
 },
 {
  "name": "specie",
  "values": ["fly", "human", "rat"],
  "forced": true
 },
 {
  "name": "genome release",
  "values": ["freetext"],
  "forced": true
 },
 {
  "name": "cell line",
  "values": ["yes", "no"],
  "forced": true
 },
 {
  "name": "development stage",
  "values": ["larva", "larvae"],
  "forced": true
 },
 {
  "name": "sex",
  "values": ["male", "female", "unknown"],
  "forced": true
 },
 {
  "name": "tissue",
  "values": ["eye", "leg"],
  "forced": false
 }
]
```

POST /annotation/field

Adds an annotation field.

Request:

```
POST /annotation/field HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "name": "species",
 "type": [
          "fly",
          "rat",
          "human"
         ],
 "default": "human",
 "forced": false
}
```

Response:

```
200 (OK)
```

PUT /annotation/field

Renames an annotation field.

I.0.0.1 Request:

```
PUT /annotation/field HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "oldName": "species",
 "newName": "mouse"
}
```

Response:

```
200 (OK)
```

DELETE /annotation/field/<field-name>

Deletes an annotation field.

Request:

```
DELETE /annotation/field/<field-name> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
```

POST /annotation/value

Adds an annotation value.

Request:

```
POST /annotation/value HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "name": "species",
 "value": "mouse"
}
```

Response:

```
200 (OK)
```

PUT /annotation/value

Renames an annotation value.

Request:

```
PUT /annotation/value HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "name": "species",
 "oldValue": "mouse",
 "newValue": "rat"
}
```

Response:

```
200 (OK)
```

DELETE /annotation/value/<field-name>/<value-name>

Request:

```
DELETE /annotation/value/<field-name>/<value-name> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
```

Genome release handling

Requests used to add, get, and delete genome releases.

GET /genomeRelease

Retrieves all genome releases.

Request:

```
GET /genomeRelease HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
[
 {
  "genomeVersion": "hy17",
  "species": "fly",
  "folderPath": "pathToVersion",
  "files": ["filename1", "filename2", "filename3"]
 },
 {
  "genomeVersion": "u12b",
  "species": "human",
  "folderPath": "pathToVersion",
  "files": ["filename1", "filename2"]
 },
 {
  "genomeVersion": "wk1m",
  "species": "human",
  "folderPath": "pathToVersion",
  "files": ["filename1", "filename2"]
 },
 {
  "genomeVersion": "fg2b",
  "species": "rat",
  "folderPath": "pathToVersion",
  "files": ["filename1", "filename2"]
 },
 {
  "genomeVersion": "abc1",
  "species": "rat",
  "folderPath": "pathToVersion",
  "files": ["filename1", "filename2"]
 }
]
```

POST /genomeRelease

Adds a genome release.

Request:

```
POST /genomeRelease HTTP/1.1
Content-Type: application/json
Authorization: token
{
 "genomeVersion": "hx16",
 "specie": "human",
 "files":
        [
         "nameOfFile1",
         "nameOfFile2",
         "nameOfFile3"
        ],
 "checkSumMD5":
              [
               "checkSum1",
               "checkSum2",
               "checkSum3"
              ]
}
```

Response:

```
200 (OK)
Content-Type: application/json
[
 {
  "URLupload": "url1"
 },
 {
  "URLupload": "url2"
 },
 {
  "URLupload": "url3"
 }
]
```

GET /genomeRelease/<species>

Retrieves the genome releases for a specific species.

Request:

```
GET /genomeRelease/<species> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Type: application/json
 [
  {
   "genomeVersion": "fg2b",
   "species": "rat",
   "folderPath": "pathToVersion",
   "files": ["filename1", "filename2"]
  },
  {
   "genomeVersion": "abc1",
   "species": "rat",
   "folderPath": "pathToVersion",
   "files": ["filename1", "filename2"]
  }
 ]
```

DELETE /genomeRelease/<species>/<genomeVersion>

Deletes a genome release.

Request:

```
DELETE /genomeRelease/<species>/<genomeVersion> HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
```

File upload/download

Requests used to upload and download files.

POST /upload?path=<path-on-server>

Uploads a file to the server.

Request:

```
POST /upload?path=/path/on/server/foo.bar HTTP/1.1
Authorization: token
Content-Type: multipart/form-data; boundary=---------------------------9051914041544843365972754266
Content-Length: 12345
---------------------------9051914041544843365972754266
Content-Disposition: form-data; name="file1"; filename="foo.txt"
Content-Type: text/plain

Contents of foo.txt.
```

Response:

```
201 (Created)
```

GET /download?path=<path-on-server>

Downloads a file from the server.

Request:

```
GET /download?path=/path/on/server/foo.bar HTTP/1.1
Authorization: token
```

Response:

