Biostar Central Documentation

Release latest

Oct 05, 2017

Contents

BioStar is a Python and Django based Q&A software licensed under the *MIT Open Source License*. It is a simple, generic, flexible and extensible Q&A framework.

The site has been developed by **scientists and for scientists**. It aims to address requirements and needs that scientific communities have.

Biostar is the software that runs several science oriented Q&A sites:

- Biostars Bioinformatics Q&A at: https://www.biostars.org
- Galaxy User support site: https://biostar.usegalaxy.org
- Metabolomics Q&A: http://www.metastars.org
- Neurostars: http://www.neurostars.org

CHAPTER 1

Features

The site has been developed by scientists for scientists. It aims to address specific needs that scientific communities have.

- Standard Q&A: post questions, answers, comments, user moderation, voting, badges, threaded discussions
- Email integration: import previous posts from mailing lists, reply to posts via email
- RSS Planet: feed aggregation from different sources
- External authentication: authenticate users with a different web service

CHAPTER 2

Support

The software is open source and free to use under the most permissible license.

The developers of the software are also available to provide commercial level support for deploying Biostar sites for entire organizations. Contact: admin@biostars.org

Requirements: *Python 2.7*

Official documentation is located at http://docs.biostars.org

CHAPTER 3

Quick Start

From the biostar source directory:

```
# Install the requirements.
pip install --upgrade -r conf/requirements/base.txt

# Load the environment variables.
source conf/defaults.env

# Initialize database, import test data, and run the site.
./biostar.sh init import index run
```

Visit **http://localhost:8080** to see the site loaded with default settings. Enjoy.

For more information see the documentation below:

Install

The sourcecode can be obtained via::

```
git clone https://github.com/ialbert/biostar-central.git
```

Getting started

Get the source and switch to the source directory. The recommended installation is via `virtualenv` and `pip`::

```
# Install the requirements.
pip install --upgrade -r conf/requirements/base.txt

# Initialize, import test data and run the site.
./biostar.sh init import run
```

Visit `http://localhost:8080` to see the site loaded with default settings.

The default admin is `1@lvh.me` password `1@lvh.me`. The default email handler will print to the console. You can reset the password for any user then copy paste the password reset url into the browser.

Run the manager on its own to see all the commands at your disposal::

```
./biostar.sh
```

To enable searching you must the content with::

```
./biostar.sh index
```

Blog Aggregation

Biostar has the ability to aggregate blog feeds and allow searching and linking to them. List the RSS feeds in a file then::

```
# Initialize with new feed urls (see example)
python manage.py planet --add biostar/apps/planet/example-feeds.txt

# Download all feeds (usually performed daily)
python manage.py planet --download

# Add one new blog entry for each feed the downloaded file (if there is any)
python manage.py planet --update 1
```

Sending Emails

By default Biostar can send email via the standard email facilities that Django provides see https://docs.djangoproject.com/en/dev/topics/email/

Biostar offers a few helper functions that allow emailing via Amazon SES::

```
# Amazon SES email settings.
EMAIL_USE_TLS = True
EMAIL_BACKEND = 'biostar.mailer.SSLEmailBackend'
```

Note: sending an email blocks the server thread! This means that the server process allocated to sending email will stop serving other users while the email is being sent. For low traffic sites this may not be a problem but for higher traffic sites the approach is not feasible.

To address that Biostar also implements a Celery based email backend that queues up and sends emails as separate worker processes, independently of the main server. Setting that up is very simple via the settings::

```
# Amazon SES email sent asynchronously.
EMAIL_USE_TLS = True
EMAIL_BACKEND = 'biostar.mailer.CeleryEmailBackend'
CELERY_EMAIL_BACKEND = 'biostar.mailer.SSLEmailBackend'
```

Receiving Emails

Biostar can be set up to receive emails and deposit them into threads. This allows users to use emails to post to Biostar.

To enable this functionality the site admins need to set up an email system that can, when a matching and address can perform a POST action to a predetermined URL. For example when delivering email via `postmaster` utility on linux the `etc/alias` file would need to contain::

```
reply: "| curl -F key='123' -F body='<-' https://www.mybiostar.org/local/email/
```

The above line will trigger a submit action every time that an email is received that matches the address words `reply`. For example: `reply@server.org`

Important: Biostar will send emails as `reply+1238429283+code@server.org`. The segment between the two + signs is unique to the user and post and are required for the post to be inserted in the correct location. The email server will have to properly interpret the + signs and route this email via the `reply@server.org` address. Now the default installations of `postmaster` already work this way, and it is an internal settings to `postmaster`. This pattern that routes the email must match the `EMAIL_REPLY_PATTERN` setting in Biostar.

The `key=123` parameter is just an additional measure that prevent someone flooding the email service. The value is set via the `EMAIL_REPLY_SECRET_KEY` settings.

The default settings that govern the email reply service are the following::

```
# What address pattern will handle the replies.
EMAIL_REPLY_PATTERN = "reply+%s+code@biostars.io"

# The format of the email address that is sent
EMAIL_FROM_PATTERN = u'''"%s on Biostar" <%s>'''

# The secret key that is required to parse the email
EMAIL_REPLY_SECRET_KEY = "abc"

# The subject of the reply goes here
EMAIL_REPLY_SUBJECT = u"[biostar] %s"
```

Note: when you set the alias remember to restart the services::

```
sudo postalias /etc/alias
sudo service postmaster restart
```

A simpler setup that requires no local SMTP servers could reply on commercial services such as mailgun and others.

