Sample rst2pdf doc

version 0.1.2

Your Name

June 23, 2023

Contents

IGEM (Integrative Genome-Exposome Method)

Version: 0.1.2

The IGEM (Integrative Genome-Exposome Method) system is a powerful platform designed to host various applications (APPs) that share common resources and interact with each other through a single database. The primary goal of IGEM is to provide a flexible and dynamic framework for genomic and exposomic research, enabling the integration of diverse data sources and facilitating complex analyses.

The GE (Gene x Exposome) application developed within the IGEM system is focuses on collecting external data sets, identifying key genetic and exposomic information, and building a comprehensive knowledge base. This knowledge base is readily available for dynamic and exploratory queries, empowering researchers to uncover valuable insights and generate novel hypotheses.

This User Guide aims to provide comprehensive documentation for utilizing the IGEM platform and its various applications. It covers installation instructions, detailed usage examples, and explanations of key functionalities. Whether you are a researcher, data scientist, or domain expert, this guide will help you leverage the IGEM system effectively to drive your genomic and exposomic analyses.

Introduction

The Integrative Genome-Exposome Method (IGEM) is a novel software to study exposure-exposure (ExE) and gene-environment (GxE) interactions in high-dimensional big data sets by integrating an automated knowledge-based user-friendly, open-source, and open-access software.

IGEM is a software to perform high-throughput quality control (QC), knowledge-driven ExE and GxE filtering, machine learning (ML) and regression-based interaction analysis, and big data visualization.

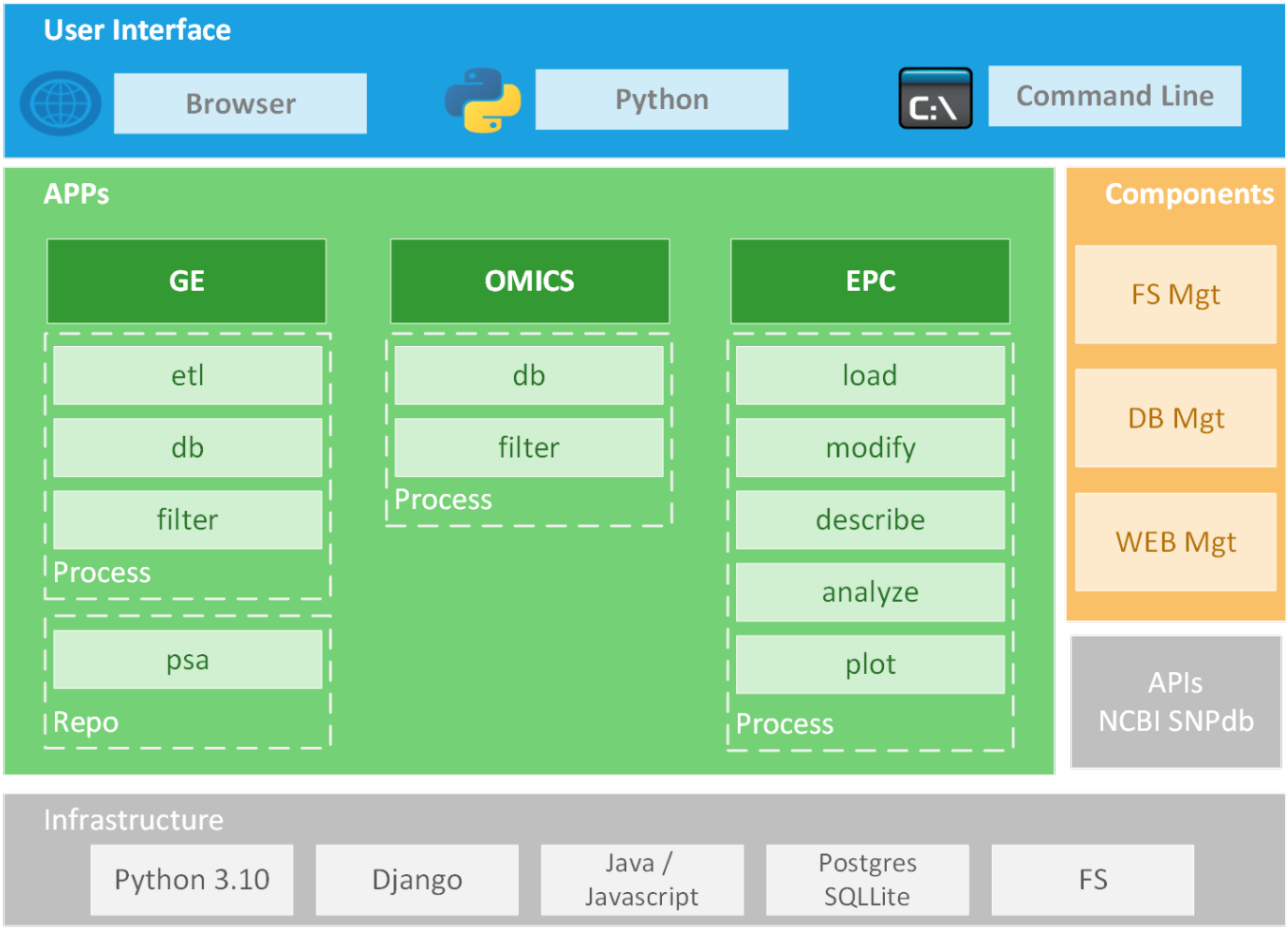
The IGEM system has a modular architecture, initially designed with three applications: GE, OMICS, and EPC, detailed throughout this document.

All applications interact with each other. For example, we can query a GxE relation from GE.db, integrate with other external data, perform regressions and analyze without additional software.

To support the applications, components were implemented that work transparently for the user, and we performed database interface operations, with the file system, among others.

The IGEM can be accessed through a WEB query interface, Python strings, or the command line.

Below is a consolidated view of the IGEM components.

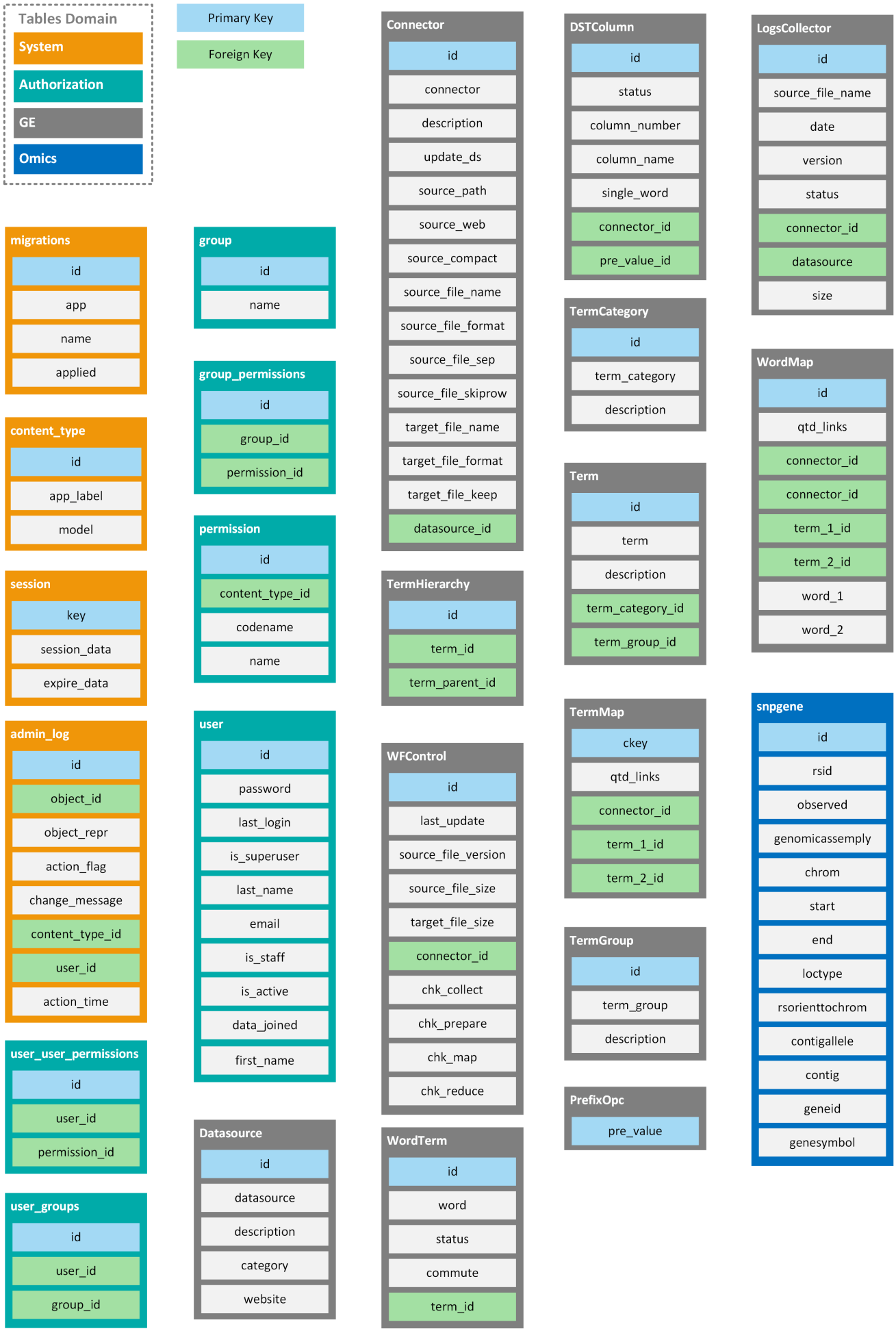


Every IGEM application has processes; we can access them through the available functions and their respective arguments.

Below is an overview of available functions.

IGEM has a database with adapters for SQLite and Postgres and the flexibility of implementation in any database with support for a connector in Python.

The IGEM database has four groups of domains, two for internal functions and the other two for hosting GE and OMICS application data. The EPC application only works with runtime data.


Tables Domain
System
Authorization
GE
Omics
Primary Key
Foreign Key
migrations
id
app
name
applied
content_type
id
app_label
model
session
key
session_data
expire_data
admin_log
id
object_id
object_repr
action_flag
change_message
content_type_id
user_id
action_time
user_user_permissions
id
user_id
permission_id
user_groups
id
user_id
group_id
group
id
name
group_permissions
id
group_id
permission_id
permission
id
content_type_id
codename
name
user
id
password
last_login
is_superuser
last_name
email
is_staff
is_active
data_joined
first_name
Datasource
id
datasource
description
category
website
Connector
id
connector
description
update_ds
source_path
source_web
source_compact
source_file_name
source_file_format
source_file_sep
source_file_skiprow
target_file_name
target_file_format
target_file_keep
datasource_id
TermHierarchy
id
term_id
term_parent_id
WFControl
id
last_update
source_file_version
source_file_size
target_file_size
connector_id
chk_collect
chk_prepare
chk_map
chk_reduce
WordTerm
id
word
status
commute
term_id
DSTColumn
id
status
column_number
column_name
single_word
connector_id
pre_value_id
TermCategory
id
term_category
description
Term
id
term
description
term_category_id
term_group_id
TermMap
ckey
qtd_links
connector_id
term_1_id
term_2_id
TermGroup
id
term_group
description
PrefixOpc
pre_value
LogsCollector
id
source_file_name
date
version
status
connector_id
datasource
size
WordMap
id
qtd_links
connector_id
connector_id
term_1_id
term_2_id
word_1
word_2
snpgene
id
rsid
observed
genomicassemply
chrom
start
end
loctype
rsorienttochrom
contigallele
contig
geneid
genesymbol

The IGEM can be used in a Client-Server scheme, with the Server being responsible for the maintenance of the Database and the clients with an instance of the IGEM pointed to the Server's Database.

The knowledge base is customized to meet different needs. It can be used only for extracting, transforming, and reading in the Database or keeping the original data in a Data Lake format for further queries.

Throughout the manual, we will detail all the functionalities of IGEM and applications.

Install

IGEM is available on PyPI or through GitHub. It can be installed in a virtual environment with Python >= 3.9. Run via the command line:

```
$ pip install igem
```

Database Customization

IGEM accepts several types of software to manage the database, including MS SQL, MySQL, Postgres, and others. By default, the system is already configured with SQLite.

To change the database manager, open the {package_path}/igem/src/settings.py file and change the DATABASES parameters. The example below demonstrates a configuration using a Postgres database:

```
DATABASES = {
  "default": {
      "ENGINE": "django.db.backends.postgresql_psycopg2",
      "NAME": "IGEM",
      "USER": "postgres",
      "PASSWORD": "your_password",
      "HOST": "127.0.0.1",
      "PORT": "5432",
  }
}
```

IMPORTANT: Changing the database is optional as the system is configured by default to create a local SQLite database.

If you want to use a database created on another computer/server, edit the base path, for example, an SQLLITE base:

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": {path} / "db.sqlite3",
    }
}
```

We created a python script to create the database, make the first admin user, and load the initial master data. If you want to start the database using this script, download the file:

_file path

Unzip and run the deploy_db.py script in the environment with IGEM.

The other way to create the database, access the IGEM folder and run the following:

```
$ python manage.py makemigrations
```

IGEM will copy all the tables and other metadata in the configured Database format.

```
Migrations for 'ge':
  ge/migrations/0001_initial.py
    - Create model Category
    - Create model Database
    - Create model Dataset
    - Create model Group
    - Create model Keyge
    - Create model LogsCollector
    - Create model PrefixOpc
    - Create model WordMap
    - Create model WFControl
    - Create model KeyWord
    - Create model KeyLink
    - Create model KeyHierarchy
    - Create model DSTColumn
```

The next command to create the database with all the IGEM metadata:

```
$ python manage.py migrate
```

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, ge, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying ge.0001_initial... OK
  Applying sessions.0001_initial... OK
```

At this point, we already have IGEM installed and the database created with the IGEM structure. To check if the system is working correctly, type:

```
$ Python manage.py check
```

```
System check identified no issues (0 silenced).
```

The IGEM system has a layer of security per user and functions. To create the first user, run:

```
$ python manage.py createsuperuser
```

Enter your username, email, and security password.

```
Username: user_name
Email address: user_name@domain.com
Password:
Password (again):
Superuser created successfully.
```

The system will be ready to parameterize the master data, perform external data load and generate reports.

Web Interface

The IGEM system has a web interface for performing activities such as master data registration and simple queries in the database.

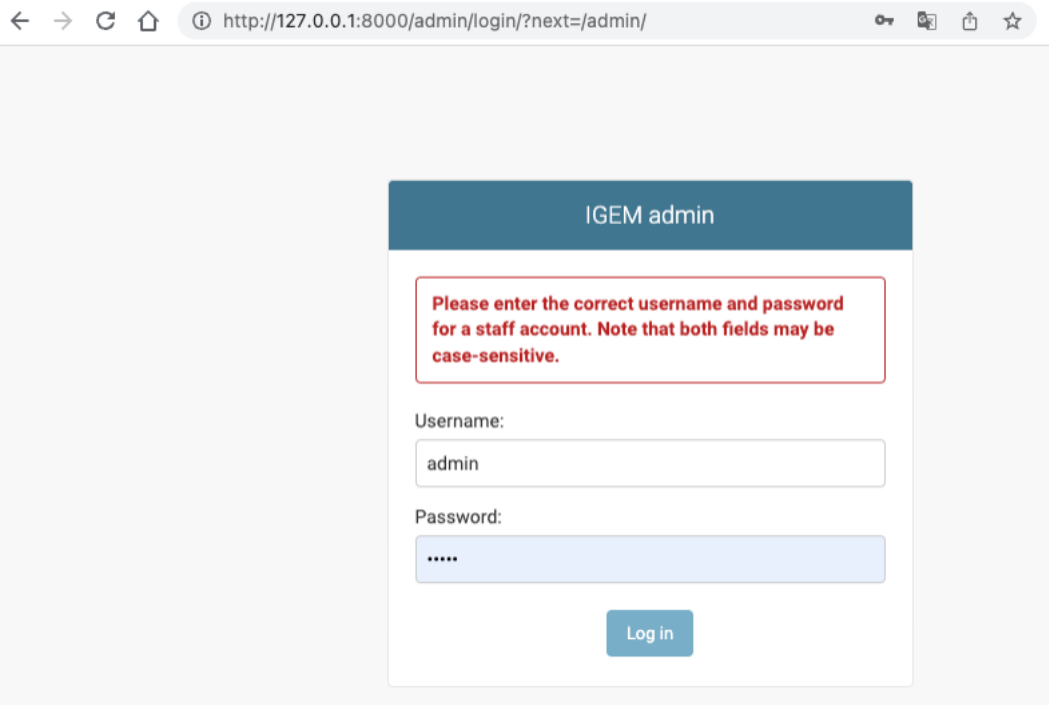
To start the WEB service, type:

```
$ python manage.py runserver
```

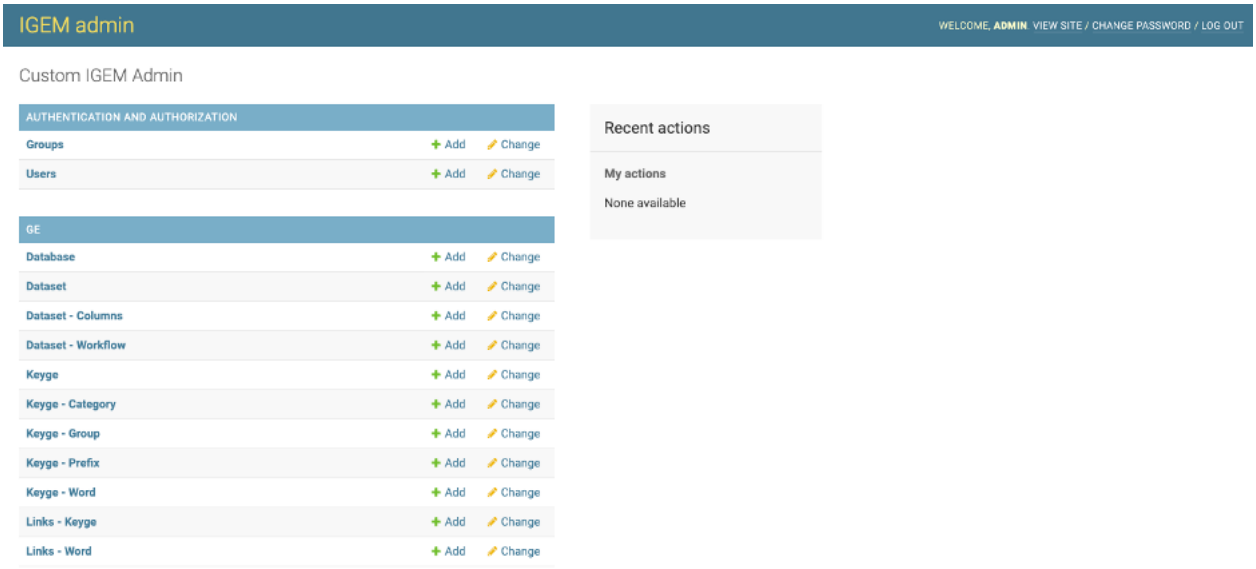
```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 07, 2022 - 15:59:18
Django version 4.0.5, using settings 'src.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

In a browser, go to http://127.0.0.1:8000/admin.

The IGEM system will show the authentication page.



Enter the username and password created in the previous steps. The administration page will be loaded after authentication.



The features of this interface will be explored in detail in Master Data and Access and Permissions.

File structure:

Inside the src directory, we will have:

- */ge/*: all source codes and interfaces for the functioning of APP GE.
- */loader/*: all input files for loading master data and output directory of the FILTER process.
- */psa/*: Persist Store Area to store the database files downloaded and processed by the ETL process. Each DATASET will have its subfolder within the PSA.
- */src/*: hosts the source code of IGEM components, configurations and parameterizations.
- */templates/*: hosts the standard web interfaces in IGEM.

PSA - Store Area Persists

The PSA is a folder that stores the Dataset files loaded in their original format and transformed during the ETL process. Each Database will be a subfolder, and each Dataset a subfolder concerning the Database. These

structures will be created automatically, and if deleted, they will be created again on the following workflow run for the corresponding dataset. Each external Dataset source will be a design solution for the original uploaded file. If you want to keep this file for queries and analyses, configure it in the Dataset register to keep the original file. Important that this file will be kept unzipped. To reduce the amount of system space, it is not recommended to keep these files. The subfolder will also have a transformed version normalized by the rules of the applied dataset.

Users and Role

The IGEM system was developed to be flexible, and it is necessary to evaluate the best configuration for the scenario and objectives of each installation. A suggestion would be a model of three functions, being:

- *Administrator*: responsible for installing and updating the environment to receive the IGEM, Customizing the IGEM, configuring the database and monitoring the performance and creating and maintaining users.
- *Super User*: responsible for registering master data such as Database, Dataset, Keyge, among others. He will also be responsible for creating the ETL JOBs and monitoring them via workflow. For this group, we will have access to the WEB interface for parameterization of the registration, the necessary tables, access to processes such as Collect, Prepare, and DB
- *Users*: they will be the clients of the system, performing queries and analysis of the IGEM data. For this group, we will have processes such as GE.filter

Users

New users can be created via command line:

```
$ python manage.py createsuperuser
```

Through IGEM's friendly web interface, it will be possible to carry out Users management activities.

Activate the IGEM web service if you have not already done so. Go to the /src/ folder and type the command line:

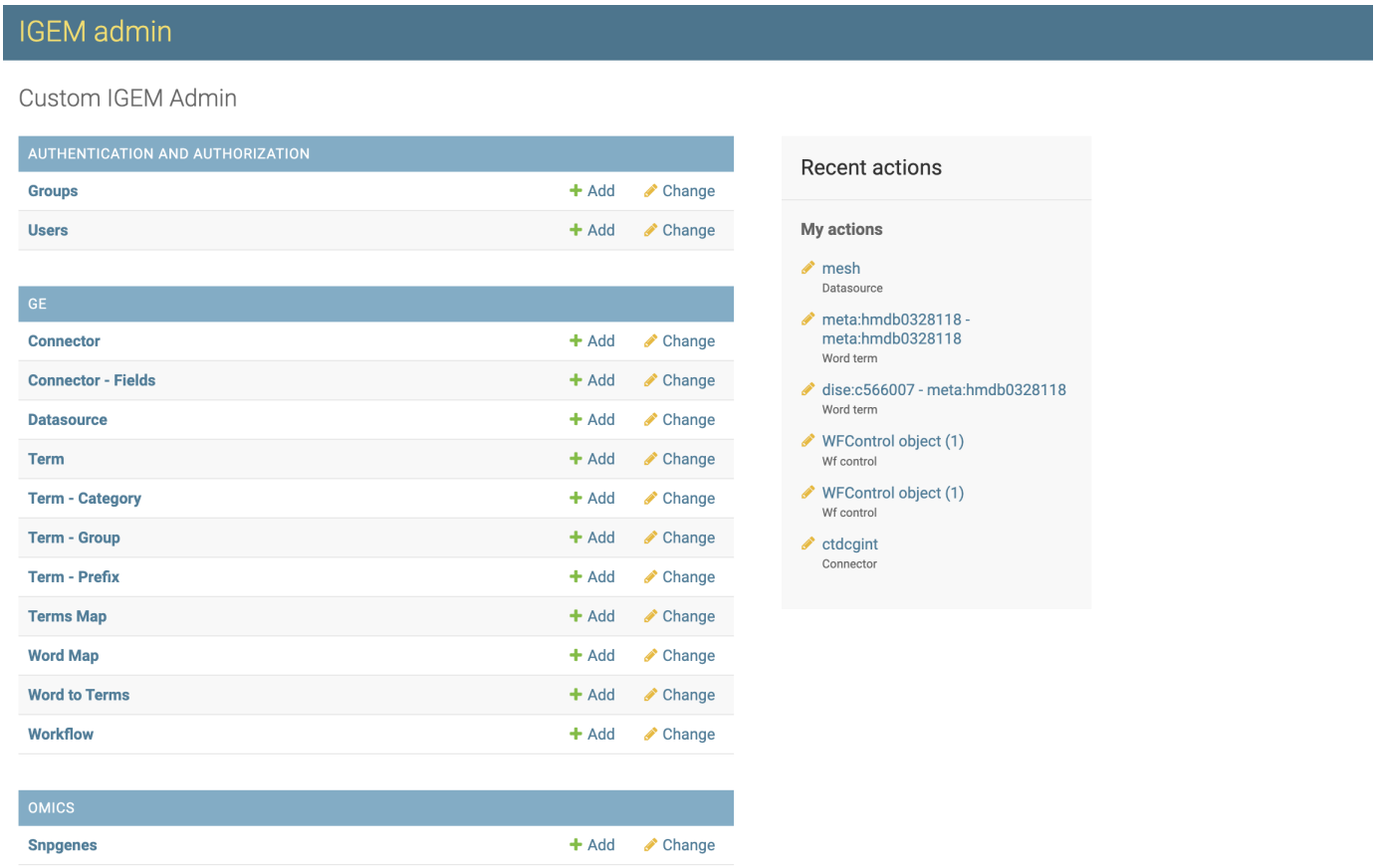
```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