```
200 (OK)
Content-Description: File Transfer
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=foo.bar
Expires: 0
Cache-Control: must-revalidate
Pragma: public
Header: foo
[contents of foo.bar]
```

J *Backup*

J.1 Introduction

To backup the genomizer-server `rsync` and `crontab` are required. The backup can either be stored on an external storage device (e.g hard drive or USB-storage) connected to the genomizer-server or a remote backup-server.

Using a remote server requires setting up `ssh` keys for the genomizer-server to remote backup communication. This can be achieved using `ssh-copy-id` or equivalent.

Keep in mind that `rsync` synchronizes the chosen directory trees on the genomizer-server and the backup-server. This means that all deleted files on the genomizer-server will be deleted on the backup-server.

In addition to the `rsync` setup, we include a script intended for use on the backup-server, which packs the synchronized directory tree into a compressed archive. If added to the backup-servers `crontab`, this script will automate the process, and also clean up old archives.

All backup scripts are located at `resources/backup/` in the genomizer-server project folder.

J.2 File backup

Two scripts are available to perform the directory synchronization part of the backup of the genomizer-server. These are named `local_file_backup.sh` and `remote_file_backup.sh`, and correspond to the local or remote storage options mentioned in the previous section.

`local_file_backup.sh` will backup to a local path on the genomizer-server. Suggested use is to mount an external storage device somewhere on the filesystem, and to backup to that.

Conversely, `remote_file_backup.sh` will backup to a remote backup-server. This is where the `ssh` keys are required.

Both scripts need to be edited to work properly. The following variables must be set:

- READPATH: Path to the directory tree that you wish to backup
- SAVEPATH: Path to the synchronize to

The script `remote_file_backup.sh` also requires the following variables to be set:

- PORT: The `ssh` port on the backup-server
- USER: The user identity to use when connecting to the backup-server
- IP: IP-address to the backup-server

J.2.1 Archiving

The archiving script, `backup_tar.sh`, compresses a given directory tree into a `gzip`'d tarball. This script should be invoked at the backup site. This is easily automated by adding the script to an appropriate `crontab` at the backup site, and setting the script to remove old archives at regular intervals.

The following variables need to be edited in the script:

- WORKDIR: Path to the parent folder of the directory to which the genomizer-server is backed up.[1]
- READFOLDER: Name of the directory in "WORKDIR" to which the genomizer-server is backed up.
- SAVEPATH: Path to the directory in which to store archives
- DAYS: Specifies how many days old archives should be stored before the script deletes them

J.3 Database backup

To backup the database a script called `db_backup.sh` is available. The following variables in the script need to be edited:

- DBNAME: Name of the database
- DBPORT: Port of the databse
- DBUSER: Username for the database
- FILENAME: Name to give to the created backup file
- SAVEPATH: Path to the directory in which to store the backup file

The variable `SAVEPATH` should be the same as the variable `READPATH` in the file backup scripts. Otherwise the database file will not be copied to the backup site.

To allow the script to access the database a file called `.pgpass` needs to be created in the home directory of the user calling the script. This file must contain the following information:

```
localhost:PORT:DATABASE:USERNAME:PASSWORD
```

J.4 Crontab

`Crontab` is the suggested method for automating the invocation of the backup scripts. Adding system-wide tasks to `crontab` is easily achieved using the following command:

```
crontab -e
```

This will open the crontab for the current user, where tasks may then be defined. Running a background task such as a backup should preferably be in the root `crontab` (using `sudo crontab -e`) or in the `crontab` of some other "non-user" user.

Make sure that the scripts added to `crontab` are kept in a secure location, and aren't editable by users without proper access rights. This is necessary to keep malicious users from adding arbitrary code to the `crontab` schedule.

Here is a short example of how to add a scheduled script:

```
0 0 * * * /path/to/script.sh
```

The above line tells `Crontab` to run the script at midnight every day. For more information on `crontab` syntax and behavior, see the `crontab` manual.

[1] That is, if the data is backed up to `EXAMPLE/PATH/genomizer/data`, then the "WORKDIR" should be `EXAMPLE/PATH/genomizer`

J.5 Restoring

Restoring a backup is much like backing up in reverse. Simply invoke `rsync` from the backup site, with the backup directory as source and the genomizer-server's data directory as target.

To restore from an archive, extract the archive contents to a suitable location, and then invoke `rsync` with that directory as source and the genomizer-server's data directory as target.

Restoring the database is achieved by piping the database dump to the live database:

```
psql dbname < FILENAME.sql
```

Equivalently, one could log in to the database and call the dump as a normal `sql` script.

Note that in either case, this operation requires the user to have permission to open and overwrite the database.

K Acceptance Tests

K.1 Introduction

This document contains acceptance tests for the `genomizer` system. It is based on the list of "use cases" found on the `genomizer-documentation` wiki.

The list is as follows[1]:

- Logging In
- Logging Out
- Upload Genome Release
- Delete Genome Release
- Add annotations
- Remove annotations
- Update/Alter Genome Release
- Create Experiment
- Add files to an experiment
- Search experiments
- Search with partial terms
- Processing single files
- Processing multiple files
- Download files
- File conversion: SGR to WIG, WIG to SGR etc

K.2 Tests

This section contains tests for each of the bullet points in the above section.

K.2.1 Logging In

`Logging In` is considered to be "working" if the following tests pass:

Table K.1: `Logging In`: Correct login.

Step	Description
Precondition	There is a running `genomizer` server available..
1	Start a `genomizer` client.
2	Attempt to log in using a valid username and a valid password.
Postcondition	The server and client are both still running. The user is logged in.

Table K.2: `Logging In`: Bad username.

Step	Description
Precondition	There is a running `genomizer` server available..

[1]I've omitted a few duplicate items in this rendition of the list.

Step	Description
1	Start a `genomizer` client.
2	Attempt to log in using an incorrect username.
Postcondition	The server and client are both still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed. The message does *not* communicate why it failed.

Table K.3: `Logging In`: Bad password.

Step	Description
Precondition	There is a running `genomizer` server available..
1	Start a `genomizer` client.
2	Attempt to log in using an incorrect password.
Postcondition	The server and client are both still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed. The message does *not* communicate why it failed.

Table K.4: `Logging In`: No username.

Step	Description
Precondition	There is a running `genomizer` server available..
1	Start a `genomizer` client.
2	Attempt to log in with no username.
Postcondition	The server and client are both still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed. The message does *not* communicate why it failed.

Table K.5: `Logging In`: No password.

Step	Description
Precondition	There is a running `genomizer` server available.
1	Start a `genomizer` client.
2	Attempt to log in with no password.
Postcondition	The server and client are both still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed. The message does *not* communicate why it failed.

Table K.6: `Logging In`: Garbage username.

Step	Description
Precondition	There is a running `genomizer` server available.
1	Start a `genomizer` client.
2	Attempt to log in with garbage[2] input in the username field.

[2]Very long random unicode string for instance.

Step	Description
Postcondition	The server and client are both still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed. The message does *not* communicate why it failed.

Table K.7: `Logging In`: Garbage password.

Step Des	cription
Precondition	There is a running `genomizer` server available.
1	Start a `genomizer` client.
2	Attempt to log in with garbage input in the password field.
Postcondition	The server and client are both still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed. The message does *not* communicate why it failed.

Table K.8: `Logging In`: Bad address.

Step	Description
Precondition	None.
1	Start a `genomizer` client.
2	Attempt to log in on a server that does not exist.
Postcondition	The client is still running. The user is *not* logged in. An error message is displayed, which communicates that the log in failed.

K.2.2 Logging Out

`Logging Out` is considered to be “working” if the following tests pass:

Table K.9: `Logging Out`: Standard behaviour.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server.
1	Attempt to log out.
Postcondition	The server and client are both still running. The user is no longer logged in.

Table K.10: `Logging Out`: Logging in after logging out.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server.
1	Log out.
2	Attempt to log in again with the same credentials.
Postcondition	The server and client are both still running. The user is logged in.

K.2.3 Upload Genome Release

`Upload Genome Release` is considered to be “working” if the following tests pass:

Table K.11: `Upload Genome Release`: Add genome release with 1 file.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to add genome releases. An experiment with processable files exists.
1	Open the administration tab.
2	Open the “genome release” view.
3	Press the “Select files to upload” button.
4	Select a valid file and upload it.
5	Select a valid species and enter a new name for the release.
6	Press “Upload”.
7	Ensure that the file upload completes successfully (or appears to, in the user interface).
Postcondition	The server and client are both still running. The user is still logged in. The new genome release is visible in the list of genome releases, and can be selected in the processing view.

Table K.12: `Upload Genome Release`: Add genome release with multiple files.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to add genome releases. An experiment with processable files exists.
1	Open the administration tab.
2	Open the “genome release” view.
3	Press the “Select files to upload” button.
4	Select several valid files and upload them.
5	Select a valid species and enter a new name for the release.
6	Press “Upload”.
7	Ensure that the file upload completes successfully (or appears to, in the user interface).
Postcondition	The server and client are both still running. The user is still logged in. The new genome release is visible in the list of genome releases, and can be selected in the processing view.

Table K.13: `Upload Genome Release`: Add a genome release with a name already in use.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to add genome releases. A previously added genome release exists on the server.
1	Open the administration tab.

Step	Description
2	Open the “genome release” view.
3	Press the “Select files to upload” button.
4	Select several valid files and upload them.
5	Select a valid species, but enter the name of an already existing genome release for the new one.
6	Press “Upload”.
Postcondition	The server and client are both still running. The user is still logged in. An error message is displayed explaining that a genome release with the chosen name already exists.

Table K.14: `Upload Genome Release`: Add a genome release without a species.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to add genome releases.
1	Open the administration tab.
2	Open the “genome release” view.
3	Press the “Select files to upload” button
4	Select several valid files and upload them.
5	Enter a valid name, but do *not* select a species for the release.
6	Press “Upload”.
Postcondition	The server and client are both still running. The user is still logged in. An error message is displayed explaining that the genome release needs to be associated with a species.

K.2.4 Delete Genome Release

`Delete Genome Release` is considered to be “working” if the following tests pass:

Table K.15: `Delete Release`: Delete an unused genome release.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to delete genome releases. A genome release which is not referenced by any experiment exists on the server.
1	Open the administration tab.
2	Open the “genome release” view.
3	Press the delete button connected to the genome release mentioned above.
4	Confirm the deletion.
Postcondition	The server and client are both still running. The user is still logged in. The chosen genome release is no longer visible in the list, and cannot be selected in the processing view.

Table K.16: `Delete Release`: Delete a genome release which is still in use.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to delete genome releases. A genome release which is referenced by at least one experiment exists on the server.
1	Open the administration tab.
2	Open the “genome release” view.
3	Press the delete button connected to the genome release mentioned above.
4	Confirm the deletion.
Postcondition	The server and client are both still running. The user is still logged in. An error message is displayed explaining that the genome release cannot be deleted.

K.2.5 Add Annotation

`Add Annotation` is considerd to be "working" if the folowing works test pass:

Table K.17: `Add Annotation`: Add forced dropdown annotation.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the "annotation" view.
3	Press "Add/Create new".
4	Fill out the annotation name field.
5	Fill out the Value field.
6	Press the plus sign if you want to add more values and fill out the new field.
7	Mark the forced annotation option.
8	Press create annotation.
Postcondition	The server is still running, the user is still logged in and a new forced Dropdown annotation has been created and is available in the list of annotations.

Table K.18: `Add Annotation`: Add non-forced dropdown annotation.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the "annotation" view.
3	Press "Add/Create new".
4	Fill out the annotation name field.
5	Fill out the Value field.
6	Press the plus sign if you want to add more values and fill out the new field.
7	Press create annotation.
Postcondition	The server is still running, the user is still logged in and a new non-forced Dropdown annotation has been created and is available in the list of annotations.

Table K.19: `Add Annotation`: Add forced freetext annotation.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the "annotation" view.
3	Press "Add/Create new".
4	Change tab to Free Text.
5	Fill out the field annotation namee
6	Mark the forced annotation option
7	Press create annotation

Step	Description
Postcondition	The server is still running, the user is still logged in and a new forced freetext annotation is created and is in the annotation list.

Table K.20: `Add Annotation`: Add non-forced freetext annotation.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the “annotation” view.
3	Press add.
4	Change tab to Free text.
5	Fill out the field annotation name
6	Press create annotation
Postcondition	The server is still running, the user is still logged in and a new forced freetext annotation is created and is available in the list of annotations.

Table K.21: `Add Annotation`: Add dropdown annotation with default values.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the “annotation” view.
3	Press add.
4	Fill out the field annotation name.
5	Press create annotation.
Postcondition	The server is still running, the user is still logged in and a new Dropdown annotation with the values Yes, No and Unknown is created.

Table K.22: `Add Annotation`: Add dropdown annotation with freetext as only value.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the “annotation” view.
3	Press “Add/Create new”.
4	Fill out the field annotation name.
5	Mark freetext in the Value field.
6	Press create annotation
Postcondition	The server is still running, the user is still logged in and a new freetext annotation is created and is available in the list of annotations.

Table K.23: `Add Annotation`: Add dropdown annotation with garbage values.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the “annotation” view.
3	Press “Add/Create new”.
4	Fill out the field annotation name or value with invalid characters[3]
5	Press create annotation
Postcondition	The server is still running, the user is still logged in and a error message should be shown and the annotation should not have been created.

Table K.24: `Add Annotation`: Add freetext annotation with garbage values.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in with an appropriate client and has permission to add annotations.
1	Open the administration tab.
2	Open the “annotation” view.
3	Press “Add/Create new”.
4	Fill out the field annotation name with invalid characters.[4]
5	Press create annotation.
Postcondition	The server is still running, the user is still logged in and a error message should be shown and the annotation should not have been created.

[3]What counts as invalid here? - NG
[4]What counts as invalid here? -NG

K.2.6 Delete annotation

`Delete annotation` is considered to be working if the following tests pass:

Table K.25: `Delete annotation`: Delete unused annotation.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to remove annotations. An unused annotation exists on the system.
1	Open the administration tab.
2	Enter the “annotation” view.
3	Press the edit button associated with the chosen annotation.
4	Press the delete button.
Postcondition	The server is still running and the user is still logged in. The chosen annotation is no longer visible in the list in the annotation view.

Table K.26: `Delete annotation`: Delete an annotation in use.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to remove annotations. An annotation, which is referenced in at least one experiment exists on the system.
1	Open the administration tab.
2	Enter the “annotation” view.
3	Press the edit button associated with the chosen annotation.
4	Press the delete button.
Postcondition	The server is still running and the user is still logged in. An error message is displayed explaining that the annotation cannot be removed while it is still in use.

Table K.27: `Delete annotation`: Delete a “hardcoded” annotation.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on a suitable `genomizer` client connected to the server and has permission to remove annotations.
1	Open the administration tab.
2	Enter the “annotation” view.
3	Press the edit button associated with the one of the “hardcoded” annotations[5].
4	Attempt to press the delete button.
Postcondition	The server is still running and the user is still logged in. One of the following is true: An error message is displayed explaining that the annotation cannot be removed **OR** the button is disabled, preventing its use.

[5]Currently these are: “Species” and “Sex”.

K.2.7 Update Annotation

`Update Annotation` is considered to be working if the following tests pass:

Table K.28: `Update annotation`: Update value in a forced dropdown.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu and is of the type “forced dropdown”.
1	Edit one of the annotation values.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The updated annotation is visible in the list of annotations, and contains the updated value.

Table K.29: `Update annotation`: Update value in a non-forced dropdown.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu and is of the type non-forced dropdown.
1	Edit one of the annotation values.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The updated annotation is visible in the list of annotations, and contains the updated value.