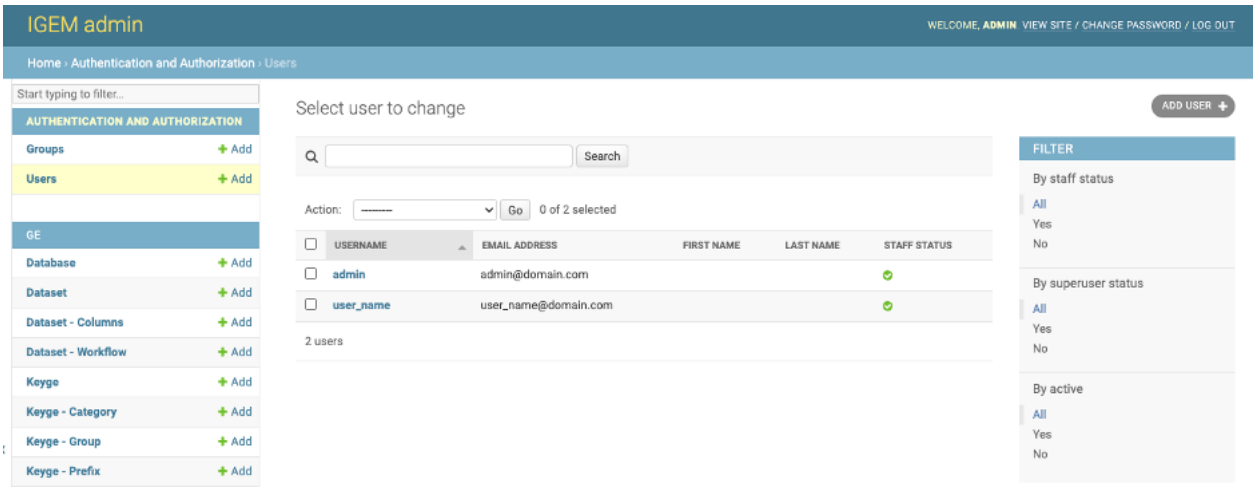
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

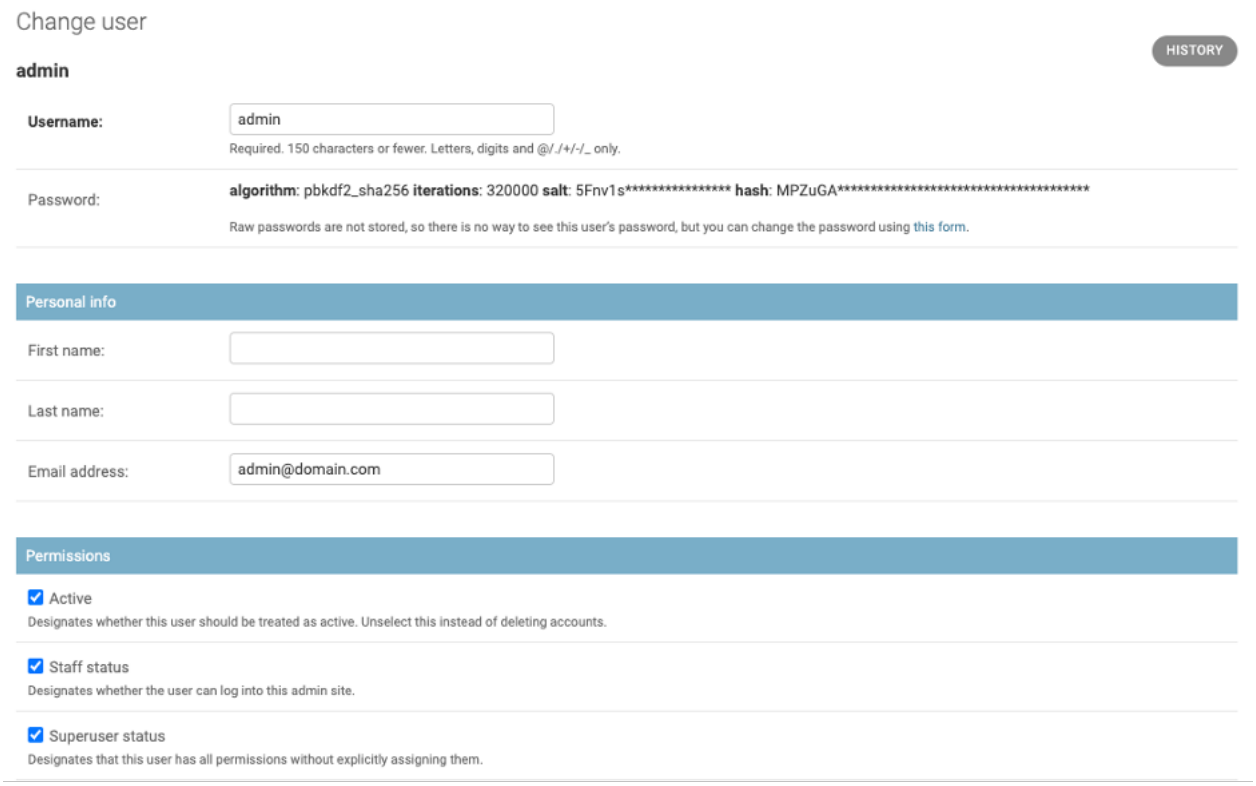
After user authentication and on the initial administration screen, select an option Users.



On the User screen, we will have options to consult, modify, add and eliminate Users.



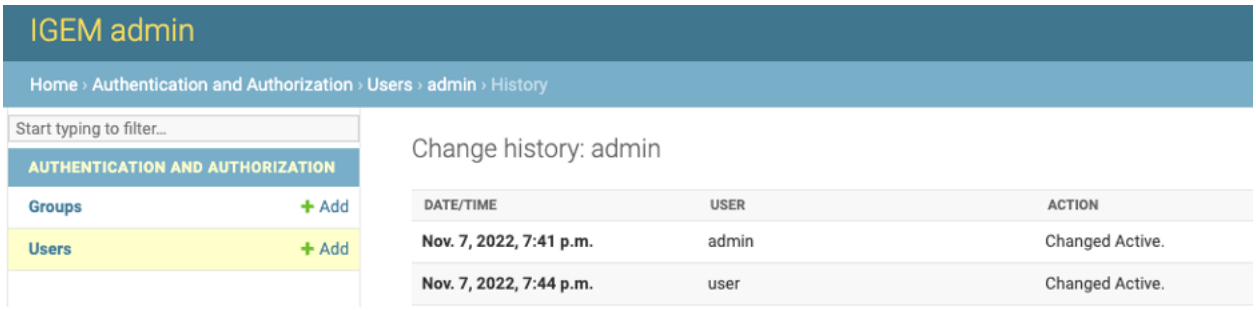
On the first screen, we have a view of all available Users. To consult, click a desired User.



On the next screen, we have all the Users fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank User screen to add a new User record.
- `Save and Continue editing`: Will save the changes and continue on the User screen.
- `Save`: Will save the changes and return to the screen with the list of User.

In the History button, we can consult all the modifications carried out in the User, this function will be important to track modifications and audit the process.



The `DELETE` button will permanently delete the User record.

Caution: when deleting a User, the system will also delete all records dependent on that User.

Deletion can also be performed en bloc. On the Users List screen, select all the User you want to delete, choose the Delete Selected User action and click on the `GO` button.

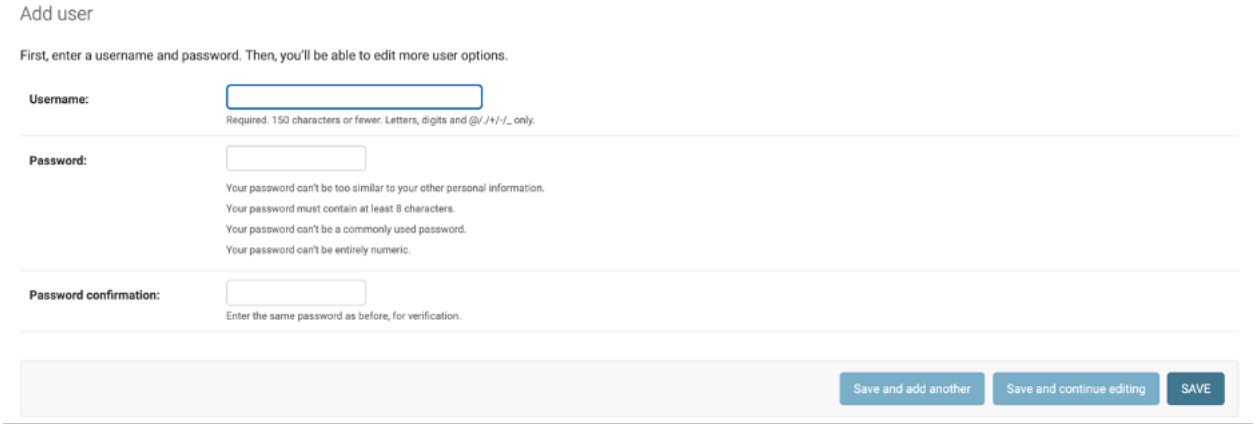
Be careful, this elimination operation will be definitive for the User and for all other records dependent on it, as already explained.

For the User, we will have two filter locations:

- First located at the top of the User List screen where we can search broadly.
- Second on the right sidebar, being able to select by status and actives.

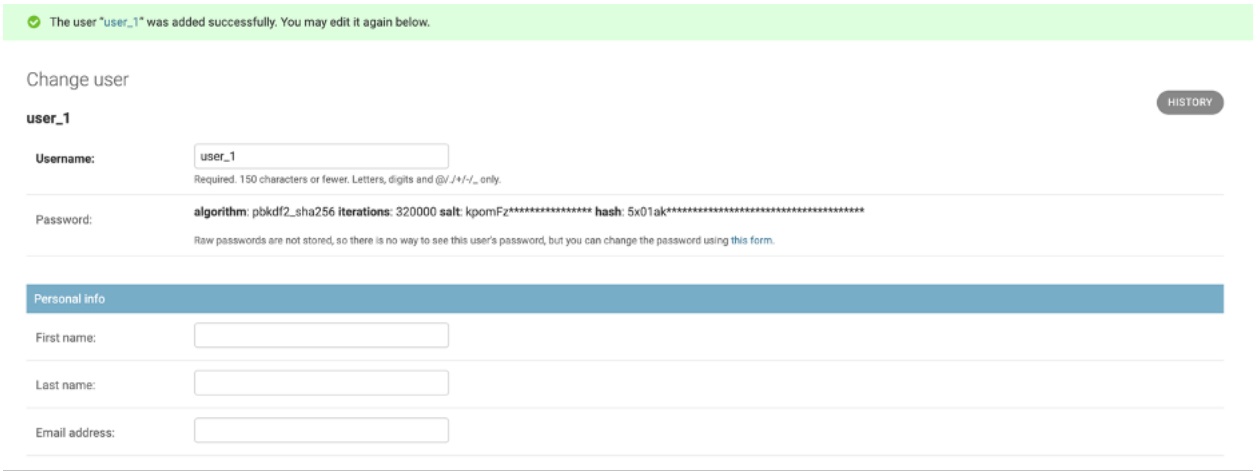
To add new User, we will have three different ways:

- by the + Add button on the left sidebar.
- Through the ADD USER + button in the right field of the Users list.
- Via the Save and add another button located within a User record.



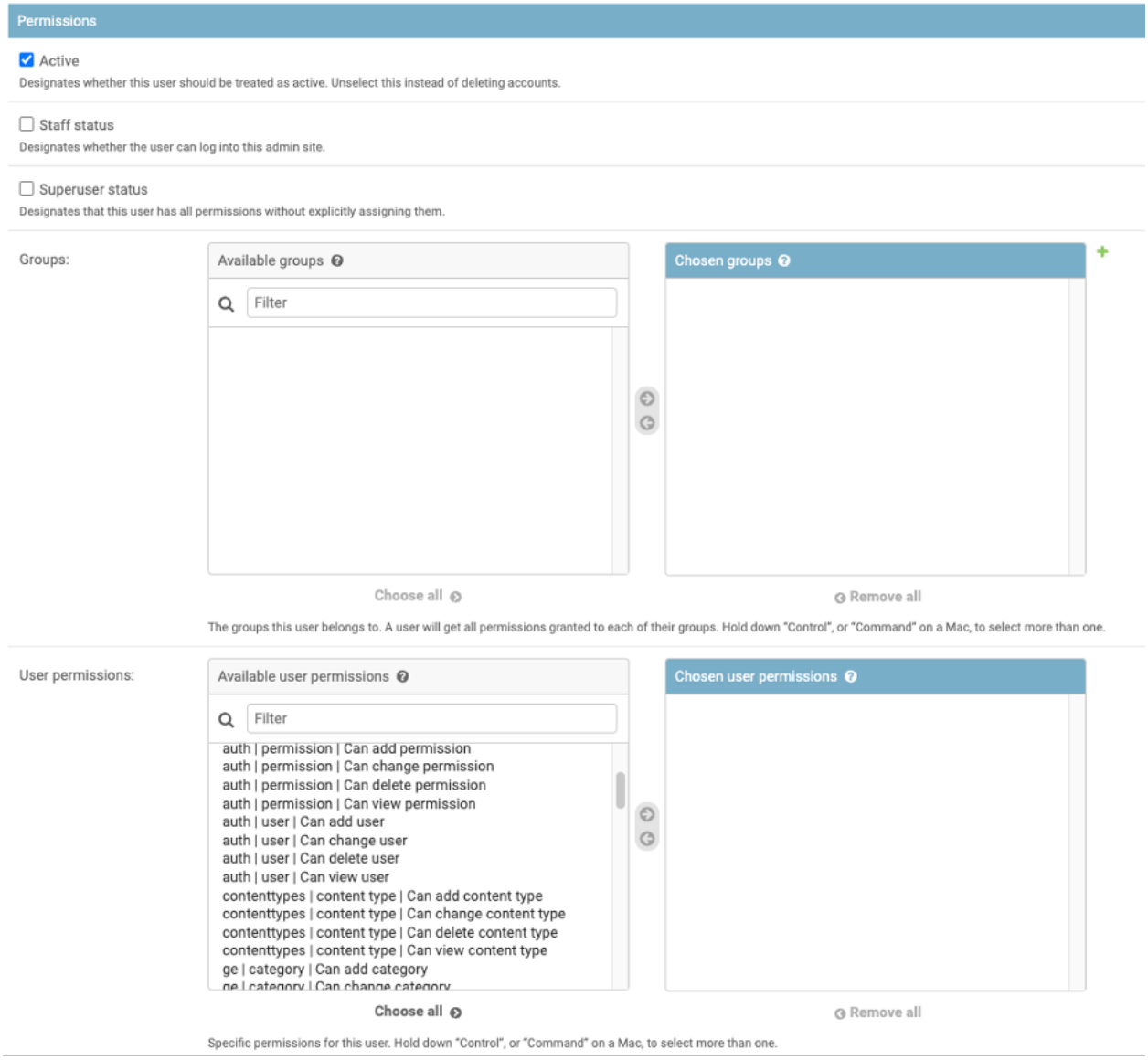
After entering the username, and password and saving, the system will be directed to the user details page.

Inform the personal data of the first name, last name, and email address.



Under permissions, check:

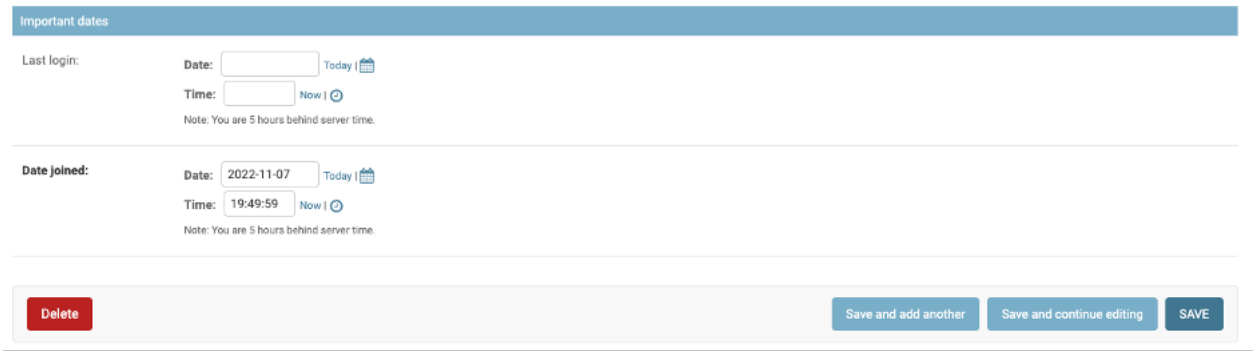
- Active Box to allow user activities.
- Staff to allow the user to access the administration page
- Superuser to give access to all data and system registration. If this option is not checked, it will be necessary to manually add which records and functions the user will have access to or add a group so that the user can access the system.



In groups, inform which groups the user will inherit the accesses to. For Super User, it will not be necessary to advertise any groups as they are given full access.

If you want to customize the user or add more system functionality and access options, access the user's permissions type.

In user date, we will have how much was the last access and the date when the user was created.

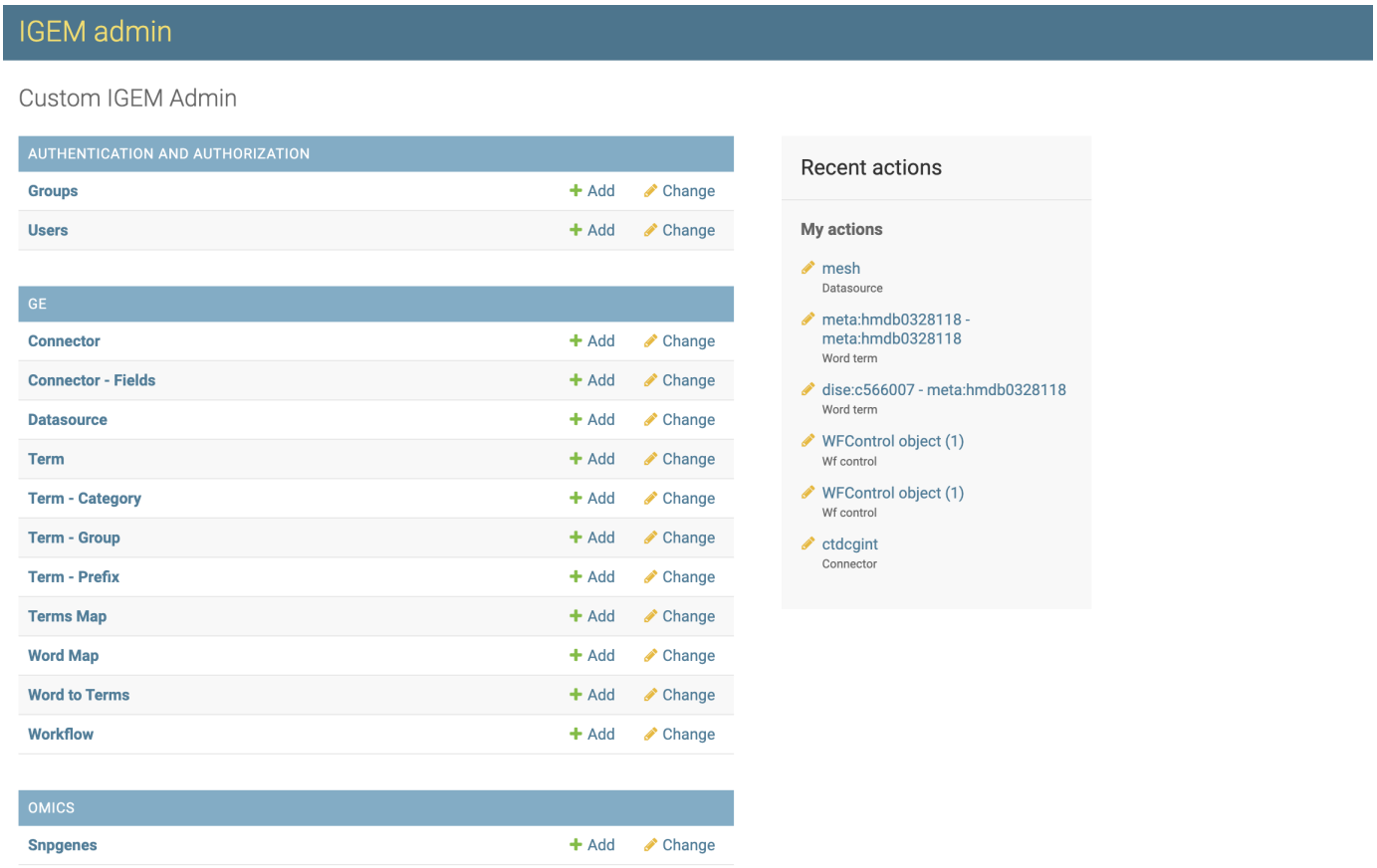


After performing the new parameterizations, save the new user

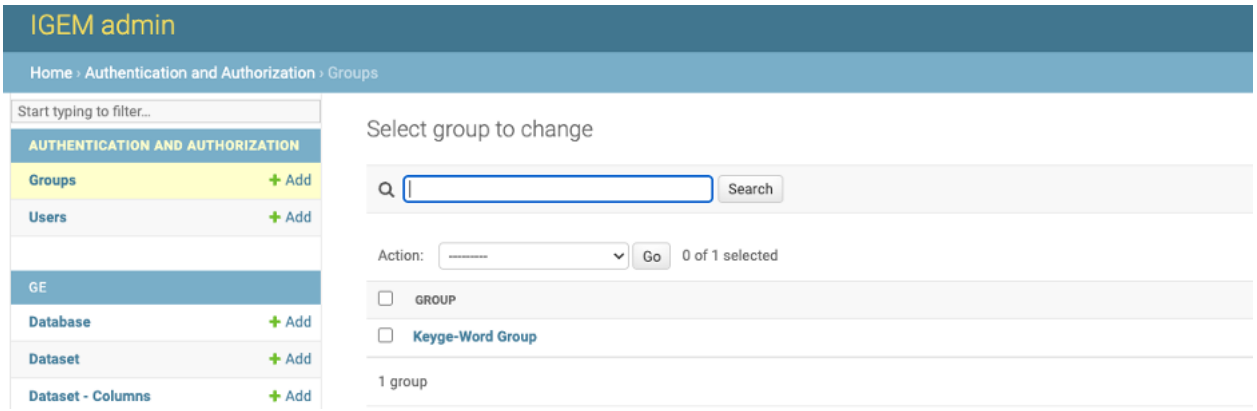
Group

The groups help maintain access; we can create groups for different functions and assign them to the users who perform them, thus avoiding giving users undue access.

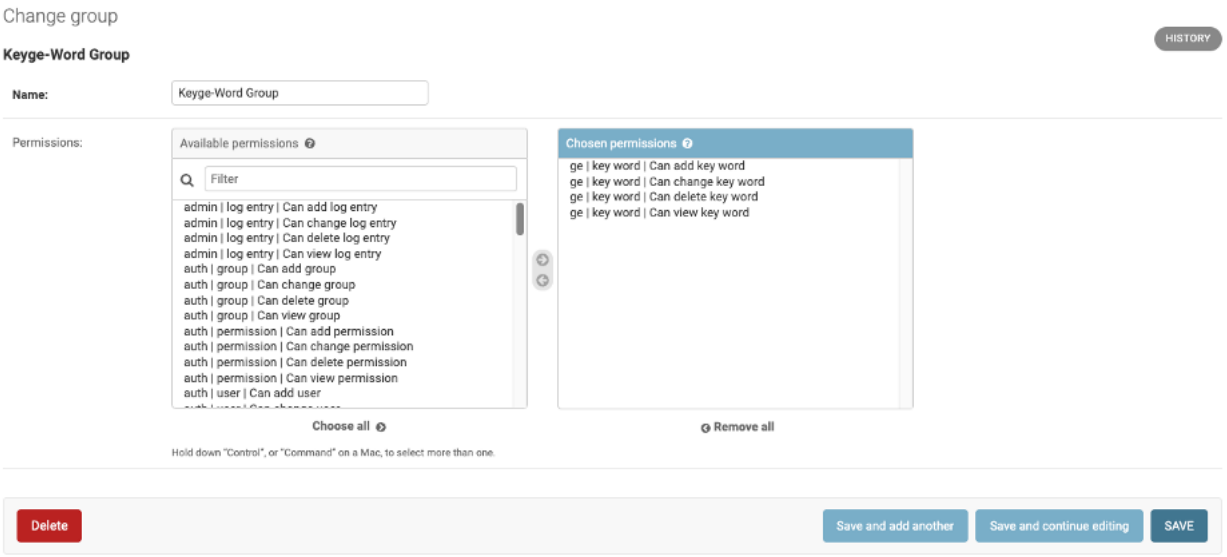
Celect an option Groups.



On the Groups screen, we will have options to consult, modify, add and eliminate Groups.



On the first screen, we have a view of all available Groups. To consult, click a desired Group.



On the next screen, we have all the Group fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank User screen to add a new Group record.
- `Save and Continue editing`: Will save the changes and continue on the Group screen.
- `Save`: Will save the changes and return to the screen with the list of Groups.

In the History button, we can consult all the modifications carried out in the Group, this function will be important to track modifications and audit the process.

The `DELETE` button will permanently delete the Group record.

Caution: when deleting a Group, the system will also delete all records dependent on that Group.

Deletion can also be performed en bloc. On the Users Group screen, select all the Group you want to delete, choose the Delete Selected Group action and click on the `GO` button.

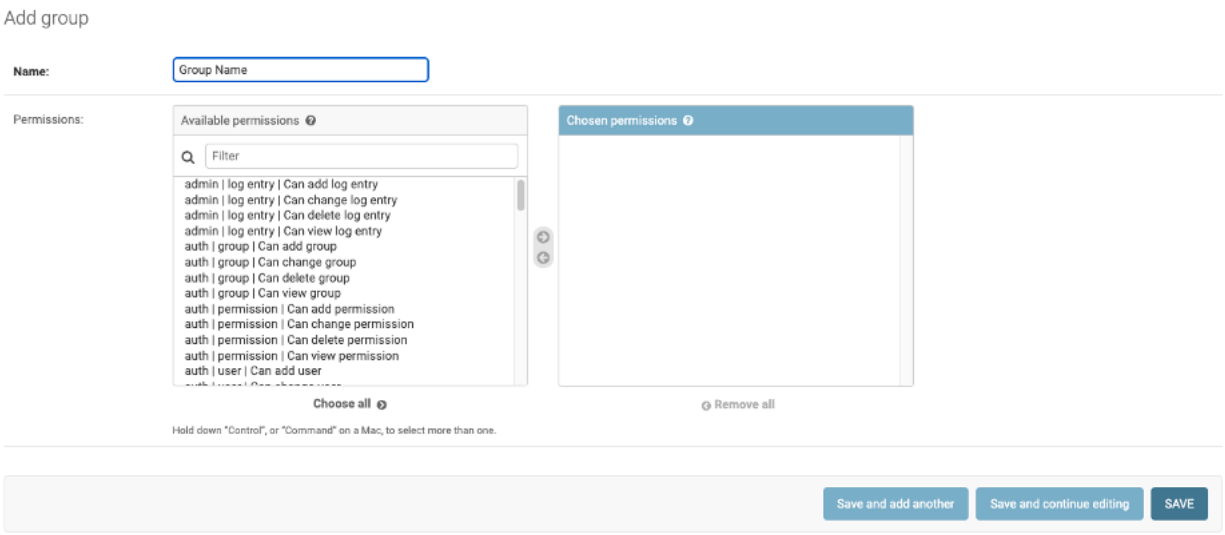
Be careful, this elimination operation will be definitive for the Group and for all other records dependent on it, as already explained.

For the Group, we will have on filter locations:

- Located at the top of the Group List screen where we can search broadly.

To add new Group, we will have three different ways:

- by the + Add button on the left sidebar.
- Through the `ADD GROUP` + button in the right field of the Group list.
- Via the `Save and add another` button located within a Group record.



After entering the username, and password and saving, the system will be directed to the user details page.

A group combines different table accesses and access types. select the tables, type by the functional relationship on the left, and click the arrow to take to the box on the right. All combinations added in the correct box will be assigned to users who inherit this access group.

After performing the new parameterizations, save the new Group.

GE Application

The GE module is a powerful component of the system that encompasses various functionalities related to data processing and analysis in the context of genomics and exposomes. It consists of two important components: GE.db and GE.Filter.

1. The GE.db provides direct access to the underlying database tables, allowing users to retrieve information directly from the IGEM Client DB. It offers the capability to query and analyze data stored in the database tables, empowering users to efficiently extract specific information for their research purposes. Additionally, the GE.db facilitates synchronization between the IGEM Client DB and the Hall Lab DB Server, ensuring the availability of up-to-date data. Users can choose between offline and online synchronization options based on their requirements.
2. The GE.Filter offers a range of functions to filter and retrieve information from the IGEM Client DB, specifically focusing on the relationships and reports related to genomics (G), exposomes (E), and their interactions (GxE and ExE).

By leveraging the functionalities of the GE.db and GE.Filter, researchers can efficiently access and analyze data, extract relevant information, and explore the relationships between various elements in the genomics and exposomes domains.

These capabilities significantly enhance the research capabilities and contribute to a deeper understanding of complex biological systems.

Note: The GE module is part of a larger system, and additional submodules and functionalities may exist to further enhance the research and analysis capabilities in genomics and exposomes.

Database Management

The Database Management within the GE module provides two main functions:

- Direct access to database tables for retrieving information
- Synchronization of the IGEM Client DB with the latest data from the Hall Lab DB Server.

Direct Access to GE.db Tables

Enables direct access to the database tables, allowing users to retrieve information directly from the IGEM Client DB.

This functionality provides a convenient way to query and analyze the data stored in the database tables.

By leveraging this function, users can efficiently retrieve specific information from the IGEM Client DB and utilize it for their research and analysis purposes.

The available tables are:

- datasource
- connector
- term_group
- term_category
- term
- ds_column
- prefix
- wordterm
- termmap
- wordmap

Python function

get_data

The get_data() function allows extracting data from the GE database and loading this data into a Pandas DataFrame structure or CSV File.

It has an intelligent filter mechanism that allow you to perform data selections simply through a conversion layer of function arguments and SQL syntax. This allows the same input arguments regardless of implemented database management system.

Parameters:

Only the table parameter will be mandatory, the others being optional, and will model the data output. In the case of only informing the table, the function will return a DataFrame with all the columns and values of the table.

- **table: str**

  datasource, connector, ds_column, term_group, term_category, term, prefix, wordterm, termmap, wordmap

- **path: str**

  With this parameter, the function will save the selected data in a file in the directory informed as the parameter argument. In this scenario, data will not be returned in the form of a Dataframe; only a Boolean value will be returned, informing whether the file was generated or not

- **columns: list[“str”]**

  Columns that will be selected for output. They must be informed with the same name as the database. It is possible to load other data from other tables as long as it correlate. For example, suppose the table only has the term field and not the category field. In that case, you can inform as an argument: “term_id__term_category_id__category”, the system selected the ID of the term, consulted the ID of the category in the Term table, and went to the Category table to choose the category

- **columns_out: list[“str”]**

  If you want to rename the header of the output fields to more familiar names, you can use this parameter, passing the desired names in the same sequential sequence in the parameter columns

- **datasource: Dict{“str”:list[”str”]}**

  Filter argument. It is used to filter datasource, with the dictionary key being the selection argument and the dictionary value being the datasources selected as the filter. Without this parameter, the function will return all datasources

- **connector: Dict{“str”:list[”str”]}**

Filter argument. It uses the same logic as the datasource, but applied to the connector field

- **word: Dict{“str”:list[”str”]}**
  Filter argument. It uses the same logic as the datasource, but applied to the word field
- **term: Dict{“str”:list[”str”]}**
  Filter argument. It uses the same logic as the datasource, but applied to the term field
- **term_category: Dict{“str”:list[”str”]}**
  Filter argument. It uses the same logic as the datasource, but applied to the term_categorty field
- **term_group: Dict{“str”:list[”str”]}**
  Filter argument. It uses the same logic as the datasource, but applied to the term_group field

Return:

Pandas Dataframe or Boolean (If the parameter path is informed, the function will generate the file; if successful, it will return the TRUE. Otherwise, it will return FALSE)

Examples:

```
>>> from igem.ge import db
>>> db.get_data(
      table=”datasource”,
      datasource={“datasource__in”: [“ds_01”,”ds_02”]},
      columns=[“id”,”datasource”],
      columns_out=[“Datasource ID”, “Datasource Name”],
      path=”{your_path}/datasource.csv”
      )

>>> df = db.get_data(
      table="connector",
      connector={"connector__start": ["conn_ds"]},
      datasource={"datasource_id__datasource__in": ["ds_01"]},
      columns=["connector", "status"]
      )

>>> x = db.get_data(
      table="termmap",
      term={"term_id__term": "chem:c112297"},
      path="{your_path},
      )
   If x:
      print("file created")
```

Command Line

Within the parameters, inform the same ones used for the functions, as well as the arguments, example:

```
$ $ python manage.py db --get_data 'table="datasource", datasource={“datasource__in”: [“ds_0
```

Get data:

```
$ python manage.py db --get_data {parameters}
```

Synchronization with the Hall Lab DB Server

The second function of the Database Management is to synchronize the IGEM Client DB with the latest data from the Hall Lab DB Server.

This synchronization process ensures that the IGEM Client DB is up to date with the most recent information available.

The function offers both offline and online synchronization options.

Offline Sync:

In the offline synchronization mode, users manually acquire the necessary DB files from a designated source. They can obtain the latest versions of the DB files from an authorized repository and update the IGEM Client DB accordingly. This mode is suitable for situations where internet connectivity is limited or when users prefer to have full control over the synchronization process. Examples:

```
>>> from igem.ge import db
>>> db.db.sync_db(table="all", source="{your_path}")
```

Online Sync:

The online synchronization mode automates the process of fetching the latest data from the web repository. The submodule accesses the web repository and retrieves the most recent versions of the DB files, ensuring that the IGEM Client DB is synchronized with the Hall Lab DB Server. This mode is ideal for users who prefer a seamless and automated synchronization process, without the need for manual intervention. Examples:

```
>>> from igem.ge import db
>>> db.db.sync_db(table="all")
```

The GE.db submodule provides researchers with a comprehensive set of tools to access and synchronize the IGEM Client DB. Whether it's directly querying database tables or ensuring up-to-date information through synchronization, this submodule facilitates efficient data management and enhances the research capabilities of users.

Reports

The GE.filter module serves as a crucial component of the GE (Genomics and Exposomes) system, specifically designed to facilitate the exploration and analysis of the Knowledge Database, referred to as GE.db. This Knowledge Database contains a wealth of information related to genomics, exposomes, and their interconnectedness.

By utilizing the functions provided by GE.filter, users gain the ability to efficiently retrieve and filter data from GE.db, enabling them to uncover valuable insights and relationships.

Whether it's examining term connections, exploring reports on GxE (Gene-Environment) interactions or ExE (Exposome-Environment) associations, accessing gene-level information in relation to SNPs (Single Nucleotide Polymorphisms), or converting words to IGEM terms, the GE.filter module empowers users to extract pertinent information and generate comprehensive reports.

These functionalities play a crucial role in understanding the complex interplay between genomics and exposomes, supporting various research and analytical endeavors

- term_map: The term_map function provides the mapping between IGEM terms and their associated metadata. It enables you to explore the attributes and properties of different terms stored in the GE.db, aiding in data exploration and analysis.
- word_to_term: This function allows you to convert individual words or a list of words into their corresponding IGEM terms. It helps in mapping user-provided words to the relevant terms stored in the GE.db, providing a standardized representation for further processing.
- gene_exposome: The gene_exposome function retrieves information about the gene-exposome relationship from the GE.db. It helps in understanding the interaction between genes and environmental factors, facilitating studies related to genomics and exposomes.
- snp_exposome: With the snp_exposome function, you can access reports and information about the impact of single nucleotide polymorphisms (SNPs) on exposomes. It helps in understanding the influence of genetic variations on environmental exposures and their potential effects on health outcomes.
- word_map: In the Word-Map function, all words mapped from an external dataset are stored in a temporary table within GE.db. This feature proves particularly useful for researchers who wish to list the relationships between words on a record-by-record basis, without relying on the IGEM pre-computing mapping process that converts external words to the standardized IGEM Terms. It allows users to perform analysis and retrieve word relationships specific to their research needs. However, it's important to note that this temporary table should be used judiciously due to its high memory consumption on the database. Users are advised to run the function on a specific dataset, extract the desired relationships for their analysis, and subsequently clean up this information to optimize database performance. By providing a flexible and efficient way to explore word relationships, the Word-Map function empowers researchers in their investigations and enhances their understanding of the data.

Parameters File

ge.filter.**parameters_file** (path=None)
generates a model file to be used as a parameter file in query functions

Parameters

- path**: str**
  path where the file will be generated.

File layout

In the file structure, new lines for the index filter can be included with additional values, and each filter line must contain only a single value. The output index and path must be unique, as they will be applied to the entire corresponding field (parameter).

In the example below, let's select all terms from two data sources from a single group. Also, the Datasource and Connector fields will be aggregated and will not appear on the results

```
index,parameter,value
filter,datasource,ds_01
filter,datasource,ds_02
filter,connector,
filter,term_group,Chemical
filter,term_category,
filter,word,
output,datasource,no
output,connector,no
output,term_group,
output,term_category,
output,term,
output,word,no
path,path,/../output_file.csv
```

Return

it return a boolean value if the file was created

Examples:

```
from igem.ge import filter
filter.parameters_file(
      path="../../folder"
      )
```

This function generates a file template with parameters created in the specified path.

Word Map

ge.filter.**word_map** (*args, **kwargs)
Queries GE.db and returns links between words without terms.

Parameters

- path_in**: str**
  parameter file path with filter information, aggregation, and result file path.
- path_out**: str**
  result file path.

- `term`**: list[str]**

  List of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

Return

It may return a boolean value if you have informed an output per file (`path_out`) or a DataFrame if you have not informed an output file.

Examples:

```
from igem.ge import filter
filter.word_map(
      path_in="../../file.csv",
      path_out="../../outcome.csv"
      )
```

This function queries GE.db and generates results showing links between words without terms.

The results can be saved in a specified output file path or returned as a DataFrame.

Term Map

`ge.filter.`**`term_map`**`(*args, **kwargs)`

TermMap table query function.

Parameters

- `path_in`**: str**

  parameter file path with filter information, aggregation, and result file path.

- `path_out`**: str**

  result file path.

- `term`**: list[str]**

  List of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

Return

It may return a boolean value if you have informed an output per file (`path_out`) or a DataFrame if you have not informed an output file.

Examples:

```
from igem.ge import filter
filter.term_map(
      path_in="../../file.csv",
      path_out="../../outcome.csv"
      )
df_result = filter.term_map(
      term=["gene:246126"]
      )
```

This function queries the TermMap table in GE.db and retrieves relationships between terms. The results can be saved in a specified output file path or returned as a DataFrame.

Words to Terms

ge.filter.**word_to_term**(path=None)
Perform a search for terms from a string base with the same ETL engine.

Parameters

- path**: str**
  File with the strings for conversion into terms. Only the first column of the file will be processed.

Return

A file will be generated with the results in the same folder as the input strings file.

Examples:

```
from igem.ge import filter
filter.word_to_term(
      path='../../file.csv'
      )
```

This function searches for terms from a string base using the ETL engine. It takes a file path as input, reads the strings from the file, and converts them into terms. The results are saved in a CSV file in the same folder as the input file.

Gene Exposome Report

ge.filter.**gene_exposome**(*args, **kwargs)
Queries GE.db and returns links between genes and exposomes based on input parameters or the parameter file.

Parameters

- path_in**: str**
  parameter file path with filter information, aggregation, and result file path.
- path_out**: str**
  result file path.
- term**: list[str]**
  List of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

Return

It may return a boolean value if you have informed an output per file (path_out) or a DataFrame if you have not informed an output file.

Examples:

```
from igem.ge import filter
filter.gene_exposome(
      path_in="../../file.csv",
      path_out="../../outcome.csv"
      )
df_result = filter.gene_exposome(
      term=["gene:246126"]
      )
```

This function queries GE.db and generates results showing links between genes and exposomes based on the provided parameters. The results can be saved in a specified output file path or returned as a DataFrame.

SNP Exposome Report

`ge.filter.`**`snp_exposome`**(`*args`, `**kwargs`)

Queries GE.db and returns links between SNPs and exposomes based on input parameters or the parameter file.

Parameters

- `path_in`**: str**

  parameter file path with filter information, aggregation, and result file path.

- `path_out`**: str**

  result file path.

- `term`**: list[str]**

  List of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

Return

It may return a boolean value if you have informed an output per file (`path_out`) or a DataFrame if you have not informed an output file.

Examples:

```
from igem.ge import filter
filter.snp_exposome(
      path_in="../../file.csv",
      path_out="../../outcome.csv"
      )
df_result = filter.snp_exposome(
      term=["gene:246126"]
      )
```

This function queries GE.db and generates results showing links between SNPs and exposomes based on the provided parameters. The results can be saved in a specified output file path or returned as a DataFrame.

Tags

A TAG in the context of the GE.Filter function is a unique identifier that helps you track and identify the version of the external dataset used in the IGEM query. It serves as a reference to the specific dataset version, allowing you to reproduce the same query in the future with the same dataset version.

When you process (ETL) a specific external dataset, the TAG will indicate the version of the dataset used. For example, if you process one external dataset with version or ETAG 1, the TAG will show that the data comes from this dataset.

In case the IGEM database is updated with a newer version of the external dataset, the TAG will reflect the most recent version. This helps researchers know which external dataset was used in their IGEM query and enables them to control and ensure the consistency of results when they want to replicate the same query in the future.

By referring to the TAG, researchers can track and document the specific dataset version used, providing transparency and facilitating reproducibility in their research.

In summary, the TAG serves as a unique identifier that indicates the version of the external dataset used in the IGEM query, allowing researchers to reproduce the same query with the same dataset version in the future.

`ge.filter.`**`create_tag`**(`connectors`)

Function to create a TAG with Current connector in TermMap. The IDs generated in the TAG are the WFControl table IDs with Current status.

Parameters

- `connectors`**: list**
  List of connector names.

Returns

str
Generated TAG string.

This function generates a TAG string based on the current connector in TermMap. It takes a list of connector names as input and retrieves the last IDs with a “Current” status from the WFControl table. It then generates a TAG string using the retrieved IDs. The TAG string helps researchers know which external dataset was used in the IGEM query and provides control to use the same version of the dataset in future consultations to IGEM.

The generated TAG string follows the format “GE.db-TAG:<tag_ids>”, where “<tag_ids>” is a hyphen-separated string of the WFControl table IDs.

`ge.filter.`**`get_tag`**`(tag)`
Function to retrieve WFControl data based on a TAG.

Parameters

- `tag`**: str**
  TAG string.

Returns

pd.DataFrame
DataFrame with WFControl data.

This function retrieves WFControl data based on a TAG string. It takes a TAG string as input and parses the TAG to obtain the corresponding WFControl table IDs. It then retrieves the WFControl data for the specified IDs and returns it as a DataFrame.

`ge.filter.`**`get_tag_data`**`(tag, path)`
Function to save TermMap and WFControl data to CSV files based on a TAG.

Parameters

- `tag`**: str**
  TAG string.
- `path`**: str**
  Path to save the files.

Returns

bool
True if the files were successfully created, False otherwise.

This function saves the TermMap and WFControl data associated with a TAG string to CSV files. It takes a TAG string and a path as input. First, it retrieves the corresponding WFControl data using the `get_tag` function. Then, it retrieves the TermMap data related to the connector IDs in the WFControl data. Finally, it saves both the TermMap and WFControl data to separate compressed CSV files in the specified path. The function returns True if the files were successfully created and False otherwise.

Server Application

The Server module serves as the backbone of the IGEM project, facilitating the seamless integration of external data into the comprehensive Exposomes and Genomics knowledge base. With its powerful ETL capabilities and efficient database management, the Server module ensures the harmonious incorporation of diverse datasets into the project ecosystem.

By leveraging advanced ETL techniques, the Server module extracts valuable information from external sources, transforms it into a standardized format, and loads it into the central knowledge base. This allows researchers and users to access and analyze a wealth of integrated data related to Exposomes and Genomics effortlessly.

Furthermore, the Server module assumes the critical responsibility of maintaining and managing the IGEM database, ensuring its accuracy, reliability, and accessibility. It oversees the seamless incorporation of new data, performs updates, and organizes the information in a coherent and user-friendly manner.

Through the collaborative efforts of the Server module, IGEM continues to expand its reach and depth of knowledge, empowering researchers and stakeholders to explore the intricate connections between Exposomes and Genomics. By bridging the gap between external data sources and the project's standardized knowledge base, the Server module serves as a catalyst for groundbreaking discoveries and valuable insights.

This user guide will provide you with comprehensive instructions on utilizing the features and functionalities offered by the Server module, enabling you to harness the full potential of the integrated Exposomes and Genomics database. Let's embark on this transformative journey together and unlock the power of data-driven exploration in the realm of IGEM.

Master Data

The master data module plays a vital role in the functioning of the system by directing data flow, filtering information, and establishing connections with the knowledge base. It enables efficient data collection and integration processes while facilitating effective filtering and linking of terms.

Before initiating the data collection, it is necessary to parameterize the master data module, which can be done either in batch processing or through an intuitive web interface. This allows users to configure the module according to their specific requirements, providing flexibility and ease of use.

The configuration of master data involves the following components, each building upon the previous one:

Datasource: Define the source of the data, specifying its origin or location from which it will be collected.

Connector: Establish a connection between the system and the designated data source, enabling seamless data retrieval and integration.