Table K.30: `Update annotation`: Update value in a freetext annotation.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu and is of the type freetext.
1	Try updating the annotation value.
Postcondition	The server and client are both still running, and the user is still logged in. The annotation value should not be able to be updated.

Table K.31: `Update annotation`: Update value to a value which already exists.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, is of the type dropdown, and has at least two annotation values.
1	Edit one of the annotation values to be identical to one of the other annotation values.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation value should not be updated, and the system should tell the user that duplicate annotation values are not allowed.

Table K.32: `Update annotation`: Update value to contain a special character.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, and is of the type dropdown.
1	Edit an annotation value to contain a special character.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation value should not be updated, and the system should tell the user that special characters values are not allowed.

Table K.33: `Update annotation`: Update value to be empty.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, and is of the type dropdown.
1	Edit an annotation value to be empty.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation value should not be updated, and the system should tell the user that empty annotation values are not allowed.

Table K.34: `Update annotation`: Add an annotation value.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, and is of the type dropdown.

Step	Description
1	Add an annotation value.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The updated annotation is visible in the list of annotations, and contains the added value.

Table K.35: `Update annotation`: Add value identical to an already existing value.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, and is of the type dropdown.
1	Add an annotation value that is identical to an already existing annotation value.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation value should not be added, and the system should tell the user that duplicate annotation values are not allowed.

Table K.36: `Update annotation`: Add value "freetext".

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, and is of the type dropdown.
1	Add the string "freetext" as an annotation value.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation value should not be added, and the system should tell the user that their annotation value is invalid.

Table K.37: `Update annotation`: Add empty value.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, and is of the type dropdown.
1	Add an empty string to annotation values.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation value should not be added, and the system should tell the user that empty annotation values are not allowed.

Table K.38: `Update annotation`: Remove value from dropdown.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, is of the type dropdown, and contains at least one annotation value.
1	Remove a value from annotation values.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The updated annotation is visible in the list of annotations, and does not contain the removed value.

Table K.39: `Update annotation`: Remove value currently in use.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu, is of the type dropdown, and contains an annotation value that is currently in use by some experiment.
1	Remove an annotation value that is currently in use.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The remove annotation value should still exist, and the system should tell the user that annotation values which are currently in use cannot be removed.

Table K.40: `Update annotation`: Update name.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu.
1	Edit the annotation name.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The updated annotation is visible in the list of annotations, and contains the updated name.

Table K.41: `Update annotation`: Update name to a name which already exists.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu.
1	Edit the annotation name to a name that an already existing annotation already has.
2	Click the update button to save annotation changes.

Step	Description
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation name should not be updated, and the system should tell the user that duplicate annotation names are not allowed.

Table K.42: `Update annotation`: Update name to be empty.

Step	Description
Precondition	There is a running `genomizer` server available. A user is logged in on an appropriate client and has permission to edit annotations. The annotation to be updated is visible in a relevant menu.
1	Edit the annotation name to be empty.
2	Click the update button to save annotation changes.
Postcondition	The server and client are both still running, and the user is still logged in. The new annotation name should not be updated, and the system should tell the user that empty annotation names are not allowed.

K.2.8 Create experiment

`Create experiment` is considered to be working if the following tests pass:

Table K.43: `Create experiment`: Create experiment without typing an experiment name. pass:

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Click the “Upload” tab.
2	Leaving the “Experiment name” field blank, click “Upload experiment”.
Postcondition	The server is still running, and the user is still logged in and in the upload menu. The system should tell the user an experiment name is needed.

Table K.44: `Create experiment`: Create experiment without filling in forced annotations. pass:

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. A forced annotation exists.
1	Click the “Upload” tab.
2	Fill in a valid experiment name.
3	Leaving a forced annotation blank, click “Upload experiment”.
Postcondition	The server is still running, and the user is still logged in and in the upload menu. The system should tell the user that forced annotations cannot be blank.

Table K.45: `Create experiment`: Create experiment without files. pass:

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Click the “Upload” tab.
2	Fill in a valid experiment name and annotation values.
3	Click “Upload experiment”.
Postcondition	The server is still running, and the user is still logged in. The new experiment should be visible when being searched, and should contain no files.

Table K.46: `Create experiment`: Create experiment with one file. pass:

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Click the “Upload” tab.
2	Fill in a valid experiment name and annotation values.

Step	Description
3	Select one file to be uploaded.
4	Click “Upload experiment”.
Postcondition	The server is still running, and the user is still logged in. The new experiment should be visible when being searched, and should contain one file.

Table K.47: `Create experiment`: Create experiment with multiple files.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Click the “Upload” tab.
2	Fill in a valid experiment name and annotation values.
3	Select multiple files to be uploaded.
4	Click “Upload experiment”.
Postcondition	The server is still running, and the user is still logged in. The new experiment should be visible when being searched, and should contain all selected files.

K.2.9 Add files to an Experiment