Terms: Define individual terms or keywords relevant to the system's knowledge base. Terms act as key identifiers for data retrieval and linkage. Each term can be associated with attributes such as Group and Category, which help users filter and organize terms based on specific criteria or themes.

Prefix: Specify prefixes or identifiers to be appended to certain terms, enhancing their contextual meaning and facilitating accurate data interpretation.

By configuring the master data module in this manner, users gain the ability to intelligently handle data origin, perform precise filtering based on Groups and Categories, and establish seamless connections to the knowledge base. This ensures the accuracy, relevance, and integrity of the data, empowering comprehensive analysis and meaningful insights.

The user-friendly approach to managing master data within the Server module enables users to leverage the full potential of the IGEM system. It provides efficient data integration, discovery, and exploration, while facilitating flexible filtering and organization of terms based on their associated attributes. This empowers users to efficiently navigate and extract relevant information from the knowledge base, facilitating their research and analysis activities.

Datasource

Datasource master data refers to an external data source and groups of Connectors over the same domain. The connection between IGEM and external data sources will be established by Connectors.

Datasource is used for selecting and grouping queries and future security and authentication features.

The Datasource data will be stored in the ge_datasource table of the IGEM DB defined in the initial parameters. The available fields are:

- *ID*: GE.db internal key
- *Datasource*: Abbreviated name of the Datasource
- *Description*: Description for identifying and consulting the Datasource

- *Category*: Category to help identify and group the Datasource
- *Website*: Electronic address of the maintainer of the available data

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ python manage.py db --load_data "table='datasource, path='{your_path}/datasource.csv'"
```

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Connector and TermMap information

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out Datasource management activities.

Activate the IGEM web service if you have not already done so. Go to the igem folder and type the command line:

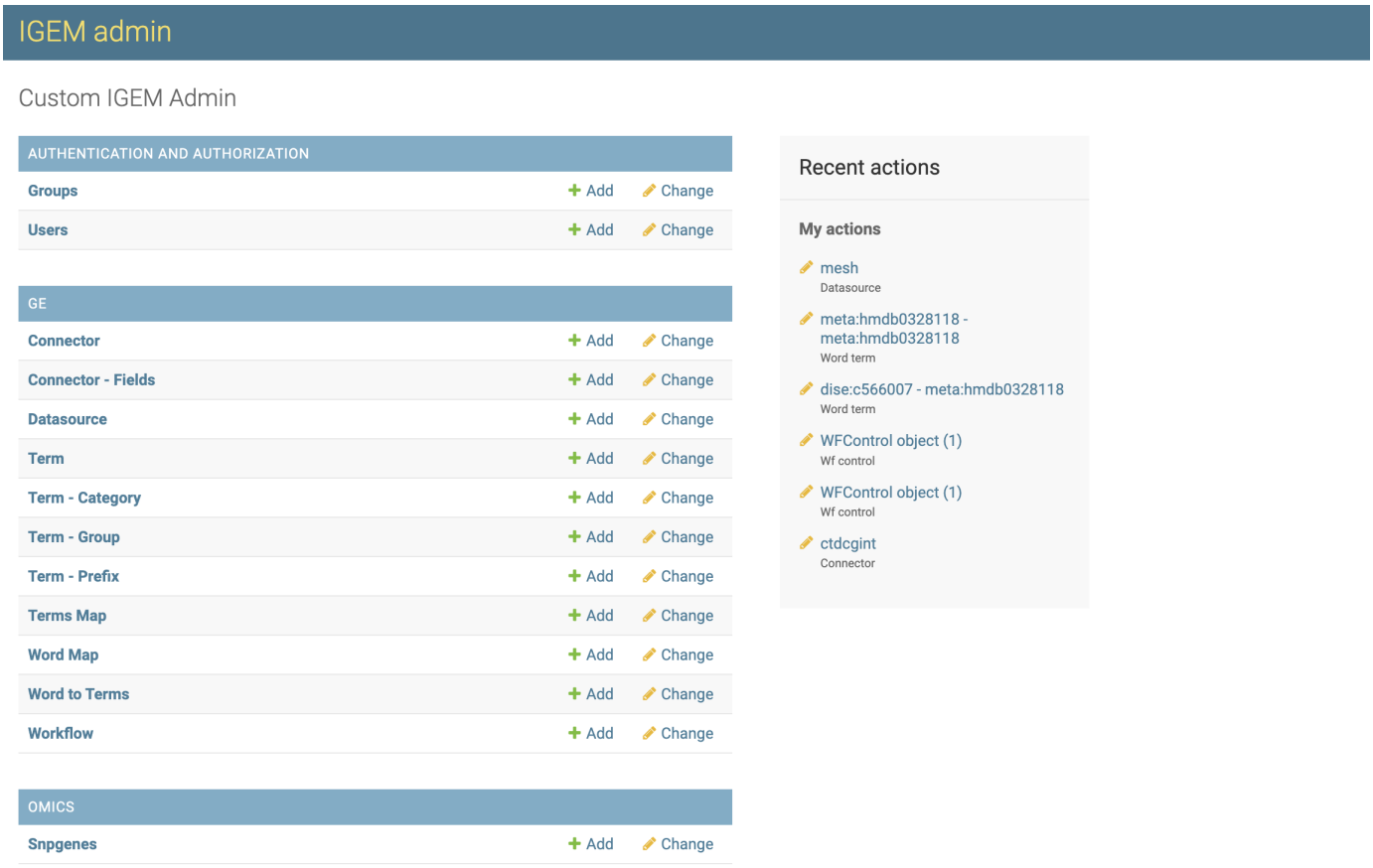
```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

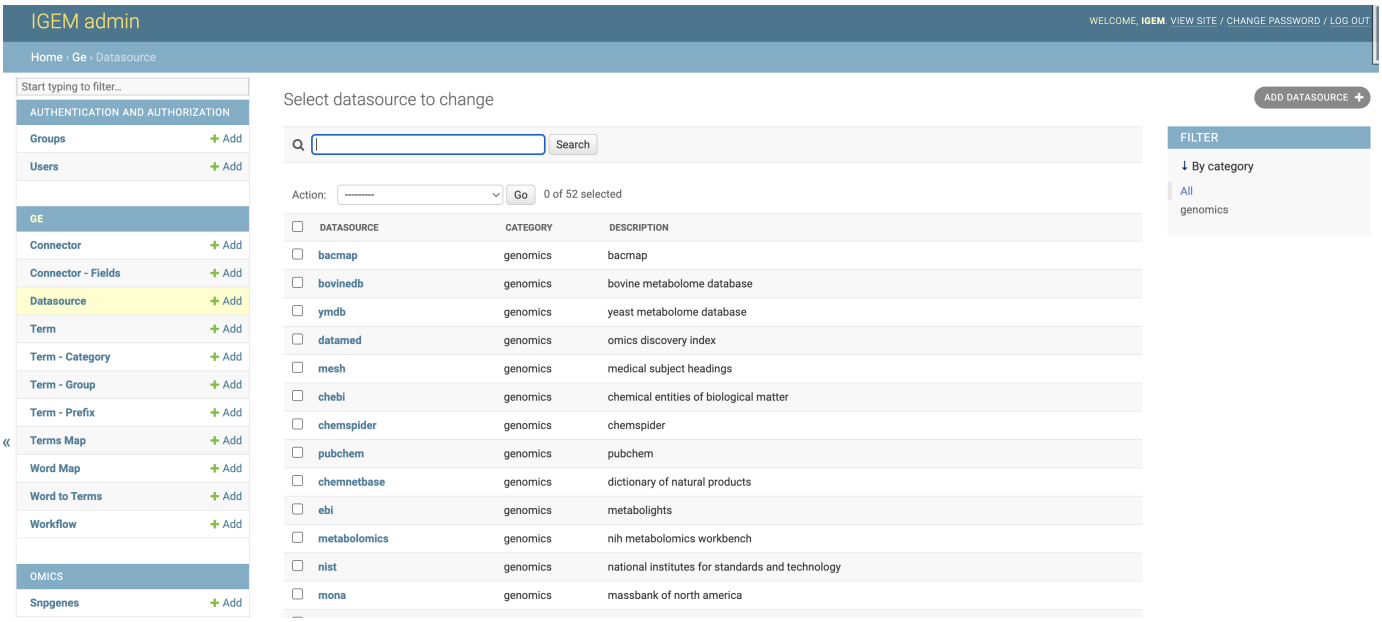
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

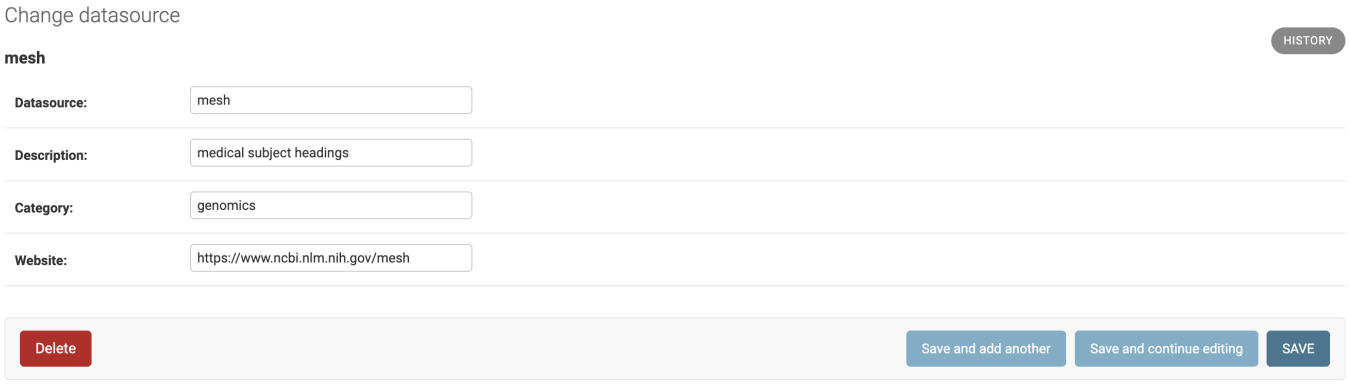
After user authentication and on the initial administration screen, select an option Datasource.



On the Datasource screen, we will have options to consult, modify, add and eliminate Datasource.



On the first screen, we have a view of all available Datasource. To consult, click a desired Datasource.



On the next screen, we have all the Datasource fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank Datasource screen to add a new Datasource record.
- `Save and Continue editing`: Will save the changes and continue on the Datasource screen.
- `Save`: Will save the changes and return to the screen with the list of Datasource.

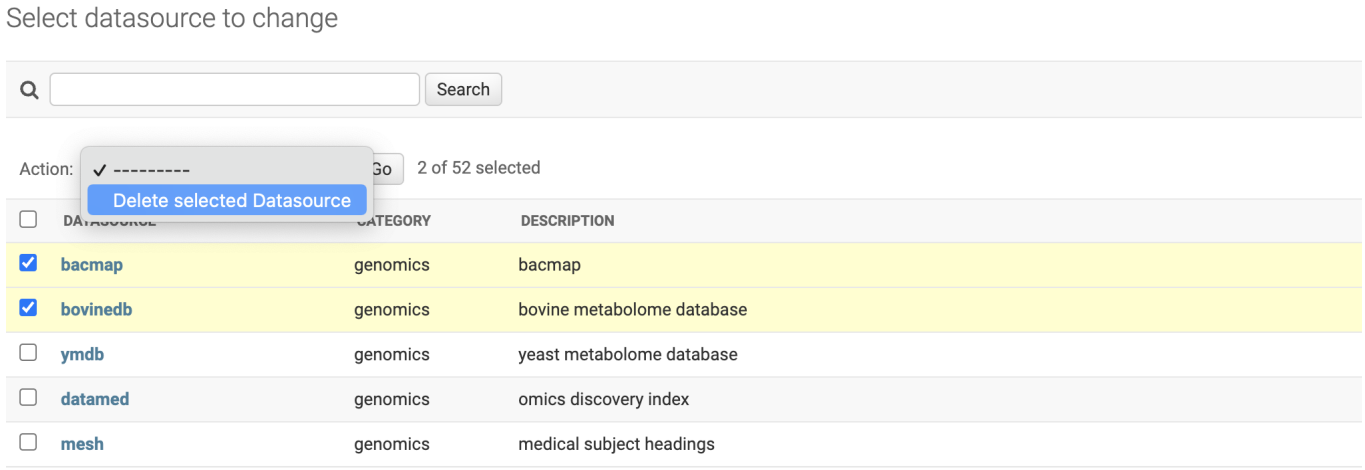
In the History button, we can consult all the modifications carried out in the Datasource, this function will be important to track modifications and audit the process.

The `DELETE` button will permanently delete the Datasource record.

Caution: when deleting a Datasource, the system will also delete all records dependent on that Datasource, which include Connector, Parameterizations of transformations and TermMaps

Deletion can also be performed en bloc. On the Datasource List screen, select all the Datasource you want to delete, choose the Delete Selected Datasource action and click on the `GO` button.

Be careful, this elimination operation will be definitive for the Datasource and for all other records dependent on it, as already explained.



To add new Datasource, we will have three different ways:

- by the + Add button on the left sidebar.
- Through the ADD DATASOURCE + button in the right field of the Datasource list.
- Via the Save and add another button located within a Datasource record.

For the Datasource, we will have two filter locations:

- First located at the top of the Datasource List screen where we can search broadly.
- Second on the right sidebar, being able to select by category of Datasource.

Connector

A Connector record stores all information needed for external extraction and controls the links of related terms by the mapping process. The control of external data extraction occurs by Connector. Each Connector will consist of the fields:

- *Datasource*: Grouping of Connector, controlling access authentication.
- *Connector*: Abbreviation for Connector identification
- *Description*: Brief description of the purpose of Connector
- *Enabled*: Flag informing if the Connector is active and will be considered in the ETL process

Group Attributes: Fields to control extraction path and file type

- *Source path from Internet*: Flag to route the extraction path. If enabled, it will be via HTTP and disabled. It will be considered as a local file path.
- *Source path*: Path where the Connector file is hosted
- *Source file name*: name of the file with the original data
- *Source file format*: file format with the original data. This information will be imported for conversion treatment to the data ingestion format in the ETL process. If compressed, inform only the compression format, type ZIP, GZ.
- *Source file sep*: inform the type of file separator if any. For tabular division, use /n
- *Source file skip row*: Inform the number of lines eliminated in the ETL process. Many files have structural information in their first lines that are not needed in the ETL process.
- *Source compact*: Flag to control if the file is compressed. If not marked, it is considered an uncompressed file.
- *Target file name*: Name of the file after unzipping
- *Target file format*: File format after unzipping, this field will be the actual file format, type CSV, TXT

- *Keep file*: Flag, if selected, will keep the file after data processing for future reference. It is essential to analyze the storage space consumption, as keeping the files may consume unnecessary space. New updates will overwrite existing files.

Group Columns: Controls rules for handling the extracted data in a format compatible with the GE.db system. Consider only columns with standard MEsH NIH codes.

- *Column Sequence*: Number of the column that receives the rule
- *Column Name*: Column name to guide and help identify the applied rule
- *Active*: a flag that informs if the practice is active
- *PREFIX*: inform the prefix of the word that will be considered and added to the column information.

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ $ python manage.py db --load_data "table='connector, path='{your_path}/connector.csv'"
```

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Rules and TermMap information

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out Connector management activities.

Activate the IGEM web service if you have not already done so. Go to the /src/ folder and type the command line:

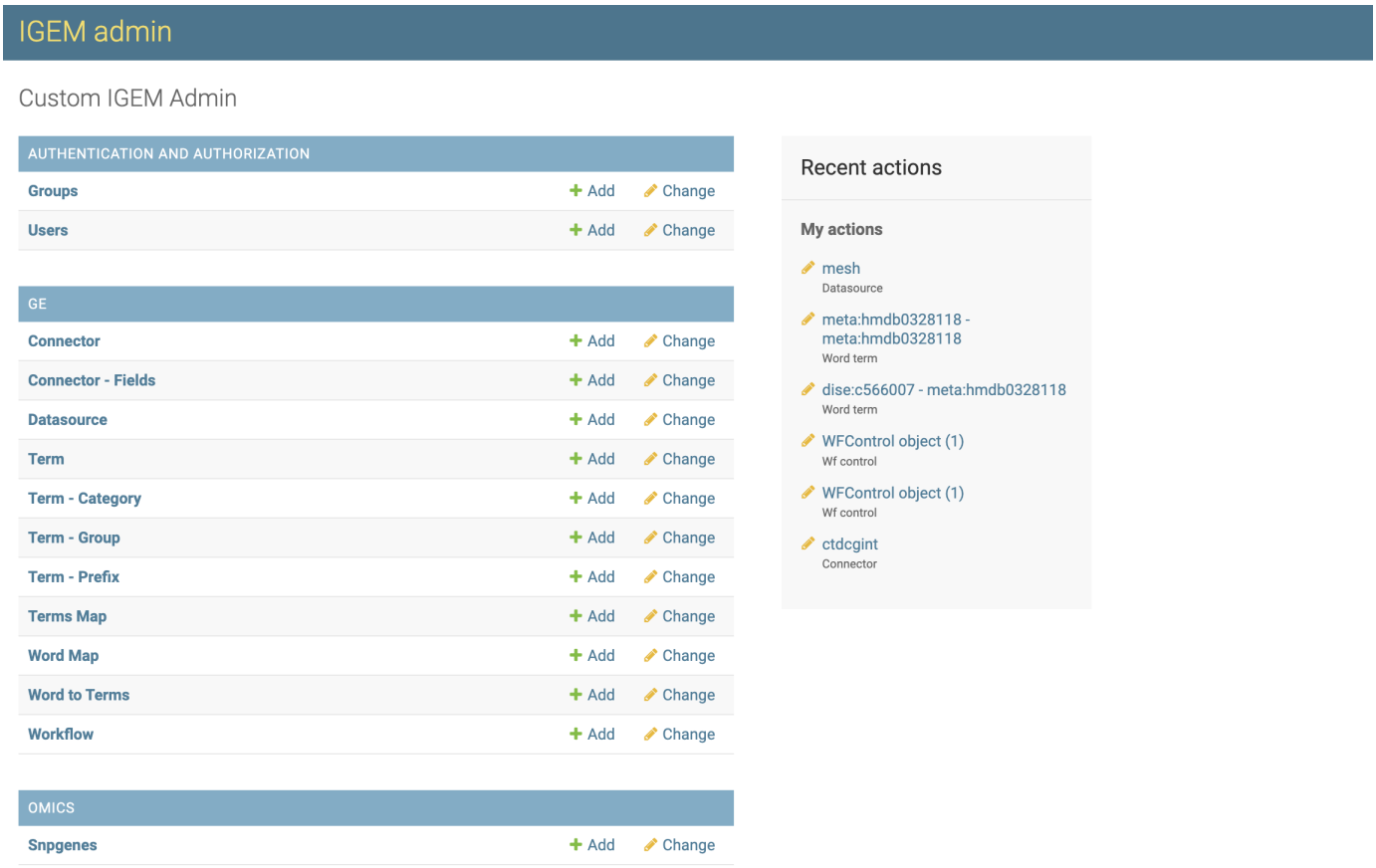
```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

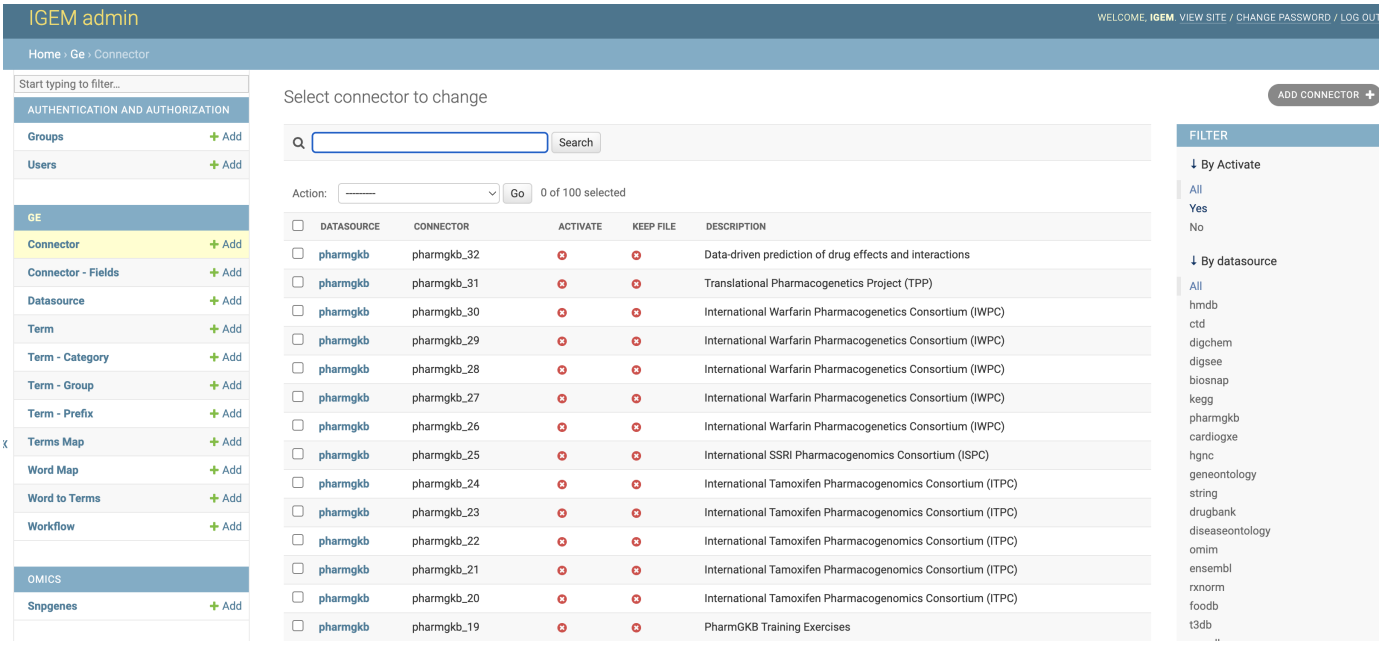
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

After user authentication and on the initial administration screen, select an option Connector.



On the Connector screen, we will have options to consult, modify, add and eliminate Connector.

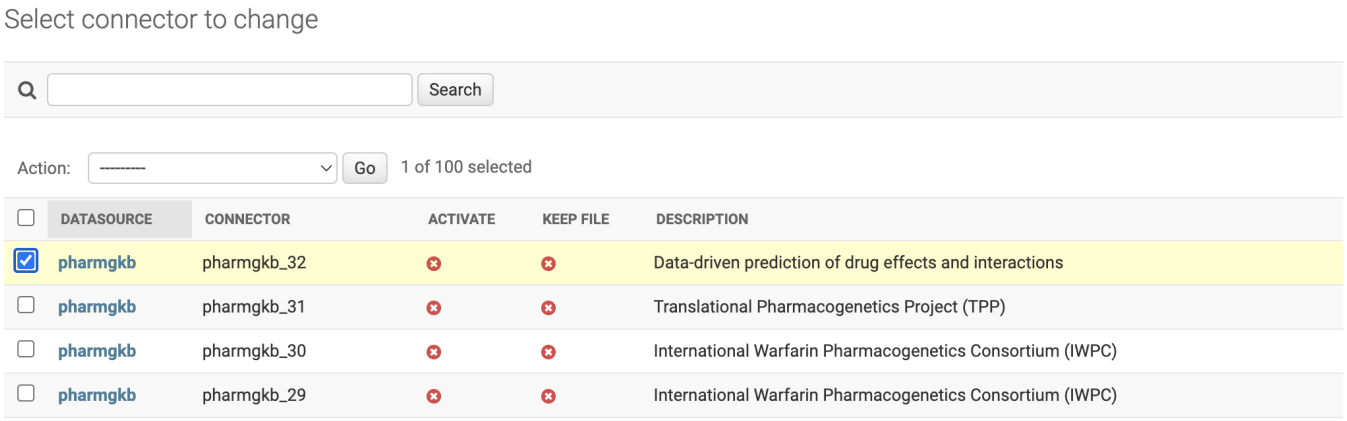


Filter Connector

For the Datasource, we will have two filter locations:

- First located at the top of the Datasource List screen where we can search broadly.
- Second on the right sidebar, being able to select by category of Datasources.