`Add files to an Experiment` is considered to be working if the following tests pass:

Table K.48: `Add files to an Experiment`: Add one file to an Experiment.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. A suitable experiment is available in the system.
1	Mark the experiment in the client.
2	Click “Upload to experiment”.
3	Select the file that will be uploaded.
4	Add the necessary attributes and press “Upload experiment”.
Postcondition	The server is still running, the user is still logged in and the uploaded file should be in the experiment view.

Table K.49: `Add files to an Experiment`: Add multiple files to an Experiment.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. A suitable experiment is available in the system.
1	Mark the experiment in the client.
2	Click “Upload to experiment”.
3	Select the files that will be uploaded.
4	Add the necessary annotations and press “Upload experiment”.
Postcondition	The server is still running, the user is still logged in and the uploaded files should be in the experiment view.

Table K.50: `Add files to an Experiment`: Add files without filling in forced annotations.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. A suitable experiment and a forced annotation is available in the system.
1	Mark the experiment in the client.
2	Click “Upload to experiment”.
3	Select the files that will be uploaded.
4	Just press “Upload experiment” without filling in the forced annotation.
Postcondition	The server is still running, the user is still logged in and a warning message should be telling the user that forced annotations must be filled in.

K.2.10 Delete Experiment

Table K.51: `Delete Experiment`: Delete empty experiment.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client and has permission to delete experiments. A "deletable" experiment with no files exists on the system.
1	Open the Search tab.
2	Search for the aforementioned experiment.
3	Mark the experiment in the client.
4	Press remove, and confirm this.
5	Search for the experiment again.
Postcondition	The server is still running, the user is still logged in. The experiment is not visible in the search results.

Table K.52: `Delete Experiment`: Delete non-empty experiment.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client and has permission to delete experiments. A "non-deletable" experiment with at least one file exists on the system.
1	Open the Search tab.
2	Search for the aforementioned experiment.
3	Mark the experiment in the client.
4	Press remove, and confirm this.
Postcondition	The server is still running, the user is still logged in. An error message is displayed explaining why the experiment cannot be deleted.

K.2.11 Searching

`Searching` is considered to be "working" if the following tests pass:

Table K.53: `Search`: Single term.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. At least one experiment with correct annotations and files exists on the system.
1	Open the search tab.
2	Enter a valid[6] search term associated with the aforementioned experiment[7].
3	Press the "search" button or equivalently the `Enter` key.
Postcondition	The server is still running, the user is still logged in. The aforementioned experiment is visible in the search results.

[6]That is, `PubMed` syntax: `[expId]`.

[7]You may want to repeat this test with several different kinds of terms; `expId`, `annotations` ..

Table K.54: `Search`: Multiple terms.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. At least one experiment with correct annotations and files exists on the system.
1	Open the search tab.
2	Enter a valid search string containing multiple terms associated with the aforementioned experiment. Include multiple operators if possible. [8]
3	Press the "search" button or equivalently the `Enter` key.
Postcondition	The server is still running, the user is still logged in. The aforementioned experiment is visible in the search results.

Table K.55: `Search`: Multiple terms using constructor.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client. At least one experiment with correct annotations and files exists on the system.
1	Open the search tab.
2	Open the query constructor.
4	Build a search string containing multiple terms associated with the aforementioned experiment. Include multiple operators if possible.
3	Press the "search" button or equivalently the `Enter` key.
Postcondition	The server is still running, the user is still logged in. No results are displayed. An error message is displayed explaining why the search failed.

Table K.56: `Search`: Incorrect syntax.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Open the search tab.
2	Enter an invalid search term, such as "asdasd".
3	Press the "search" button or equivalently the `Enter` key.
Postcondition	The server is still running, the user is still logged in. An error message is displayed, explaining that the query failed to bad syntax.

Table K.57: `Search`: Empty string.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Open the search tab.
2	"Enter" the empty string as the search term.
3	Press the "search" button or equivalently the `Enter` key.

[8] If necessary, the number of operators/terms can be "artificially" increased by adding a meaningless `OR` or `AND NOT` term.

Step	Description
Postcondition	The server is still running, the user is still logged in. All experiments (visible to the user in question) are visible in the search results.

Table K.58: `Search`: Garbage input.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client.
1	Open the search tab.
2	Enter garbage into the search field.
3	Press the “search” button or equivalently the `Enter` key.
Postcondition	The server is still running, the user is still logged in. No results are displayed. An error message is displayed explaining why the search failed.

Table K.59: `Search`: Search after add.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client and has permission to add experiments.
1	Open the search tab.
2	Search for an `expId` that does not exist.
3	Ensure that there are no search results.
4	Open the upload tab, and add a new experiment with the chosen `expId`.
5	Return to the search tab and search again.
Postcondition	The server is still running, the user is still logged in. The new experiment is visible in the search results.

Table K.60: `Search`: Search after add.

Step	Description
Precondition	There is a `genomizer` server running. A user is logged in on a client and has permission to delete experiments. A “deleteable” experiment exists on the system.
1	Open the search tab.
2	Search for the `expId` of the aforementioned experiment.
3	Ensure that the experiment is visible in the search results.
4	Mark the experiment and press the “Remove” button.
5	Confirm the deletion, and search for the `expId` again.
Postcondition	The server is still running, the user is still logged in. The experiment is no longer visible the search results.

K.3 Not yet specified

The following has not yet been dealt with:

- Processing
- File conversion
- User addition/deletion/updates

L API tester

The *genomizer-server-tester* is a separate program from the rest of the server system that uses parts from the desktop clients code. The purpose of the program is to in sequence execute all possible commands that the server should be able to handle.

The program was given a repository under the main *genomizer* repository. The repository is named *genomizer-server-tester*.

L.1 User guide

The repository contains the a build script that creates a jar of the program with the help of *Ant* as follows:

```
ant jar
```

The program then is ran with the help of one argument which is the address of the server.

```
java -jar genomizer-server-tester.jar <address>
```

The address supplied has to be `https://` followed by a `/api` to redirect the program right.

The program then runs a mulitple of pre-defined tests and prints the result on standard output.

Example 9 ...

```
...
...
--------------- USER ----------------------
TEST: POST ADMIN USER  STATUS: SUCCESS
TEST: DELETE ADMIN USER     STATUS: FAILED
.
.
.
-------------------------------------------
Total tests run: 113
Successfull tests: 109
Failed tests: 4
-------------------------------------------
Failed:
DELETE ADMIN USER
-------------------------------------------
```

L.2 Program structure

The program uses the desktop style for upload, requests and connection so the tests ran will give some feedback to desktop code as well.

The tests is built with the help of two classes `SuperTestClass` and `TestCollection`. `TestCollection` contains a list of `SuperTestClass` and has an execute where it runs tests.

The `SuperTestClass` insures that for each tests constructed the test promises to have at least three things, some have a forth.

1. The test has a identifier string.
2. The test has a expected result.
3. The test has a execute method.
4. Some test has a expected result string to match the result received from the server.

Example

First each test has to have a collection.

```
public class CollectionExample extends TestCollection {
private String testName;

public CollectionExample () {
super()
this.testName = "test";

super.commandList.add(new exampleTest("IDENT", true));
}

@Override
public boolean execute() {
System.out.println("------------ TEST COLLECTION -----------");
boolean bigResult = true;

for(SuperTestCommand stc: super.commandList) {
stc.execute();

runTests++;

boolean succeeded = stc.finalResult == stc.expectedResult;

            if (succeeded){
                succeededTests++;
            } else {
                failedTests++;
                nameOfFailedTests.add(stc.ident);
                bigResult = false;
            }

            Debug.log(stc.toString(), succeeded);
            try {
                Thread.sleep(delay);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

}
return bigResult;
}
}
```

Then construct a test.

```
public class ExampleTest extends SuperTestCommand {

public ExampleTest(String ident, boolean expectedResult) {
super(ident, expectedResult);
}

@Override
public void execute() {
try {
CommandTester.conn.sendRequest(new SearchRequest(""),
                    CommandTester.token, Constants.JSON);

            if (CommandTester.conn.getResponseCode() == 200){
                super.finalResult = true;
            }
        } catch (RequestException e) {
            if (super.expectedResult) ErrorLogger.log(e);
        }
   }

}
```

And finally add the test to the `CommandTester.java`.

```
CollectionExample ex = new CollectionExample();
ex.execute();
```

M Known problems

M.1 Web application

M.1.1 Error handling when uploading experiments

If there is an error when adding files to an experiment the experiment collapses so the user won't get a chance to correct their mistake

Start uploads of files when all files have been added without errors.

M.1.2 Old authorization token causes page redirect

If the authorization token expires the user will be sent to the login screen and any input entered will disappear.

Show login modal without redirecting to root url and save erroneous ajax request and resend it when the login has been completed.

M.1.3 Code duplication in SearchResults and Experiments

The collections SearchResults and Experiments represents the same models. But are different collections as they have different URL. It might be better to have the both use the same collection.

Merge SearchResults and Experiments one collection.

M.1.4 No warning when closing tab during upload.

If a upload is in progress, there is no warning when closing the tab and the upload is canceled directly.

There are some code for this in view/Upload.js, but it's currently broken.

M.1.5 Uploading genome release - does not update list automatically

After adding a genome release the list does not update automatically.

Build functionality to render when upload is done.

M.1.6 The annotation list can't be sorted

The annotation list should be re-ordered by clicking on table headers.

Rebuild list to match design of GenomeReleaseView. We had some problems as we are using a separate list for the search-bar.

M.1.7 No warning when closing tab during uploading genome releases

If an upload is in progress, there is no warning when closing the tab and the upload is cancelled directly.

M.1.8 The page have to be refreshed after adding a new annotation

If the page is not refreshed, the annotation will not be visible in any view, which is a problem especially when the annotation is forced.

M.1.9 Closing raw processing status window does not stop update JSON messages to be sent

Having the processing status window open is correctly causing JSON objects reporting the processing status to be sent continuously, but closing the window does not stop the objects from being sent.

M.1.10 Pressing ”select all” in the convert view allows reconversion

Files are not supposed to be converted several times, but when ”select all” is pressed, the GUI allows for files that has already been converted to be marked again.

Change the behaviour of the ”select all” button.

M.1.11 Pressing ”upload experiment” when editing an experiment causes the ”update annotations” button to stop working

Should be clickable at all times during editing of an already existing experiment, perhaps the ”upload experiment” button should only be clickable when new files are added.

M.1.12 File selection in process view should be improved

A file that results from one processing step should turn up in the selection menu of the subsequent processing steps. Now, mainly simple text fields are used.

M.2 *Android* application

M.2.1 Processing

The Android in it’s current state is limited to just convert/process from raw to profile. The layout for the other processes is the same so implementation of step,ratio and smoothing is not as big of a task. More list has to be created and matching JSON package has to be made.

M.2.2 Convert between file formats

This function is not yet implemented.

M.2.3 Security

SSL is not safe in it's current state, right now the application trusts all host. That should be change to trust just the hosts that the application needs.

M.3 *iOS* application

M.3.1 Processing

The iOS application is in it current state limited to only convert from raw to profile and to make step size processing, ratio processing and smoothing.

M.3.2 Advanced Search

The application doesn't save the advanced search query if edited which can cause confusion for the end-user. It doesn't either update the search constructor if a advanced search query is added.

M.4 Desktop

M.4.1 Desktop

Some of these following issues are better documented via *GIT* issues.

M.4.1.1 Cosmetic Issues

Some cosmetic and usability issues of the GUI is described below.

M.4.1.1.1 Component size and resize

The components making up the GUI will in several places not have well adapted sizes, making them more difficult to use than is neccessary. Several will also not resize correctly together with the main window, due to bad layout managers used.

M.4.1.1.2 Repainting status and selectors not direct

When for example hovering over selector arrows, or doing something that updates the status panel, the GUI element in question might not automatically update its visible part, due to not calling repaint at the correct time.

M.4.1.1.3 Flickering tab panes

Some GUI elements seem to flicker on mouseover, or update at incorrect times. This might lead to minor inconvenience or irritation for the user.

M.4.1.1.4 Feedback limited

The feedback on some operations is rather limited, and the only way to make sure whether some things worked is to reload the data from the search panel and check.

M.4.1.2 Event Dispatch and Threads

Some more serious threading issues are described in the following sections.

M.4.1.2.1 Threading solutions

The threading solutions are at several places very messy and not very well understood. It might not be thread-safe, and at some, rare, times give the user issues. A complete overhaul and restructuring might be neccessary.

M.4.1.2.2 The event dispatch

The *swing* event dispatch thread should be used for all GUI related operations. It is not correctly used so at all places. This leads to rare issues with certain panels not loading as expected. Until the threading is restructured, the workaround is to simply reload the application.

M.4.1.3 Transfer Handling

The communication and transfer of data to the server has some known issues, described below.

M.4.1.3.1 Empty files

Empty files might not be up/down-loaded correctly, or these processes not started at all.

M.4.1.3.2 Feedback

The feedback for when uploading a large file might not be appropriate, or come at the expected time. The same might hold for the download. Generally, the feedback for file transfers could be improved. The system status could also be updated better.

M.4.1.3.3 Limitations and restrictions

Some restrictions of what names and values are allowed exist, and controlled by the server. These include dissallowing certain file-names, and extensions.

M.5 Server

M.5.1 Business logic

M.5.2 HTTP Headers

There is an issue that causes the response from the server to have multiple HTTP headers. This only happens when upload/download requests are sent.

M.5.3 Processing

A PutProcessingCommands can hold a list of processes to run. It is currently impossible to cancel a single process from this list. The entire list must instead be cancelled.

M.5.4 Apiary

The apiary site does not match the API to 100%. Some items need further explanation and some inconsistensies exist.

M.5.5 Upload and download

The two scripts used for file transferring in the Genomizer system have some limitations. These will be presented below. Please note that both the scripts reads the settings.cfg file to get information to be able to access the database. Make sure to put a copy of the settings.cfg file into the */var/www/*.

M.5.5.1 Upload script

When a user tries to upload a file and the upload is interrupted the file entry will remain in the database but the file will not exist in the file system. The file will have the status 'In Progress' but will never be uploaded if the user do not try to upload the file again. Furthermore the script will not return good error messages to the user if a file transfer is interrupted.

M.5.5.2 Download script

If a file download is interrupted the user will not receive a error message from the script containing and explaining the reason for the interrupt.

M.5.6 Process limitations

- Ratio calculation has a limitation that it requires processing to be run on two files and that one of these files needs to be explicitly named in the input.
- One known problem with the smoothing subprogram, is that if a chromosome is smaller than the windowsize, the program will then smooth that chromosome together with the following chromosome. In practice this problem should never occur on a regular file when doing smoothing once.

  However, if stepping is done on a file with a step size of, for example 10 000, and we want to smooth the new file again with a window size of 100, then the shortest chromosome in the original file must be atleast 1 000 000 rows. From what we have seen of the melanogaster data the shortest chromosome there is roughly 200 000 rows.

Therefore a user should be cautious when smoothing the second time on file that has been stepped with a large step size.