To consult, click a desired Connector.



Add Connector

To add new Datasource, we will have three different ways:

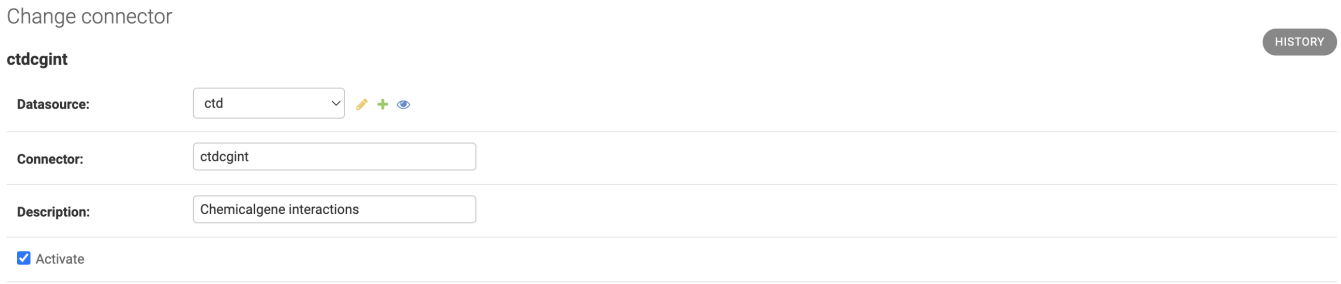
- by the + Add button on the left sidebar.
- Through the ADD Datasource + button in the right field of the Datasources list.
- Via the Save and add another button located within a Datasource record.

The Connector interface is divided into three parts with each one performing a different activity in the ETL (Extraction, Transformation and Load) process of external data to IGEM.

PART 1

It has fields for identifying the Connector, among them we will have which Datasource the Connector belongs to, an abbreviation that will identify the Connector in processes and queries, a description and a flag for activating or not the Connector.

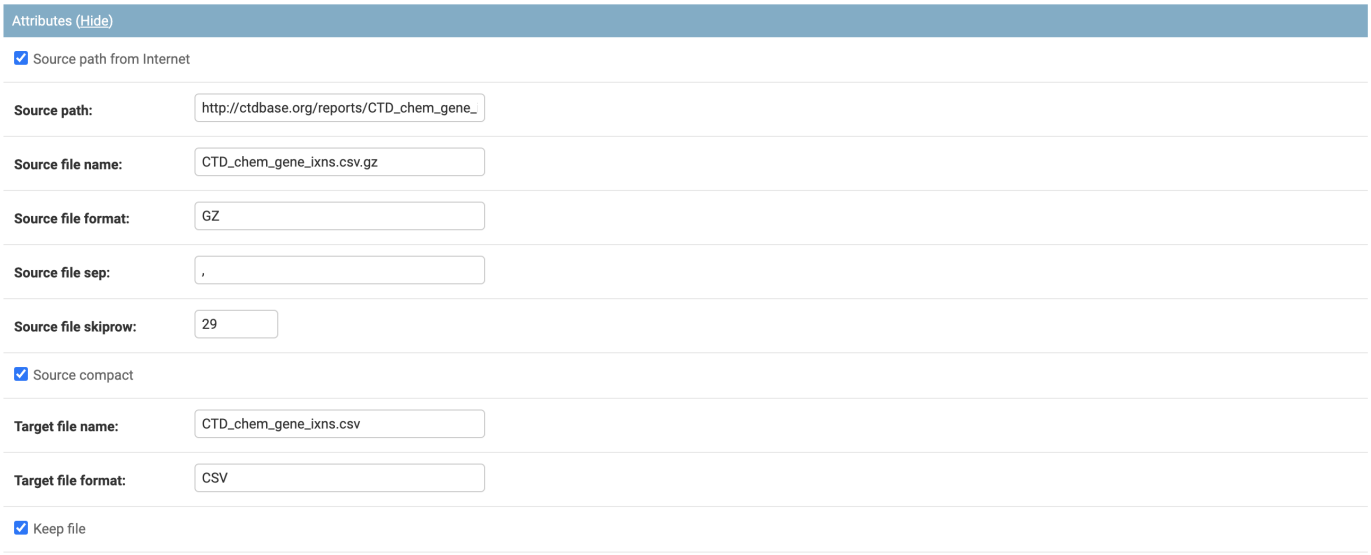
If the Activate FLAG is not selected, the Connector will not perform new extraction of external data, but all data already loaded will continue to be available for queries



PART 2

The second part will be composed of the attributes to make the connection with external sources and format of the extracted data.

Each field has already been detailed at the beginning of this session.



PART 3

The third part stores the transformation rules that the IGEM system needs to interpret the input data and how to handle each column.

If the input files are not in tabular format, as in the case of files with XML extension, the COLLECT process will transform into tabular and keep a new file in the PSA folder as detailed in the Collect

The sequence of columns does not need to be in order, however it will be necessary to identify the first column with the number 0 and so on.

If a column is not informed, the system will understand it as a basic rule and process the column as detailed in the process Prepare

To delete a column rule, check the DELETE? On the desired line and save the Connector.

To add a new column rule, select the + Add another DST column button

The column name will be a header identifier used in the file generated during the ETL process. This file can be kept for future queries and Data Lake projects for example, as it has already passed through a normalization layer.

The Active field tells the ETL process to read and handle the source column. If the line is not marked, it will be discarded during the PREPARE process and not going through MAPREDUCE.

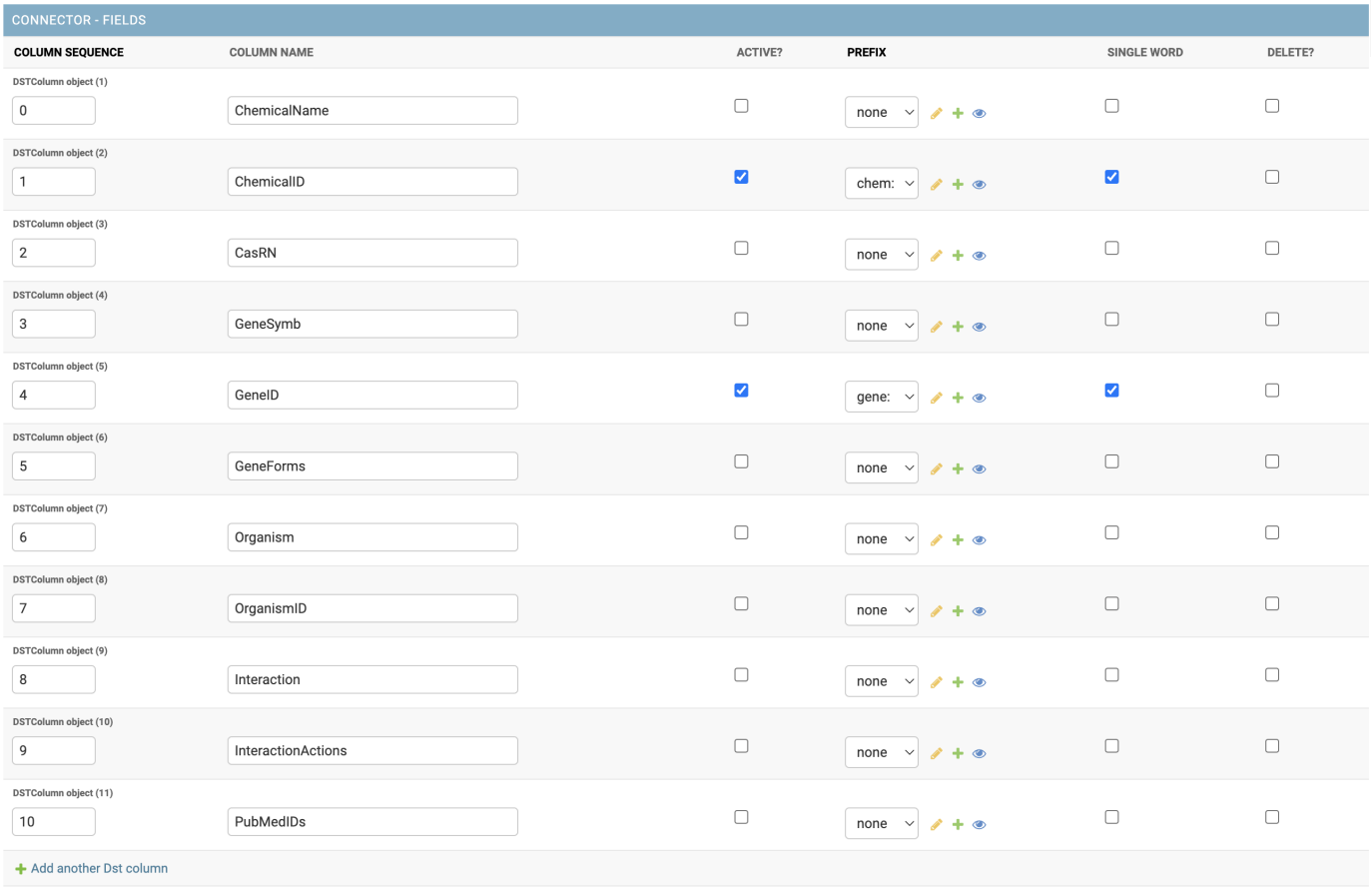
Important to balance workload, storage space, and other factors before keeping a column active. Many columns are unnecessary information for the purpose of term mapping and would not make sense to keep them active.

The prefix field adds an identification to fields composed only of codes: For example, we will have a column with a code of a Numeric Gene only, for the IGEM to identify that this sequence of numbers is a Gene, we add a prefix that differentiates it from other types of also numerical information.

It is important to maintain a synchronism between the prefixes and the registration of Keyge and Words so that the MAPREDUCE process can identify and ingest the terms found into the GE.db base.

We can perform two maintenance operations on the prefixes using the change buttons and quickly add a new prefix on the Connector screen without having to leave the register. Prefixes are also treated as master data within IGEM.

The SINGLE WORD option field has the function of informing the ETL process if this column is composed of a single word and does not need to go through the process of breaking up and identifying terms. This makes the process faster and reduces memory consumption during the Connector ETL. When we have selected a prefix for the column, the SINGLE WORD will have no function, because with the prefix it is assumed that it is a KEYGE without the replacement processing by the KEYWORD.



Save Connector

On the next screen, we have all the Connector fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank Connector screen to add a new Connector record.
- `Save and Continue editing`: Will save the changes and continue on the Connector screen.
- `Save`: Will save the changes and return to the screen with the list of Connector.

Delete Connector

The `DELETE` button will permanently delete the Datasource record.

Caution: when deleting a Datasource, the system will also delete all records dependent on that Datasource, which include Connectors, Parameterizations of transformations and KEYLINKS

Deletion can also be performed en bloc. On the Datasource List screen, select all the Datasource you want to delete, choose the Delete Selected Database action and click on the `GO` button.

Be careful, this elimination operation will be definitive for the Datasources and for all other records dependent on it, as already explained.

History Change

In the History button, we can consult all the modifications carried out in the Datasource, this function will be important to track modifications and audit the process.

Group

The group master data acts as a qualitative characteristic for the Term being the highest level of the hierarchical structure, followed by the Category and then the Term.

The system uses the Group information as a filter in queries and other interfaces. An example of the use of Group will be the Gene Exposome Report, in which the system will use the Group to select which Term will be considered as Exposome.

The Group data will be stored in the ge_group table of the IGEM DB defined in the initial parameters. The available fields are:

- *ID*: GE.db internal key
- *group*: Abbreviated name of the Group
- *Description*: Description for identifying and consulting the Group

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ python manage.py db --load_data "table='term_group, path='{your_path}/term_group.csv'"
```

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Term and TermMap information

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out Group management activities.

Activate the IGEM web service if you have not already done so. Go to the igem folder and type the command line:

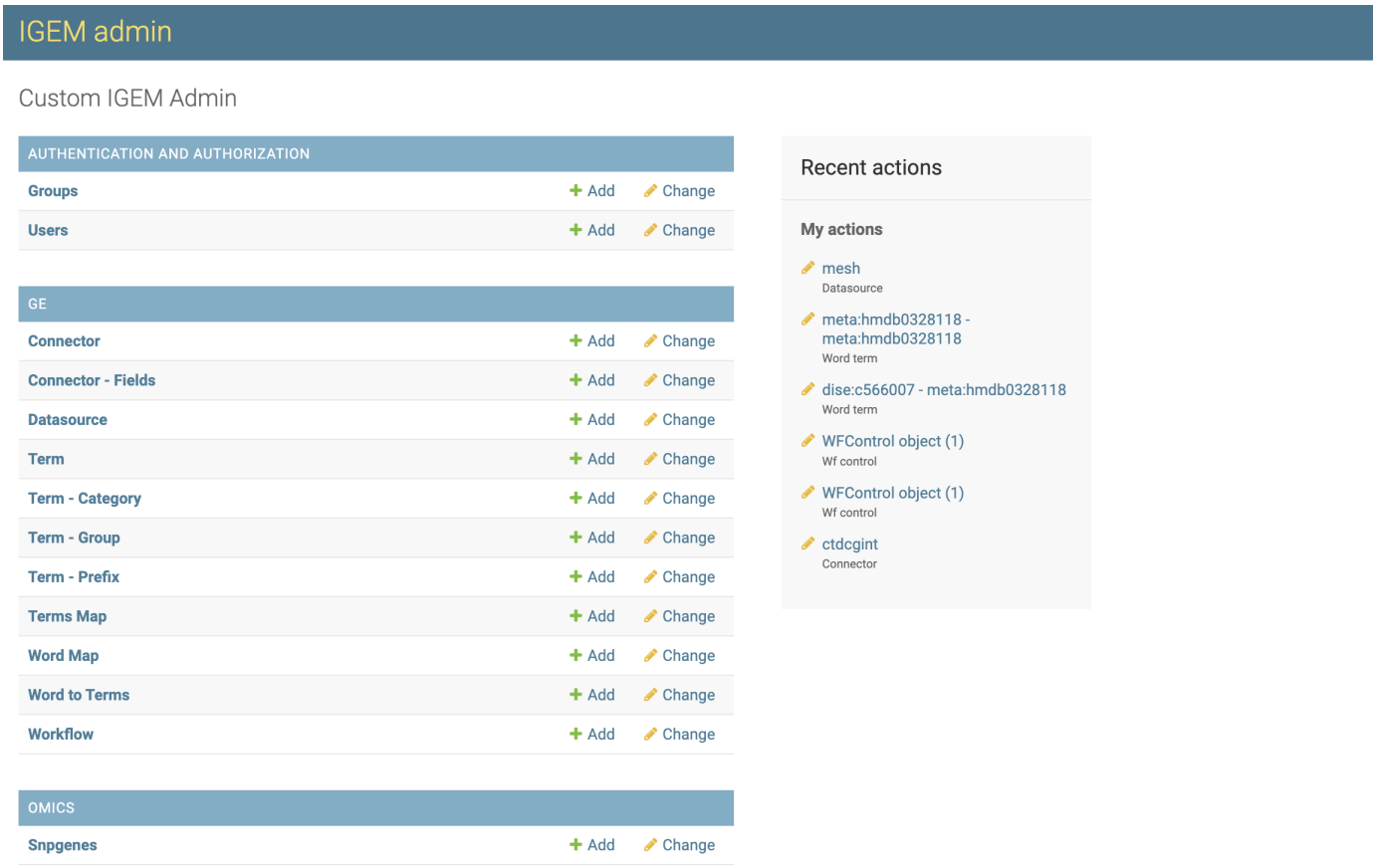
```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

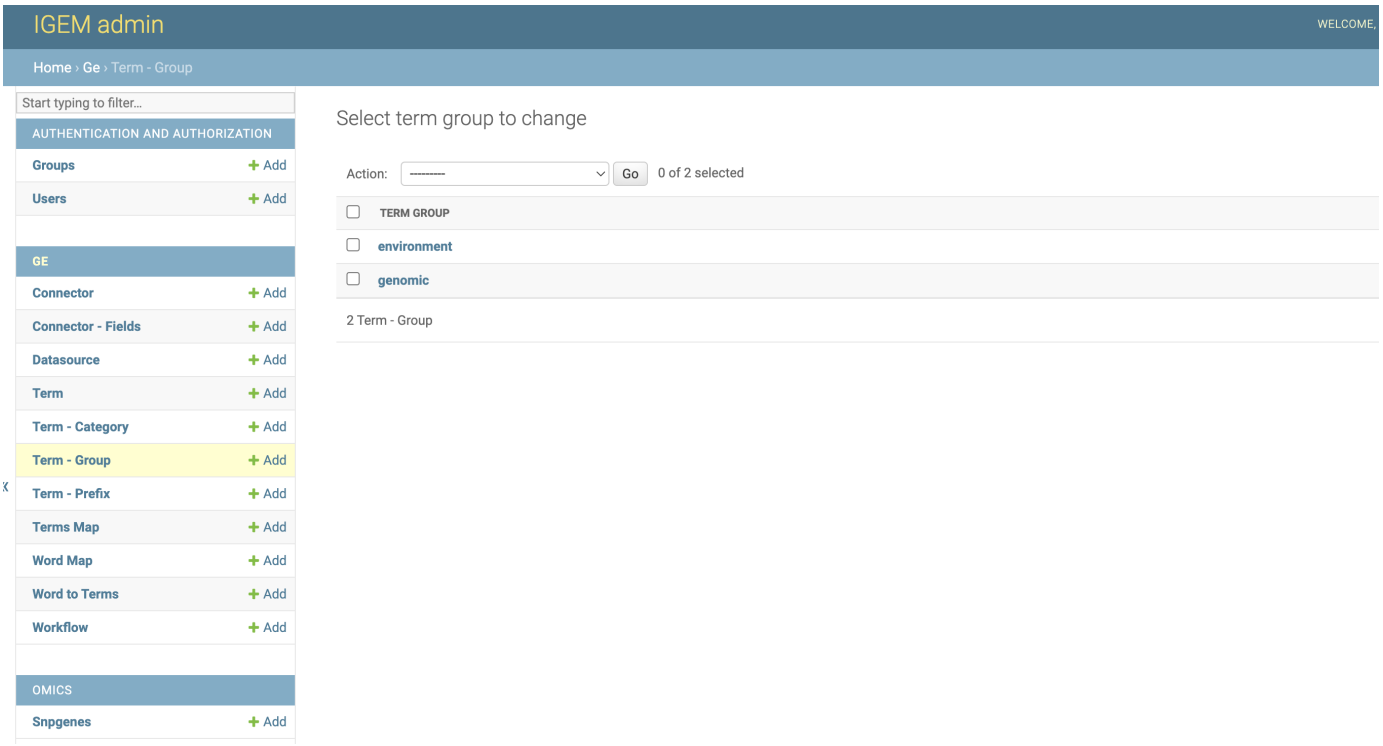
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

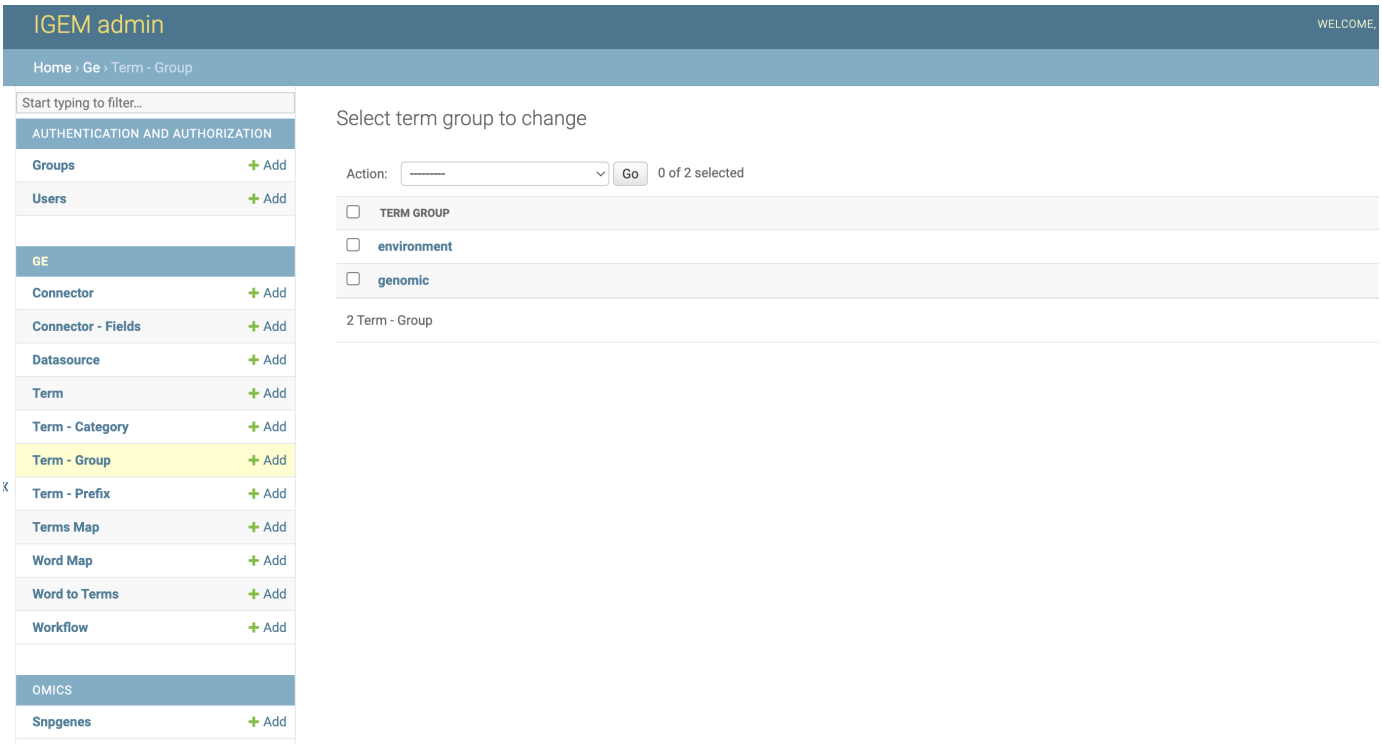
After user authentication and on the initial administration screen, select an option Keyge-Group.



On the Group screen, we will have options to consult, modify, add and eliminate Group.



On the first screen, we have a view of all available Group. To consult, click a desired Group.



On the next screen, we have all the Group fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank Group screen to add a new Group record.
- `Save and Continue editing`: Will save the changes and continue on the Group screen.
- `Save`: Will save the changes and return to the screen with the list of Group.

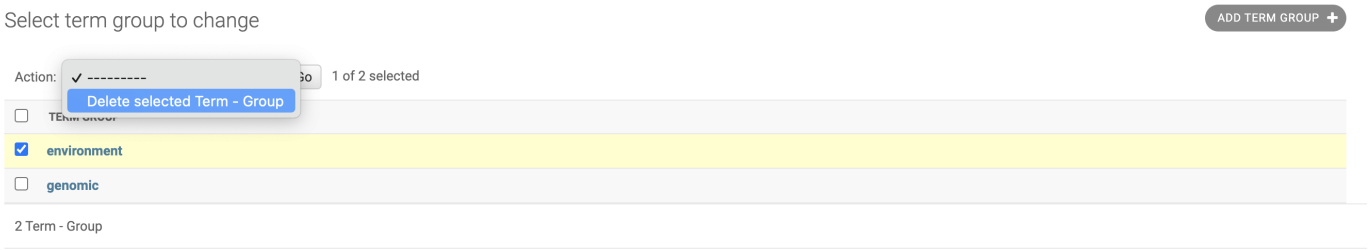
In the History button, we can consult all the modifications carried out in the Group, this function will be important to track modifications and audit the process.

The `DELETE` button will permanently delete the Group record.

Caution: when deleting a Group, the system will also delete all records dependent on that Group, which include Term, and KEYLINKS

Deletion can also be performed en bloc. On the Group List screen, select all the GroupS you want to delete, choose the Delete Selected Keyge - Groups action and click on the `GO` button.

Be careful, this elimination operation will be definitive for the GroupS and for all other records dependent on it, as already explained.



To add new Group, we will have three different ways:

- by the + Add button on the left sidebar.
- Through the `ADD Group` + button in the right field of the GOUP list.
- Via the `Save and add another` button located within a Group record.

Category

Category master data acts as a grouping of Term at a lower level than the Group.

The system uses the Category information as a filter in queries and other interfaces. An example of the use of Category will be the Gene Exposome Report, in which the system will use the Category to select all Gene **KEYGE**.

The Category data will be stored in the ge_category table of the IGEM DB defined in the initial parameters. The available fields are:

- *ID*: GE.db internal key
- *Category*: Abbreviated name of the Category
- *Description*: Description for identifying and consulting the Category

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ $ python manage.py db --load_data "table='term_category, path='{your_path}/term_category.c
```

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Term and TermMap information

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out GROUP management activities.

Activate the IGEM web service if you have not already done so. Go to the igem folder and type the command line:

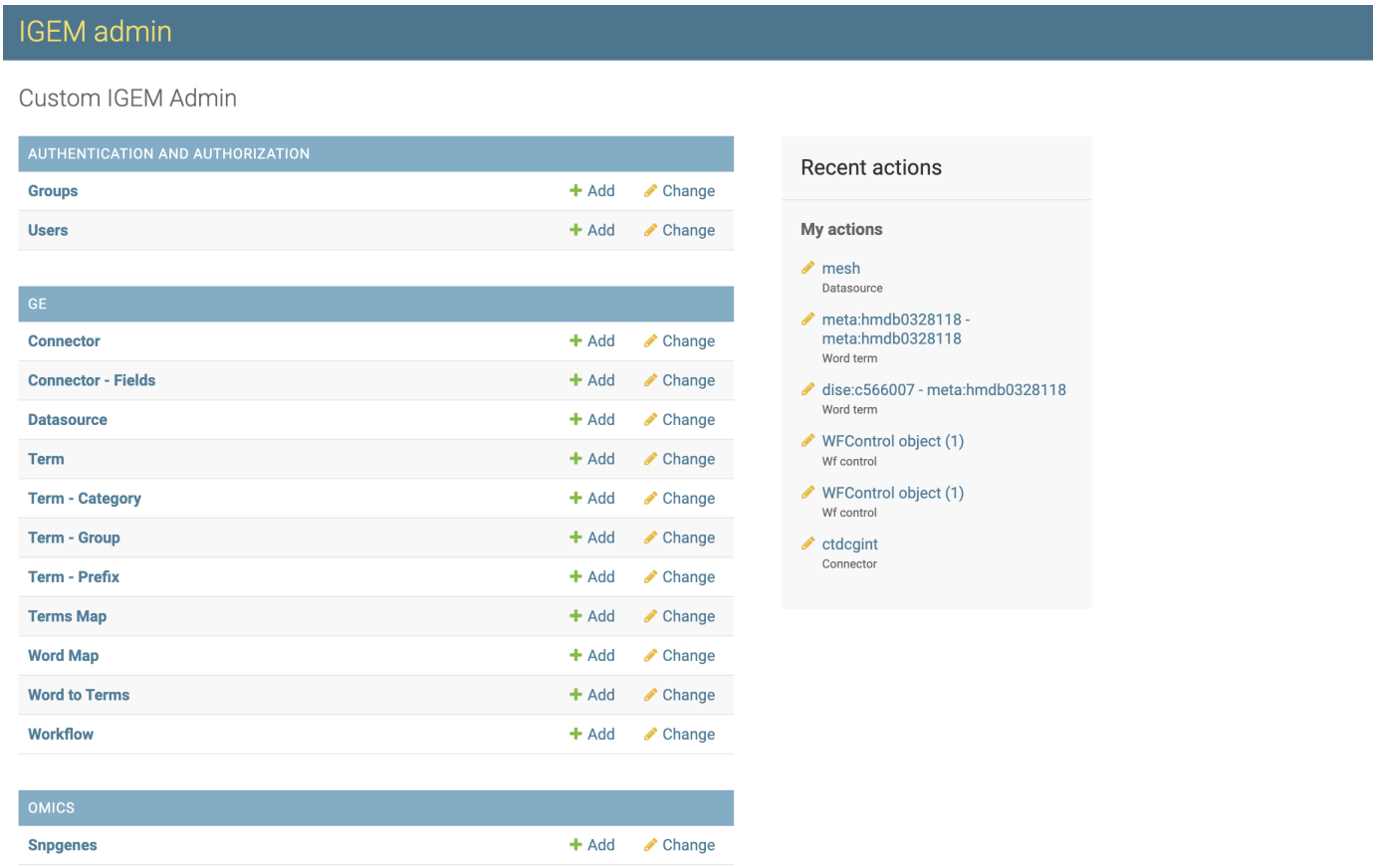
```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

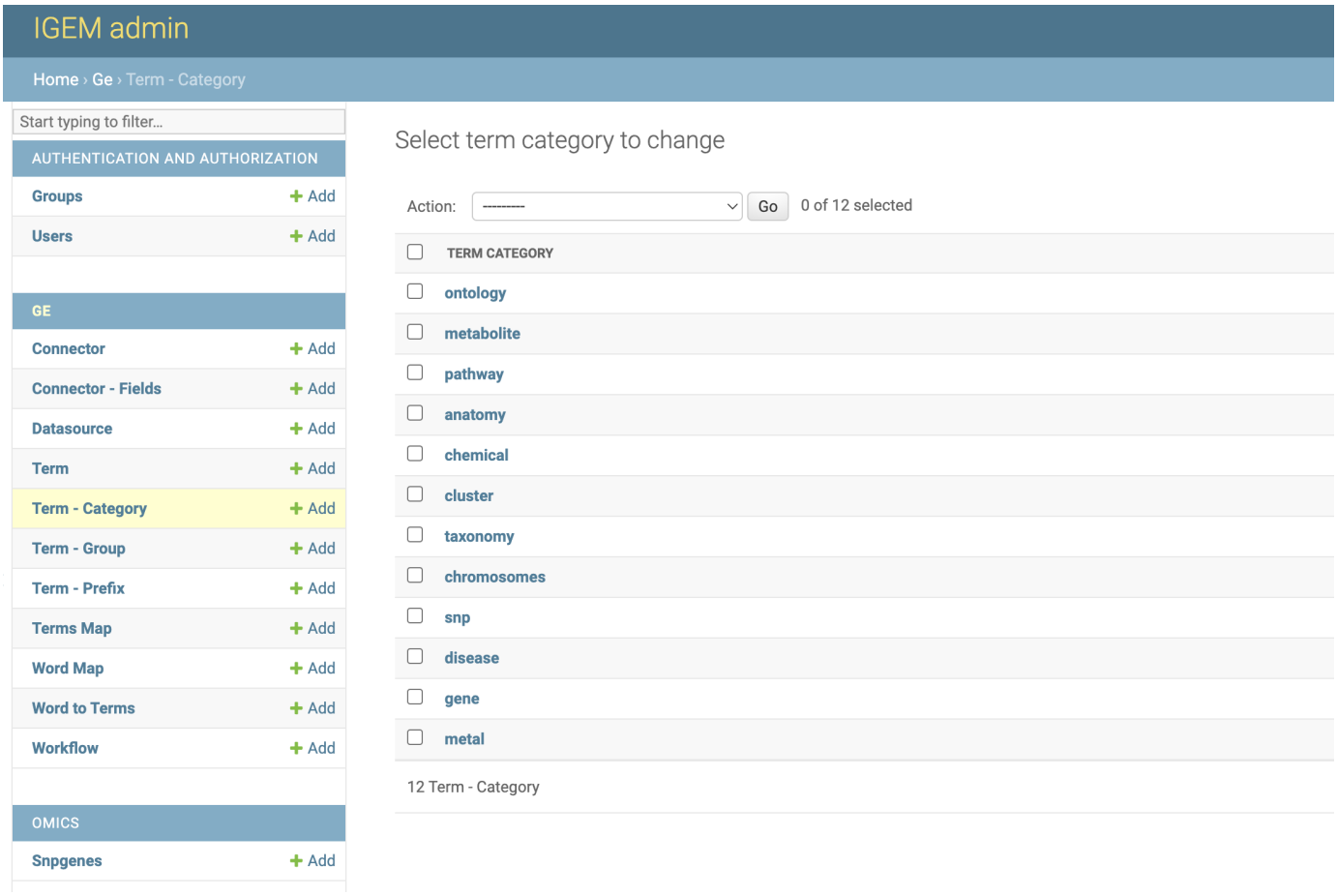
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

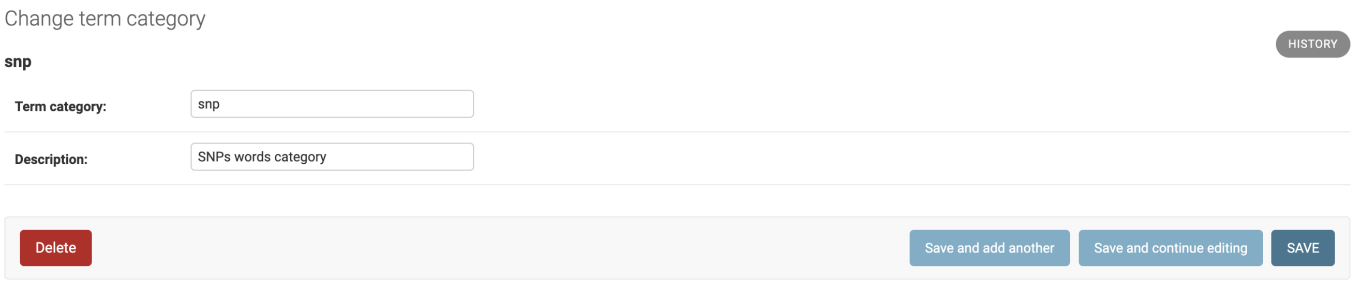
After user authentication and on the initial administration screen, select an option Term-Category.



On the Category screen, we will have options to consult, modify, add and eliminate Category.



On the first screen, we have a view of all available Category. To consult, click a desired Category.



On the next screen, we have all the Category fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank Category screen to add a new Category record.
- `Save and Continue editing`: Will save the changes and continue on the Category screen.
- `Save`: Will save the changes and return to the screen with the list of Category.

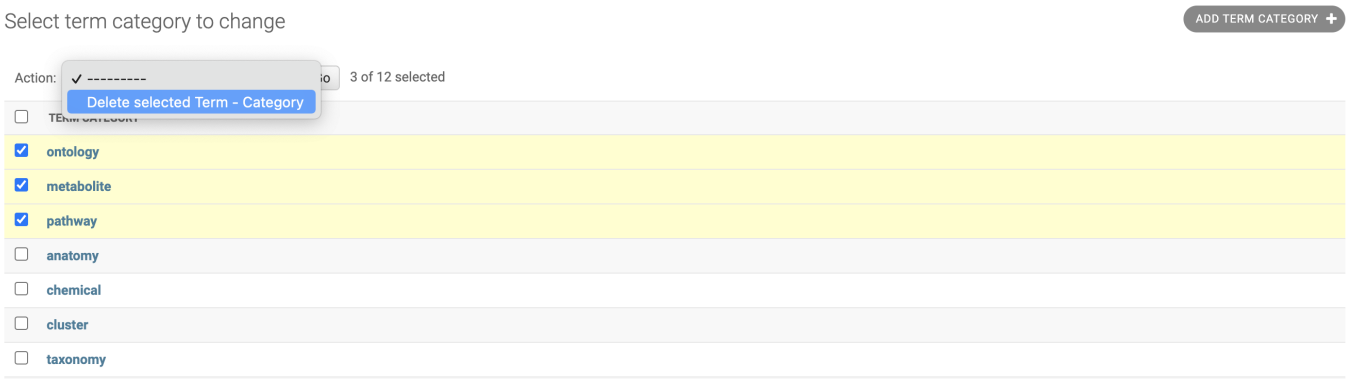
In the History button, we can consult all the modifications carried out in the Category, this function will be important to track modifications and audit the process.

The `DELETE` button will permanently delete the Category record.

Caution: when deleting a Category, the system will also delete all records dependent on that Category, which include KEYGE, and KEYLINKS

Deletion can also be performed en bloc. On the Category List screen, select all the Category you want to delete, choose the Delete Selected Keyge - Category action and click on the `GO` button.

Be careful, this elimination operation will be definitive for the Category and for all other records dependent on it, as already explained.



To add new Category, we will have three different ways:

- by the + `Add` button on the left sidebar.
- Through the `ADD Category +` button in the right field of the Category list.
- Via the `Save and add another` button located within a Category record.

Term

Term is the main component in GE.db and GE.filter and was created to specify a search term in external data sources. A Term can be assigned to a gene, a chromosome, an SNP, a disease, a chemical, an environmental factor, or any other term necessary to keep in the GE.db knowledge base.

A Term will have as attributes the Group and Category records to qualify and group, helping during searches, queries, and analysis of the GE.db knowledge base.

A Term inside GE.db can be kept as a code number, a prefix + code, or even a word, depending exclusively on the initial planning adopted. Thus allowing high flexibility in the use of the IGEM system.

The system has an interface for mapping external words to a Term, with this link having several external combinations for a single Term. The system does not allow mapping the same external word to more than one Term, a process necessary to guarantee the integrity of the knowledge base.

As described in the introduction, the purpose of GE.db will be to search an external record for all Terms found, correlate these Terms and maintain a frequency and origin, allowing, like GE.filter, to perform searches for combinations between Term in different external data sources quickly and easily.

The Term data will be stored in the ge_keyge table of the IGEM DB defined in the initial parameters. The available fields are:

- *ID*: GE.db internal key
- *Term*: Abbreviated name of the Term
- *Description*: Description for identifying and consulting the Term
- *Category_id*: foreign_key from ge_category
- *Group_id*: foreign_key from ge_group

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ python manage.py db --load_data "table='term, path='{your_path}/term.csv'"
```

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes TermMap amd WordMap information

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out Term management activities.

Activate the IGEM web service if you have not already done so. Go to the IGEM folder and type the command line:

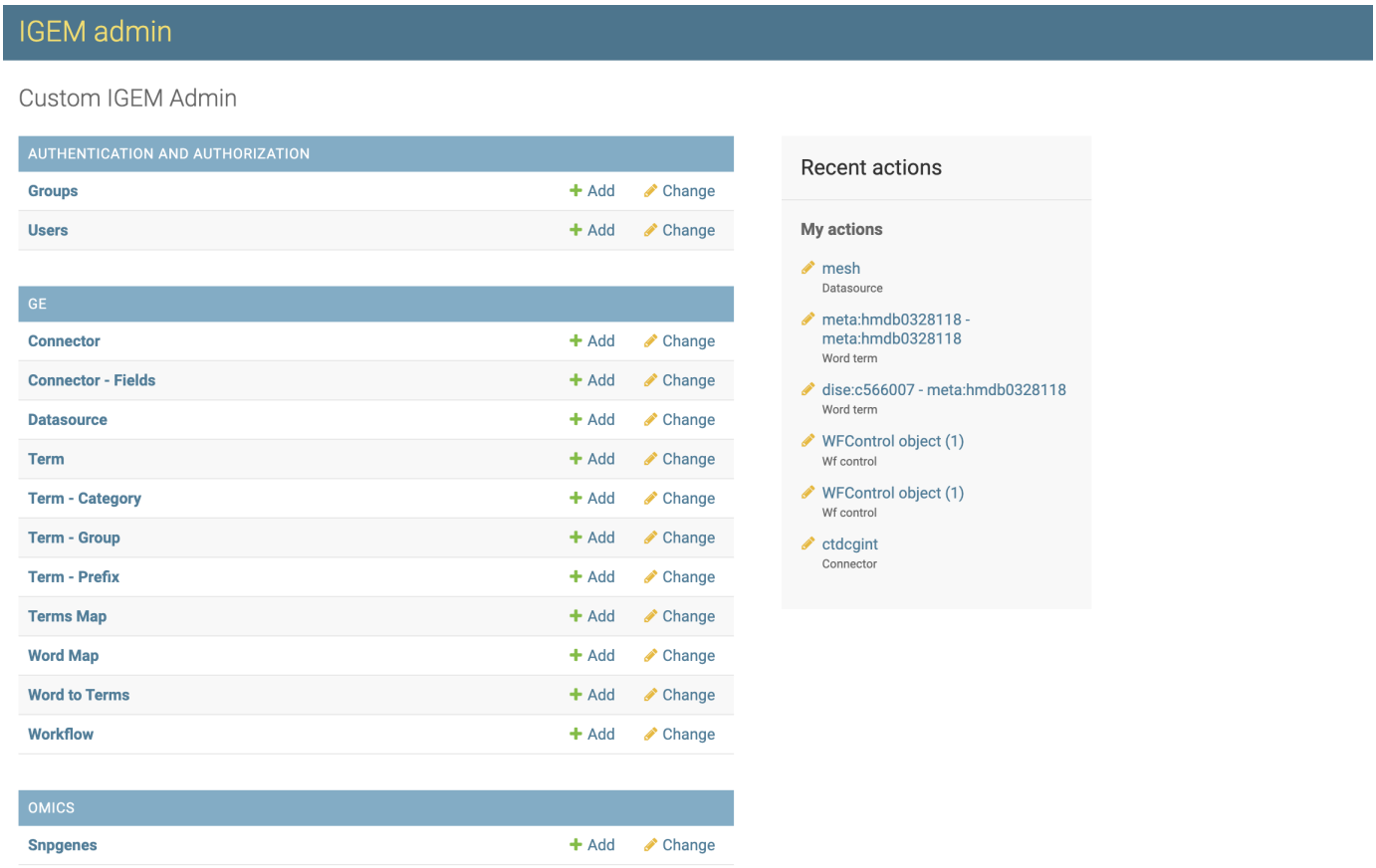
```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

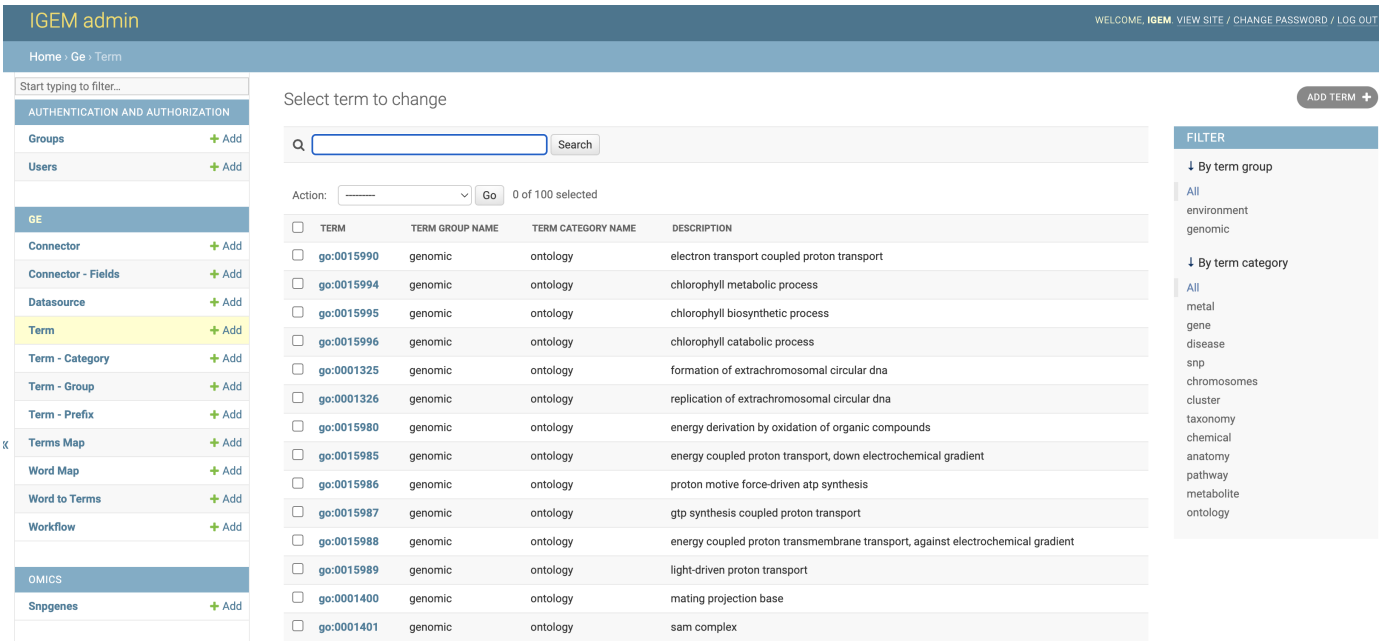
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

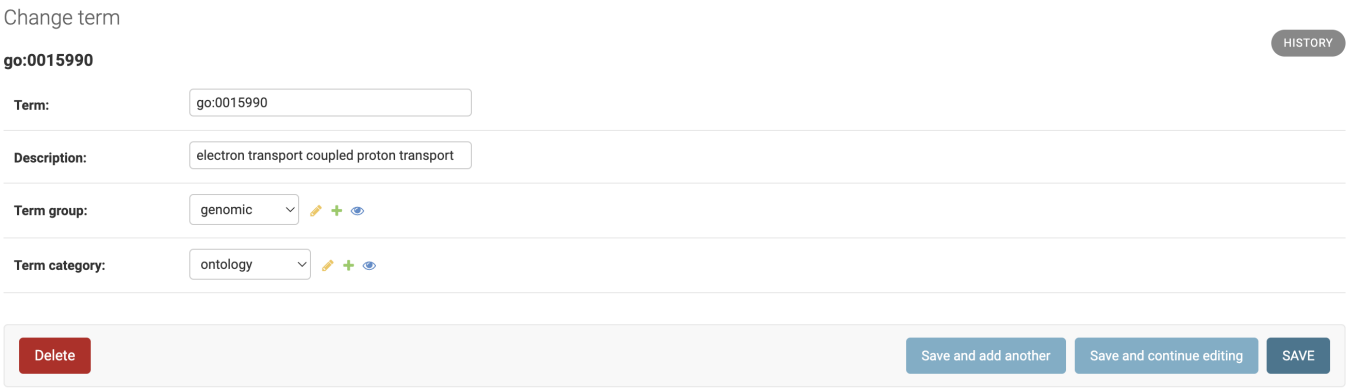
After user authentication and on the initial administration screen, select an option Database.



On the Database screen, we will have options to consult, modify, add and eliminate Term.



On the first screen, we have a view of all available Term. To consult, click a desired Term.



On the next screen, we have all the Term fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank Term screen to add a new Term record.
- `Save and Continue editing`: Will save the changes and continue on the Term screen.
- `Save`: Will save the changes and return to the screen with the list of Term.

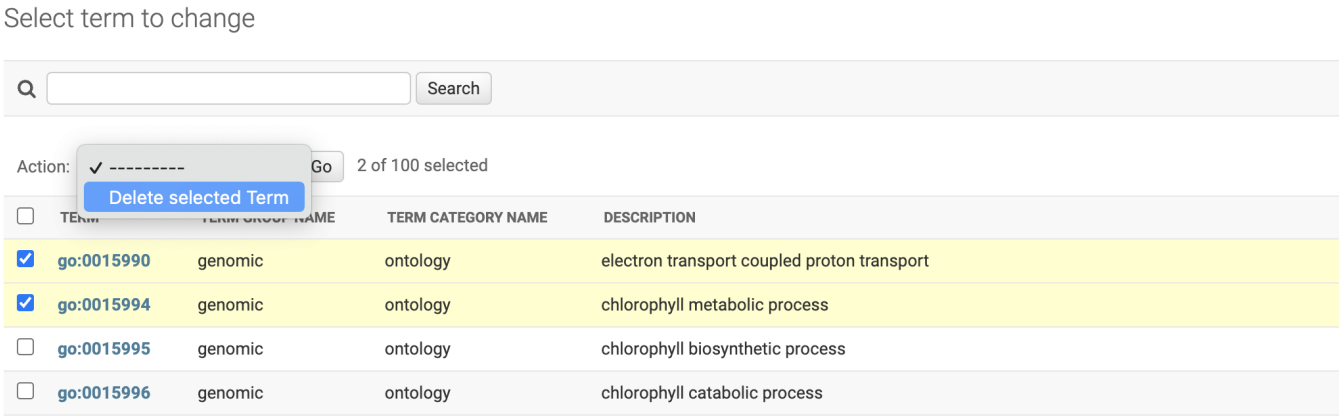
In the History button, we can consult all the modifications carried out in the Term, this function will be important to track modifications and audit the process.

The DELETE button will permanently delete the Term record.

Caution: when deleting a Term, the system will also delete all records dependent on that Term, which include KEYWORDs, and KEYLINKs

Deletion can also be performed en bloc. On the Term List screen, select all the Term you want to delete, choose the Delete Selected Term action and click on the GO button.

Be careful, this elimination operation will be definitive for the Term and for all other records dependent on it, as already explained.



To add new Term, we will have three different ways:

- by the + Add button on the left sidebar.
- Through the `ADD Term +` button in the right field of the Term list.
- Via the `Save and add another` button located within a Term record.

For the Term, we will have two filter locations:

- First located at the top of the Term List screen where we can search broadly.
- Second on the right sidebar, being able to select by Category and Group of Term.

Prefix

Prefixes play a fundamental role in the logic of the IGEM system for the correct identification of ref:*Term*.

Prefixes are assigned as input structure columns by Connector. Necessary due to identification only by the code of terms mapped to Term and that however conflict with other categories of Term.

For the correct identification of the Term, the system will add a prefix to the code located in the source Connector before MAPREDUCE processing.

Keeping without a record in none, it is important that the record is used during the IGEM ETL where additional cases from prefix to source code do not occur.

The Prefix data will be stored in the ge_prefixopc table of the IGEM DB defined in the initial parameters. The available fields are:

- *pre_value*: prefix name

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ python manage.py db --load_data "table='term, path='{your_path}/term.csv'"
```

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Connector columns rules.

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out GROUP management activities.

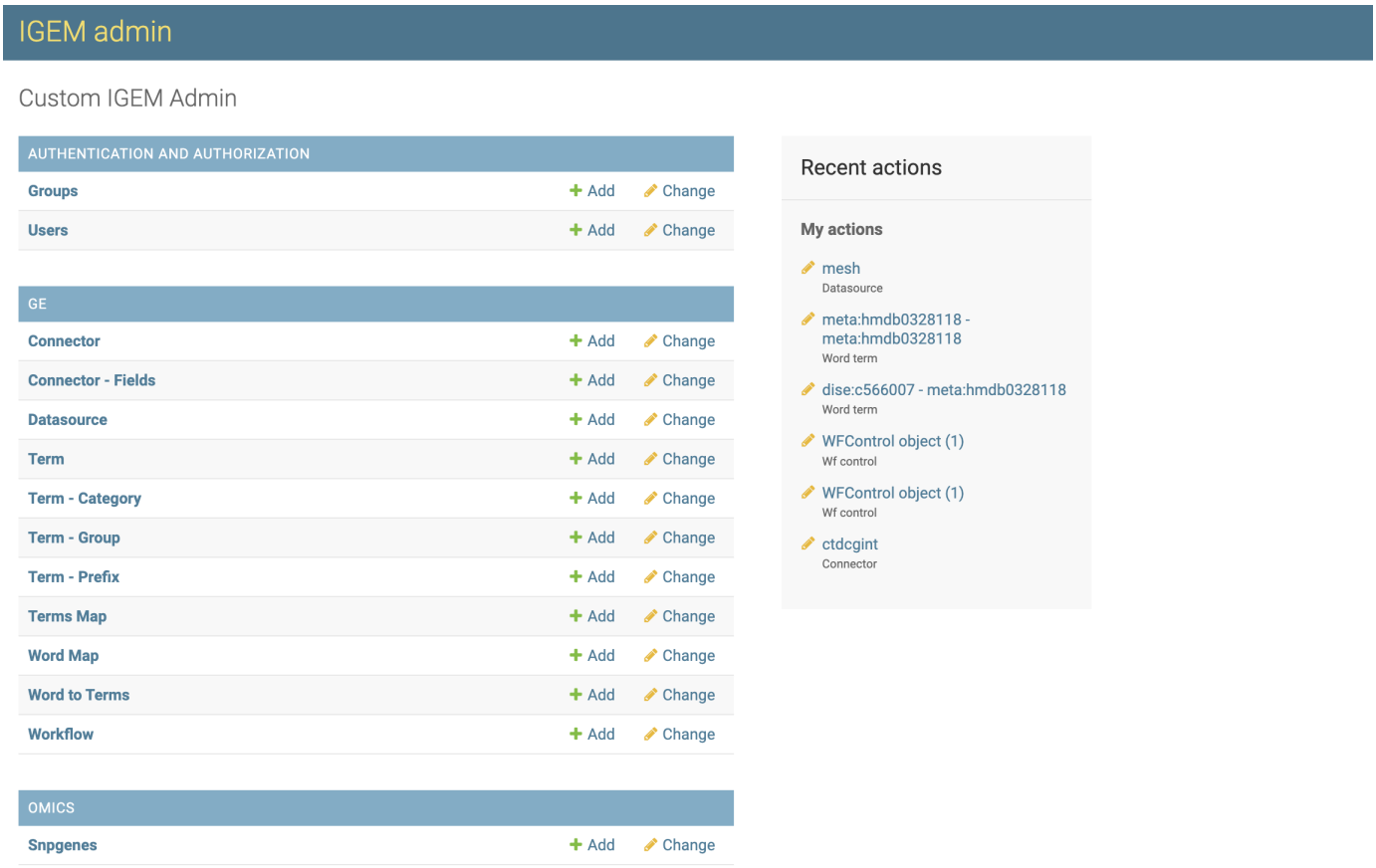
Activate the IGEM web service if you have not already done so. Go to the IGEM folder and type the command line:

```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
  March 24, 2023 - 12:56:26
  Django version 4.1.5, using settings 'src.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
If it returns a port error, you can specify a different port::
```

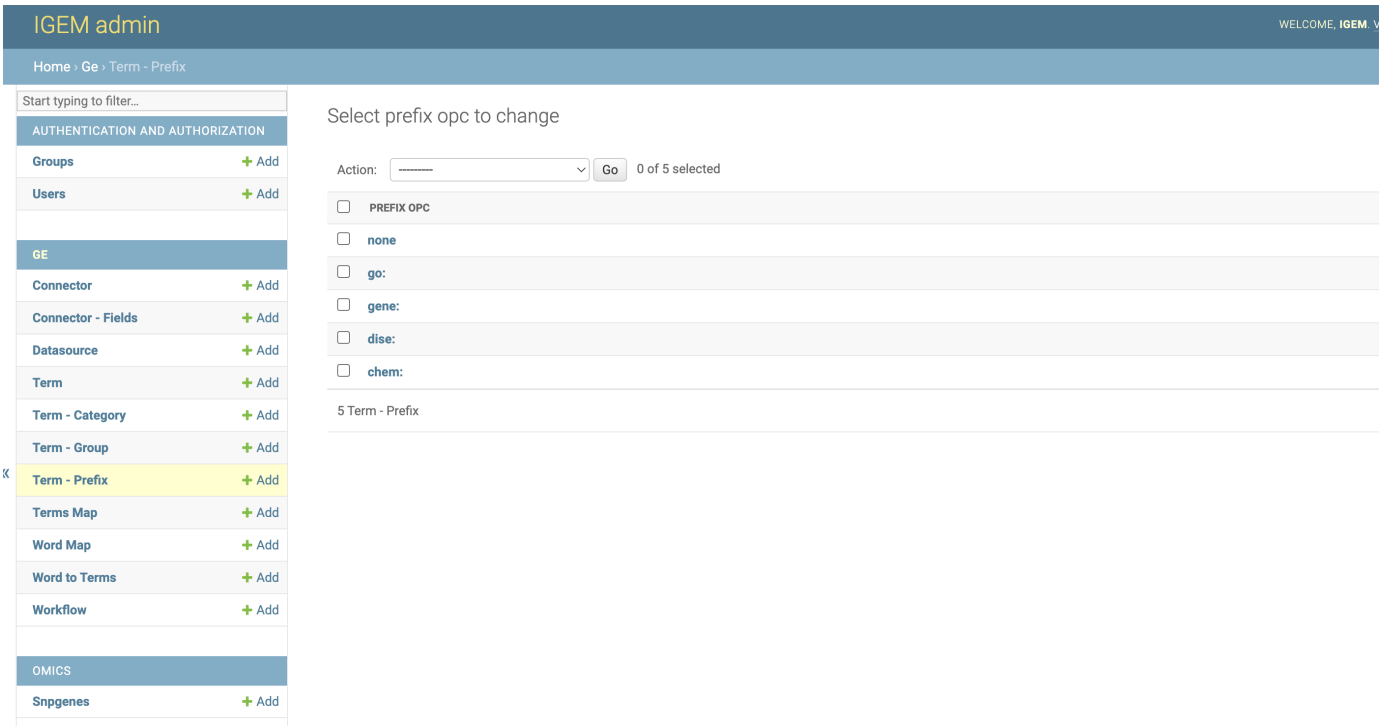
$ python manage.py runserver 8080

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

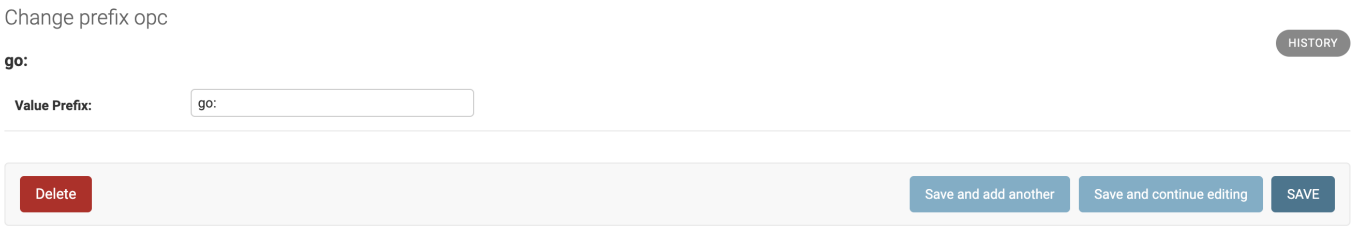
After user authentication and on the initial administration screen, select an option Keyge-Prefix.



On the Prefix screen, we will have options to consult, modify, add and eliminate Prefix.



On the first screen, we have a view of all available Prefix. To consult, click a desired Prefix.



On the next screen, we have all the Prefix fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank Prefix screen to add a new Prefix record.
- `Save and Continue editing`: Will save the changes and continue on the Prefix screen.
- `Save`: Will save the changes and return to the screen with the list of Prefix.

In the History button, we can consult all the modifications carried out in the Prefix, this function will be important to track modifications and audit the process.

The `DELETE` button will permanently delete the Prefix record.

Caution: when deleting a Prefix, the system will also delete all records dependent on that Prefix, which include Connector Columns Rules

Deletion can also be performed en bloc. On the Prefix List screen, select all the Prefix you want to delete, choose the Delete Selected Keyge - Prefix action and click on the `GO` button.

Be careful, this elimination operation will be definitive for the Prefix and for all other records dependent on it, as already explained.

To add new Prefix, we will have three different ways:

- by the `+ Add` button on the left sidebar.
- Through the `ADD Prefix OPC +` button in the right field of the Prefix list.
- Via the `Save and add another` button located within a Prefix record.

Word to Terms

The WordTerm data will be stored in the ge_WordTerm table of the IGEM DB defined in the initial parameters. The available fields are:

- *word*: The word or set of words that convert to Term (unique)
- *term_id*: foreign_key to ge_Term that link word with one Term
- *commute*: Flag used to convert. If it is the same criterion between Term and WORD, disable this flag to reduce memory consumption during the ETL process.
- *status*: Flag to activate the relationship

The inclusion of new data can be performed via the process `db` . On the command line:

```
$ python manage.py db --load_data "table='term', path='{your_path}/term.csv'"
```

Other commands and functions for manipulating master data can be found in the database management tab.

Web Interface

Through IGEM's friendly web interface, it will be possible to carry out Term management activities.

Activate the IGEM web service if you have not already done so. Go to the IGEM folder and type the command line:

```
$ python manage.py runserver

>>> Watching for file changes with StatReloader
  Performing system checks...
  System check identified no issues (0 silenced).
```

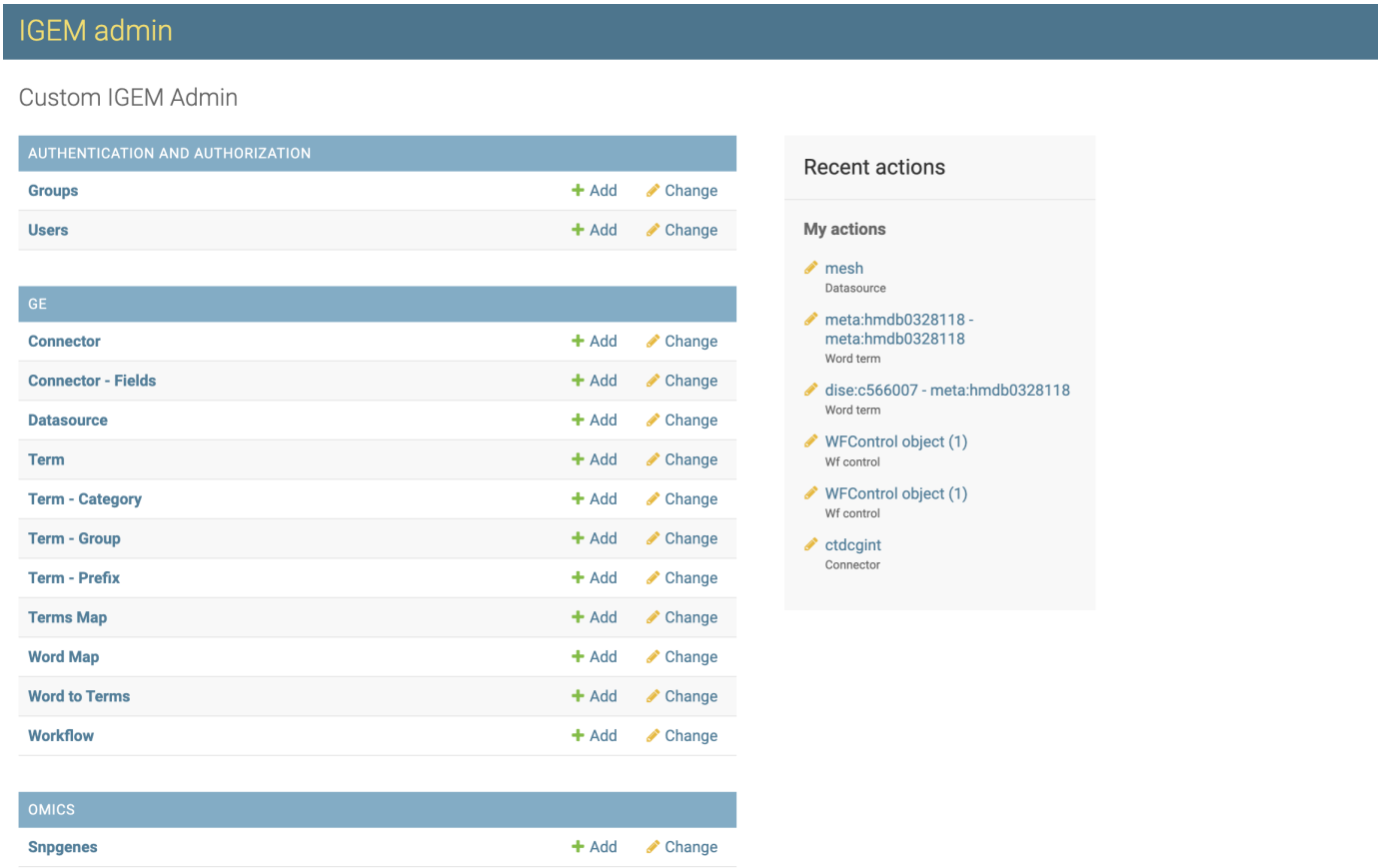
```
March 24, 2023 - 12:56:26
Django version 4.1.5, using settings 'src.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

If it returns a port error, you can specify a different port:

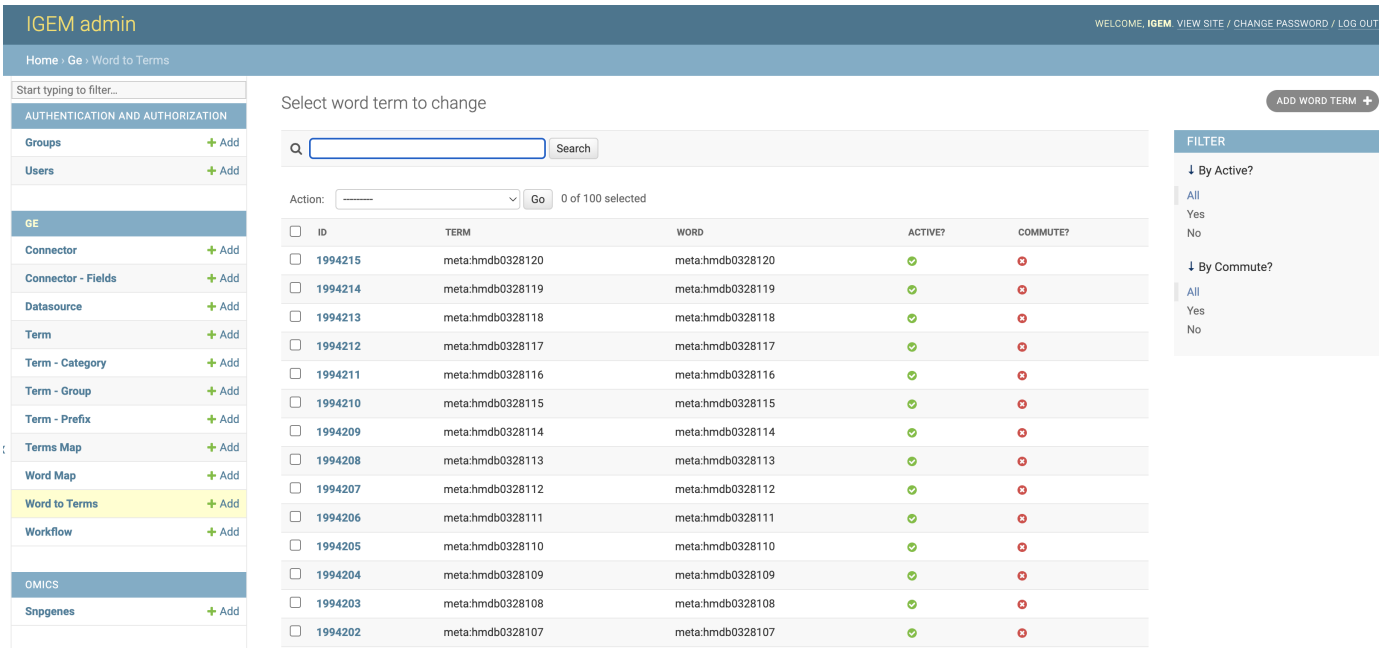
```
$ python manage.py runserver 8080
```

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

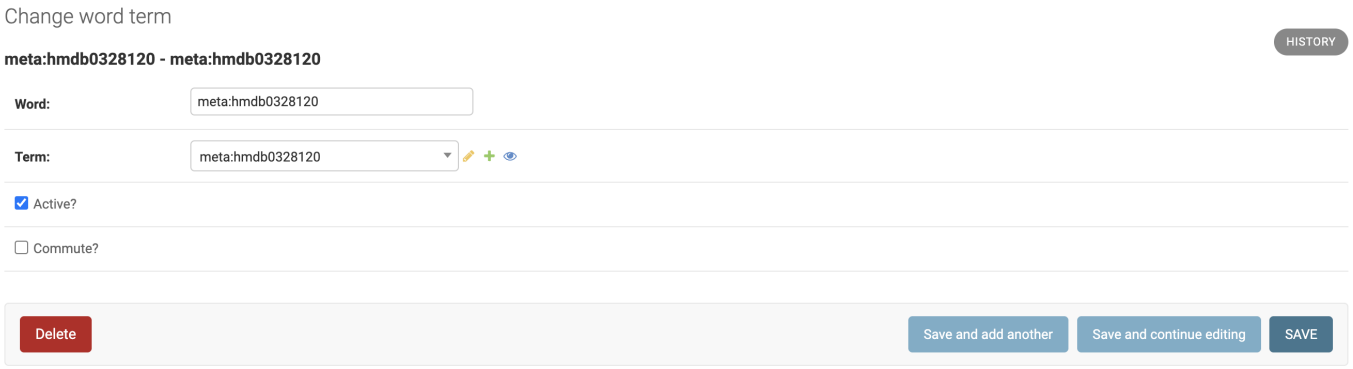
After user authentication and on the initial administration screen, select an option Database.



On the Database screen, we will have options to consult, modify, add and eliminate WordTerm.



On the first screen, we have a view of all available WordTerm. To consult, click a desired WordTerm.



On the next screen, we have all the WordTerm fields open for modifications. To modify, change the desired information and select one of the three button options:

- `Save and add another`: Will save the changes and open a blank WordTerm screen to add a new WordTerm record.
- `Save and Continue editing`: Will save the changes and continue on the WordTerm screen.
- `Save`: Will save the changes and return to the screen with the list of WordTerm.

In the History button, we can consult all the modifications carried out in the WordTerm, this function will be important to track modifications and audit the process.

The `DELETE` button will permanently delete the WordTerm record.

Caution: when deleting a WordTerm, the system will also delete all records dependent on that WordTerm, which include WordTerms, and KEYLINKs

Deletion can also be performed en bloc. On the WordTerm List screen, select all the WordTerm you want to delete, choose the Delete Selected WordTerm action and click on the `GO` button.

Be careful, this elimination operation will be definitive for the WordTerm and for all other records dependent on it, as already explained.

To add new WordTerm, we will have three different ways:

- by the + Add button on the left sidebar.
- Through the `ADD WordTerm +` button in the right field of the WordTerm list.
- Via the `Save and add another` button located within a WordTerm record.

For the WordTerm, we will have two filter locations:

- First located at the top of the WordTerm List screen where we can search broadly.
- Second on the right sidebar, being able to select by Actives status and Commute status.

Database Management

With the sql process, it will be possible to carry out data extraction operations, data loading, deletion, and cleaning of IGEM tables

The available tables are:

- datasource
- connector
- term_group
- term_category
- term
- ds_column
- prefix
- wordterm
- termmap
- wordmap

Python function

get_data

The get_data() function allows extracting data from the GE database and loading this data into a Pandas DataFrame structure or CSV File.

It has an intelligent filter mechanism that allow you to perform data selections simply through a conversion layer of function arguments and SQL syntax. This allows the same input arguments regardless of implemented database management system.

Parameters:

Only the table parameter will be mandatory, the others being optional, and will model the data output. In the case of only informing the table, the function will return a DataFrame with all the columns and values of the table.

- **table: str**

  datasource, connector, ds_column, term_group, term_category, term, prefix, wordterm, termmap, wordmap

- **path: str**

  With this parameter, the function will save the selected data in a file in the directory informed as the parameter argument. In this scenario, data will not be returned in the form of a Dataframe; only a Boolean value will be returned, informing whether the file was generated or not

- **columns: list[“str”]**

  Columns that will be selected for output. They must be informed with the same name as the database. It is possible to load other data from other tables as long as it correlate. For example, suppose the table only has the term field and not the category field. In that case, you can inform as an argument: “term_id__term_category_id__category”, the system selected the ID of the term, consulted the ID of the category in the Term table, and went to the Category table to choose the category

- **columns_out: list[“str”]**

If you want to rename the header of the output fields to more familiar names, you can use this parameter, passing the desired names in the same sequential sequence in the parameter columns

- **datasource: Dict{“str”:list[”str”]}**

  Filter argument. It is used to filter datasource, with the dictionary key being the selection argument and the dictionary value being the datasources selected as the filter. Without this parameter, the function will return all datasources

- **connector: Dict{“str”:list[”str”]}**

  Filter argument. It uses the same logic as the datasource, but applied to the connector field

- **word: Dict{“str”:list[”str”]}**

  Filter argument. It uses the same logic as the datasource, but applied to the word field

- **term: Dict{“str”:list[”str”]}**

  Filter argument. It uses the same logic as the datasource, but applied to the term field

- **term_category: Dict{“str”:list[”str”]}**

  Filter argument. It uses the same logic as the datasource, but applied to the term_categorty field

- **term_group: Dict{“str”:list[”str”]}**

  Filter argument. It uses the same logic as the datasource, but applied to the term_group field

Return:

Pandas Dataframe or Boolean (If the parameter path is informed, the function will generate the file; if successful, it will return the TRUE. Otherwise, it will return FALSE)

Examples:

```
>>> from igem.server import sql
>>> sql.get_data(
      table=”datasource”,
      datasource={“datasource__in”: [“ds_01”,”ds_02”]},
      columns=[“id”,”datasource”],
      columns_out=[“Datasource ID”, “Datasource Name”],
      path=”{your_path}/datasource.csv”
      )

>>> df = sql.get_data(
      table="connector",
      connector={"connector__start": ["conn_ds"]},
      datasource={"datasource_id__datasource__in": ["ds_01"]},
      columns=["connector", "status"]
      )

>>> x = sql.get_data(
      table="termmap",
      term={"term_id__term": "chem:c112297"},
      path="{your_path},
      )
   If x:
      print("file created")
```

load_data

Loads data from a CSV file into the IGEM database. This process does not update existing data, it only inserts new records.

Parameters:

- **table: str**

  datasource, connector, ds_column, term_group, term_category, term, prefix, wordterm, termmap, wordmap

- **path: str**

  full path and file name to load

Layout of data file:

- **Datasource:**

  (datasource, description, category, website)

- **Connector:**

  (connector, datasource, description, update_ds, source_path, source_web, source_compact, source_file_name, source_file_format, source_file_sep, source_file_skiprow, target_file_name, target_file_format)

- **Ds_column:**

  (connector, status, column_number, column_name, pre_value, single_word)

- **Term_group:**

  (term_group, description)

- **Term_category:**

  (term_category, description)

- **Term:**

  (term, category, group, description)

- **Prefix:**

  (pre_value)

- **Wordterm:**

  (term, word, status, commute)

- **Termmap:**

  (ckey, connector, term_1, term_2, qtd_links)

- **Wordmap:**

  (cword, datasource, connector, term_1, term_2, word_1, word_2, qtd_links)

We can generate an example file with the get_data() function and manipulate and load it with the new data.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

```
>>> from igem.server import sql
>>> sql.load_data(
      table="datasource”
      path=”{your_path}/datasource.csv”
      )
```

delete_data

Allows deleting a record from the given table. The deletion will be carried out in all records related to the informed parameter. For example, if we delete a datasource, the connectors, ds_columns, and termmap associated with the datasource will be deleted.

Parameters:

Only the table parameter will always be requested, the others will depend on the selected table, functioning as a record that will be eliminated.

- table: str

(datasource, connector, ds_column, term_group, term_category, term, prefix, wordterm, termmap, wordmap, workflow) - datasource: Dict{“str”:list[”str”]} - connector: Dict{“str”:list[”str”]} - word: Dict{“str”:list[”str”]} - term: Dict{“str”:list[”str”]} - term_category: Dict{“str”:list[”str”]} - term_group: Dict{“str”:list[”str”]} - prefix: Dict{“str”:list[”str”]}

(Filter argument. It is used to filter the field, with the dictionary key being the selection argument and the dictionary value being the field selected as the filter. Without this parameter, the function will return all values of the field.)

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

```
>>> from igem.server import sql
>>> sql.delete_data(
      table='datasource',
      datasource={'datasource__in': [ds_01]}
      )
```

truncate_table

will delete all records from a table, never use this function, with excess if the need is to restart a new instance of the database, free up log table space or in test environments.

Parameters:

- **table: str**

  (datasource, connector, dst, term_group, term_category, term, prefix, wordterm, termmap, wordmap, workflow, logs)

If inform table=”all”, the function will truncate all table on GE database. The other tables of the IGEM system will be maintained.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

```
>>> from igem.server import sql
>>> sql.truncate_table(
         table='datasource'
         )
```

backup

Backup the database with the internal keys. It can be performed at once for all GE.sql tables

Parameters:

- **table: str**

  (datasource, connector, dst, term_group, term_category, term, prefix, wordterm, termmap, wordmap, workflow, logs)

- **path_out: str**

  Folder path to store the generated backup files

If inform table=”all”, the function will backup all table on GE database.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

```
>>> import igem
>>> igem.server.sql.backup(
         table="",
         path_out="/root/back")
```

restore

Restore the database with the internal keys. It can be performed at once for all GE.sql tables

Parameters:

- **table: str**

  (datasource, connector, dst, term_group, term_category, term, prefix, wordterm, termmap, wordmap, workflow, logs)

- **path_out: str**

  Folder path to store the generated backup files

If inform table=”all”, the function will restore all table on GE database.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

```
>>> import igem
>>> igem.server.sql.restore(
        table="",
        path_out="/root/back")
```

Command Line

Within the parameters, inform the same ones used for the functions, as well as the arguments, example:

```
$ $ python manage.py sql --get_data 'table="datasource", datasource={“datasource__in”: [“ds_
```

Get data:

```
$ python manage.py sql --get_data {parameters}
```

Load data:

```
$ python manage.py sql --load_data {parameters}
```

Delete data:

```
$ python manage.py sql --delete_data {parameters}
```

Delete all table:

```
$ python manage.py sql --truncate_table {parameters}
```

Backup (get data with internal ID):

```
$ python manage.py sql --backup {parameters}
```

Restore (load data with internal ID):

```
$ python manage.py sql --restore {parameters}
```

ETL

The ETL process is responsible for fetching data from external sources, transforming it into a compatible standard, searching for term relationships, and writing the data to GE.db. It consists of five distinct phases to efficiently manage resources and ensure successful execution:

Collect: This phase involves gathering data from external sources.

Prepare: In this phase, the collected data is processed and prepared for further transformation and loading.

Map: The data is mapped to relevant terms and categories, establishing relationships between them.

Reduce: Unnecessary or redundant data is filtered out, ensuring that only relevant information is retained.

Workflow: This phase coordinates the entire ETL process, orchestrating the execution of the preceding phases.

Each phase is explained in detail in the respective files:

Collect

The “Collect” process is responsible for selecting active connectors and checking if new versions of data are available. It performs the following tasks:

Connector Selection: The process selects active connectors to fetch data from various sources.

Data Extraction: If a new version of the data is available, the process extracts the latest data.

File Handling: If necessary, the extracted file is uncompressed and stored in the Persists Storage Area (PSA).

Logs and Version Controls: The process updates logs and version controls to track the execution and status of each connector.

Currently, the execution version of the steps in the web interface is still under development.

The process is executed through the command line using the following script::

```
$ python manage.py etl –collect {all or connector}
```

If the “all” option is used, the process collects data for all active connectors in the master data table.

If a specific connector is provided, only that connector’s data will be collected.

Prepare

This second phase of the process aims to transform the original data, thus reducing the need for computational resources in the subsequent steps. Based on the briefly configured connector parameters, in this phase, we will have:

- Deleting header lines
- Deleting unnecessary columns
- Transforming ID Columns with Suffix Identifiers
- Replacement the terms
- Deletion of the original file

The output will be a new temporary file for consumption in the next phase::

```
# python manage.py etl –prepare {all or connector}
```

It will start the data preparation phase for all connectors or just one specified. Essential to have the file in PSA. Otherwise, the system will display a warning:

The reset option will reset the control for all or a specific connector in the preparation phase and the two later ones.

Map

The map will process each line of the file and combine all found words. The result will be recorded in the WordMap table::

```
# python manage.py etl –map {all or connector}
```

It will start the data term switching phase for all connectors or just one specified. Essential to have the file in PSA. Otherwise, the system will display a warning:

The reset option will reset the control for all or a specific connector to the switching phase and the next phase

Reduce

Last step of the process. It has a mechanism to find Term (terms) per line called Mapper and then activate the Reducer subprocess that will count the number of links found in the connector. After all processing, the result will be recorded in the Keylinks table. It is important to note that the new data will fully replace the previous data in the processed connector.::

```
$ python manage.py etl –reduce {all or connector}
```

Essential to have the file in PSA. Otherwise, the system will display a warning. It will start the MapReduce phase of data terms for all Connector or just a specific one. In this phase, there is a large consumption of memory and processing, so it will be essential to allocate resources compatible with the size of the processed data.:

Reset option will Press the control to all or a specific connector in the current phase.

In all commands with run argument, possible multiprocessing, and control for file chunks. However, it will be necessary or necessary between the size of the extracted files and the resources allocated, such as memory and the amount of proposed balancing.

Workflow

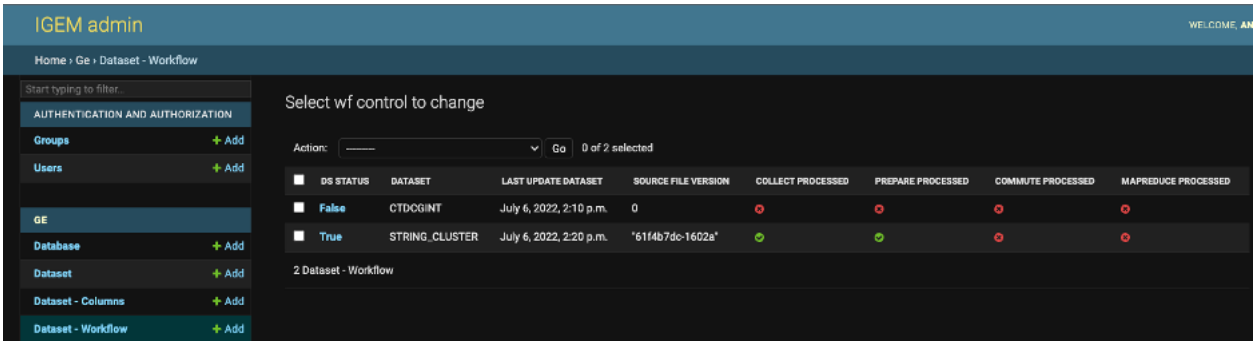
In addition to the reset commands shown above to control the process workflow, the system also has a web interface where the user can consult and manage the flow and status of the phases.

Browse http://127.0.0.1:8000/admin/ge/wfcontrol/ or select the Dataset – Workflow option in the GE application In the first one, Connectors that have already been started will be started, with the following references:

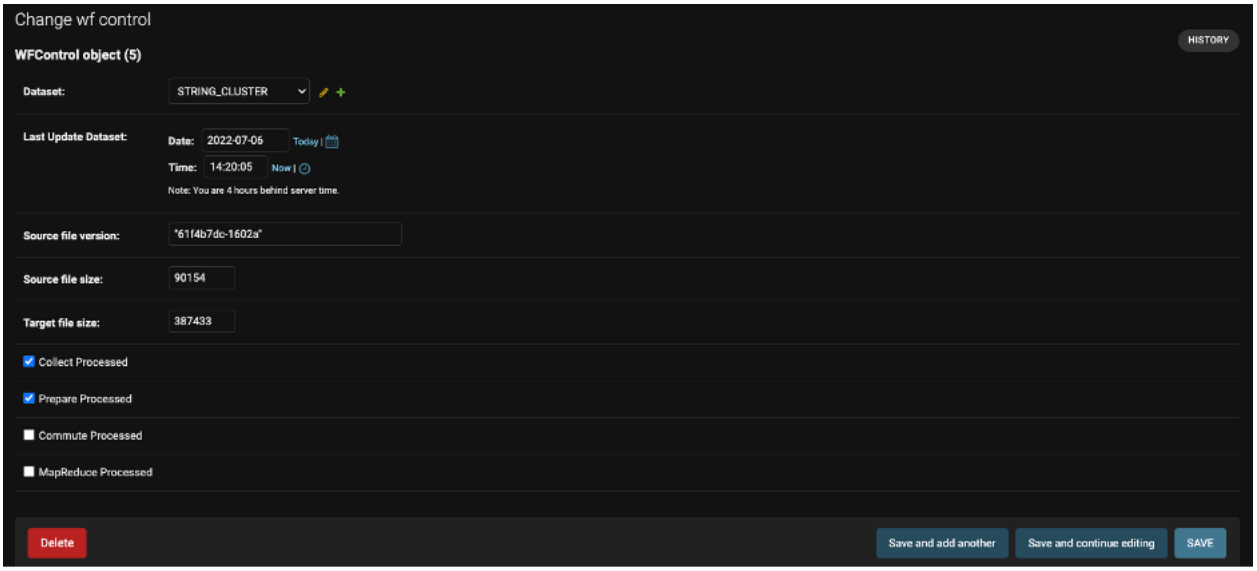
- DS STATUS: informs whether the Connector is active or not for processing the 4 phases
- Connector: Abbreviation for Connector
- Last Update Dataset: Date of the previous data update.
- Source file version: Version of the final processed file.

The following four columns display the statuses still of the processes by phase. The green symbol indicates the status-completed successfully and the group not processed.

Data were only available in the GE.be database after all phases had been successfully executed.



When selecting a Connector, it will be a details screen allowing the specific opening by field.



Important: it will not be necessary to include new Connectors in the Workflow monitor. The system will automatically create a new control after the first load of data from the Connector. If one of the workflow records is deleted, it will also be completed after the next data load.

To ensure a successful ETL process, proper parameterization and accurate master data entries are crucial. These files will guide you through each phase, helping you understand and execute the ETL process effectively.

EPC Application

The EPC (Extend Process Call) module in the IGEM software provides a comprehensive set of functionalities that enable users to create an end-to-end pipeline for data analysis. This module offers various tools and functions to load external datasets, perform data description, and modify the data to adapt it to different types of analyses such as EWAS, Association Study, and ExE Pairwise analysis. Here is an overview of the key functionalities offered by the EPC module:

Loading External Datasets

Allows users to seamlessly load external datasets into the script. It supports loading data from CSV and TSV files. This functionality enables researchers to integrate their data with the IGEM ecosystem for further analysis.

Data Description

Users can obtain a comprehensive description of their datasets. This includes calculating correlations between variables, generating frequency tables for categorical variables, determining data types of variables, calculating the percentage of missing values, computing skewness of variables, and generating summary statistics for variables. These descriptive statistics provide valuable insights into the dataset and help researchers understand its characteristics.

Data Modification

Offers a wide range of data modification functions to prepare the dataset for specific analyses. Users can categorize variables based on defined criteria, filter columns based on specific conditions, convert variables to binary or categorical format, merge observations or variables based on specified conditions, move variables within the dataset, record specific values for variables, remove outliers, filter rows with incomplete observations, and perform transformations on variables. These data modification functions enable researchers to tailor the dataset to their analysis requirements.

Data Analysis

The EPC module includes functionalities specifically designed for conducting:

- Environment-Wide Association Studies (EWAS). Researchers can leverage these functions to analyze the association between epigenetic modifications and phenotypic traits. The EPC module provides dedicated tools to perform statistical tests, correct p-values, and generate graphical representations such as Manhattan plots.
- Association Study by providing tools to analyze the relationships between variables in the dataset. Users can perform association tests and explore the strength and significance of associations between variables. This functionality is particularly useful for identifying potential relationships and dependencies within the data.
- ExE (Exposure by Exposure) Pairwise analysis, allowing researchers to examine the pairwise relationships between exposures. By applying this analysis, users can identify potential interactions or dependencies between different exposures in the dataset.

Survey Design and Modeling

Users can define survey designs with specific sampling strategies and create survey models for analyzing survey data. These features cater to researchers working with survey datasets and provide specialized tools for accurate analysis.

Plot Functions

The EPC module provides various plot functions to visualize the data and gain deeper insights. These plot functions include:

- Distributions: Generate visual representations of variable distributions, such as histograms and kernel density plots. These plots help researchers understand the underlying distribution of variables in the dataset.
- Histograms: Create histograms to visualize the distribution of a single variable. This plot provides a visual summary of the frequency distribution of values in the dataset.
- Manhattan Plot: Generate a Manhattan plot, commonly used in genetic association studies, to visualize the genomic location of associations. This plot displays the significance of associations along the genome.
- Manhattan Plot with Bonferroni Correction: Similar to the Manhattan plot, this function incorporates Bonferroni correction to account for multiple hypothesis testing. It helps identify significant associations while controlling for the family-wise error rate.
- Manhattan Plot with False Discovery Rate (FDR): This function applies the False Discovery Rate (FDR) correction to the associations in the Manhattan plot. It allows researchers to control the expected proportion of false discoveries while identifying significant associations.
- Top Results Plot: Create a plot displaying the top results of an analysis, such as the most significant associations or the highest-ranked variables. This plot helps researchers focus on the most important findings in the data.

By utilizing the functionalities offered by the EPC module, users can create a streamlined and comprehensive pipeline for data analysis within the IGEM software. This module empowers researchers to load external datasets, describe the data, modify it to suit specific analyses, and perform advanced statistical tests and visualizations.

Load

Load data from different formats or sources.

In the above example, the *from_tsv* function loads data from a tab-separated file, while the *from_csv* function loads data from a comma-separated file. Both functions return a DataFrame, where the index column is used for merging. The examples demonstrate how to use these functions to load files and provide information about the number of observations and variables loaded.

Analyze

Describe

Functions that are used to gather information about some data.

ness for continuous variables:

```
import igem
skewness = igem.epc.describe.skewness(df)
skewness
```

Modify

Functions used to filter and/or change some data, always taking in one set of data and returning one set of data.

Plot

Functions that generate plots

histogram

Plot a histogram of the values in the given column.

Parameters

- *data*: pd.DataFrame The DataFrame containing data to be plotted
- *column*: str The name of the column that will be plotted

- *figsize*: Tuple[int, int], default (12, 5) The figure size of the resulting plot
- *title*: str or None, default None The title used for the plot
- *figure*: matplotlib Figure or None, default None Pass in an existing figure to plot to that instead of creating a new one (ignoring figsize)
- ***kwargs*: Other keyword arguments to pass to the histplot or catplot function of Seaborn

Examples

``` python import igem igem.epc.plot.histogram(

nhanes_discovery_cont, column="BMXBMI", title=x, bins=100

)

## *distributions*

Create a pdf containing histograms for each binary or categorical variable and one of several types of plots for each continuous variable.

### Parameters

- *data*: pd.DataFrame The DataFrame containing data to be plotted
- *filename*: str Name of the saved pdf file. The extension will be added automatically if it was not included.
- *continuous_kind*: str, default "count" What kind of plots to use for continuous data. Binary and Categorical variables will always be shown with histograms. One of {'count', 'box', 'violin', 'qq'}
- *nrows*: int, default 4 Number of rows per page
- *ncols*: int, default 3 Number of columns per page
- *quality*: str, default "medium" Adjusts the DPI of the plots (150, 300, or 1200)
- *variables*: List[str] or None Which variables to plot. If None, all variables are plotted.
- *sort*: bool, default True Whether or not to sort variable names

### Examples

``` python import igem igem.epc.plot.distributions(

df[['female', 'occupation', 'LBX074']], filename="test"

)

## *manhattan*

Create a Manhattan-like plot for a list of EWAS Results.

### Parameters

- *dfs*: Dict[str, pd.DataFrame] Dictionary of dataset names to pandas dataframes of ewas results (requires certain columns)
- *categories*: Dict[str, str] or None, default None A dictionary mapping each variable name to a category name for optional grouping
- *bonferroni*: float or None, default 0.05 Show a cutoff line at the p-value corresponding to a given bonferroni-corrected p-value
- *fdr*: float or None, default None Show a cutoff line at the p-value corresponding to a given false discovery rate (FDR)

- *num_labeled*: int, default 3 Label the top <num_labeled> results with the variable name
- *label_vars*: List[str] or None, default None Label the named variables (or pass None to skip labeling this way)
- *figsize*: Tuple[int, int], default (12, 6) The figure size of the resulting plot in inches
- *dpi*: int, default 300 The figure dots-per-inch
- *title*: str or None, default None The title used for the plot
- *figure*: matplotlib Figure or None, default

## Survey

### Survey Design Specification

`your_module_name.`**`SurveyDesignSpec`** (`survey_df`, `strata`=None, `cluster`=None, `nest`=False, `weights`=None, `fpc`=None, `single_cluster`='fail', `drop_unweighted`=False)

Holds parameters for building a statsmodels SurveyDesign object.

**Parameters:**

- **survey_df** (*pandas.DataFrame*) – A pandas DataFrame containing Cluster, Strata, and/or weights data.
- **strata** (*str, optional*) – The name of the strata variable in the survey_df, defaults to None.
- **cluster** (*str, optional*) – The name of the cluster variable in the survey_df, defaults to None.
- **nest** (*bool, optional*) – Whether or not the clusters are nested in the strata, defaults to False.
- **weights** (*str or dict, optional*) – The name of the weights variable in the survey_df, or a dictionary mapping variable names to weight names, defaults to None.
- **fpc** (*str, optional*) – The name of the variable in the survey_df that contains the finite population correction information, defaults to None.
- **single_cluster** (*str, optional*) – Setting controlling variance calculation in single-cluster (‘lonely psu’) strata. Valid options are ‘fail’, ‘adjust’, ‘average’, and ‘certainty’. Defaults to ‘fail’.
- **drop_unweighted** (*bool, optional*) – If True, drop observations that are missing a weight value. This may not be statistically sound. Otherwise, the result for variables with missing weights (when the variable is not missing) is NULL. Defaults to False.

**Returns:** A SurveyDesignSpec object.

**Return type:** SurveyDesignSpec

### Survey Model

`your_module_name.`**`SurveyModel`** ()

Creates a SurveyModel object used to fit a model to survey data.

**Returns:** True

**Return type:** bool

# Index