Social authentication

The social logins settings will need to be initialized with the proper authentication parameters. Typically this involves creating an application at the provider and obtaining the credentials.

See the `conf/defaults.env` for the proper variable naming.

Adding Facebook authentication:

- Create Authentication App: http://developers.facebook.com/setup/
- More information: Facebook Developer Resources: http://developers.facebook.com/docs/authentication/

Adding Google authentication:

- Google Developer Console: https://cloud.google.com/console/project
- Create new project and copy data from credentials
- Callback must be `http://domain/accounts/google/login/callback/`

Twitter:

- Add your application at Twitter Apps Interface: http://twitter.com/apps/

ORCID:

- Enable "Developer Tools" in your ORCID account, following these instructions: http://support.orcid.org/knowledgebase/articles/343182-register-a-client-with-the-public-api
- Create new application: https://orcid.org/developer-tools
- Redirect URI must be `http://domain/accounts/orcid/login/callback/`

External authentication

Other domains can provide authentication for Biostar by setting a cookie with a certain value. For this to work Biostar will have to be set to run as a subdomain of the hosting site.

Cookie settings ^^^^^^^^^^^^^^^

The cookie value needs to contain the `email:hash` as value. For exampl if the `EXTERNAL_AUTH` django settings are::

```
# Cookie name, cookie secret key pair
EXTERNAL_AUTH = [
    ("foo.bar.com", "ABC"),
]
```

If an unauthenticated user sends a cookie named `foo.bar.com` with the value::

```
foo@bar.com:d46d8c07777e3adf739cfc0c432759b0
```

then Biostar will automatically log in the user. It will automatically create an account for the user if the email does not already exist.

Setting the `EXTERNAL_LOGIN_URL` and `EXTERNAL_LOGOUT_URL` settings will also perform the redirects to the external site login and logout urls::

```
EXTERNAL_LOGIN_URL = "http://some.site.com/login"
EXTERNAL_LOGOUT_URL = "http://some.site.com/logout"
```

Generating the value is simple like so::

```
email = "foo@bar.com"
digest = hmac.new(key, email).hexdigest()
value = "%s:%s" % (email, digest)
```

Prefill post ^^^^^^^^^^^^

Set the `title`, `tag_val`, `content` and `category` fields of a get request to pre-populate a question::

```
http://localhost:8080/p/new/post/?title=Need+help+with+bwa&tag_val=bwa+samtools&
↪content=What+does+it+do?&category=SNP-Calling
```

Migrating from Biostar 1.X

Due to the complete rework there is no database schema migration.

Instead users of Biostar 1 site are expected to export their data with a script provided in Biostar 1 then import it with a management command provided with Biostar 2.

The migration will take the following steps:

1. Set the `BIOSTAR_MIGRATE_DIR` environment variable to point to a work directory that will hold the temporary data, for example `export BIOSTAR_MIGRATE_DIR="~/tmp/biostar_export"`
2. Load the environment variables for the Biostar 1 site then run `python -m main.bin.export -u -p -v`. This will dump the contents of the site into the directory that `BIOSTAR_MIGRATE_DIR` points to.
3. Load the environment variables for you Biostar 2 site then run the `./biostar.sh import_biostar1` command.

Some caveats, depending how you set the variables you may need to be located in the root of your site. This applies for the default settings that both sites come with, as the root is determined relative to the directory that the command is run in.

##Manage

There are a number of data management commands that come with Biostar.

The biostar.sh manager

The **biostar.sh** shell command automatizes a number of commonly used tasks. Run it with no parameters to get help on a typical usage::

```
Usage:

  $ biostar.sh <command>

Multiple commands may be used on the same line:

  $ biostar.sh init import run

Commands:

  init       - initializes the database
  run        - runs the development server
  index      - initializes the search index
  test       - runs all tests
  env        - shows all customizable environment variables

  import     - imports the data fixture JSON_DATA_FIXTURE=import/default-fixture.json.
↪gz
  dump       - dumps data as JSON_DATA_FIXTURE=import/default-fixture.json.gz
  delete     - removes the sqlite database DATABASE_NAME=biostar.db

  pg_drop        - drops postgres DATABASE_NAME=biostar.db
  pg_create      - creates postgres DATABASE_NAME=biostar.db
  pg_import f.gz - imports the gzipped filename into postgres DATABASE_NAME=biostar.db

Use environment variables to customize settings. See the docs.

DJANGO_SETTINGS_MODULE=biostar.settings.base
```

Subcommands

In addition there are a number of data management commands that are implemented for the each app. Run::

```
python manage.py help
```

And look for the output for the app `[server]`, these commands will look like::

```
[server]
    biostar_pg_dump
    delete_database
    import_biostar1
    import_mbox
    initialize_site
    prune_data
    usermod
    sqlfix
    sitemap
    user_crawl
    test_email
    test_task
    patch
```

You can run each of these subcommands with the `-h` flag to get more information on them.

Command line tagging

There is a command line tool to perform content tagging based on a regular expression. The invocation is::

```
workon biostar
source live/deploy.env
python manage.py patch --tag "regexp:tag1,tag2,tag3"
```

Where the regular expression `regexp` will be searched against the content and when found matching tags `tag1`, `tag2`, `tag3` will be applied. Example::

```
python manage.py patch --tag "gff:gff,interval"
```

To detect what posts would be tagged but not actually perform the tagging pass the `--dry` command. In that case only the post titles will be listed::

```
python manage.py patch --tag "gff:gff,interval" --dry
```

This command will navigate through all questions in the database.

Example commands

Frequently used commands::

```
# Set the password for a user identified by their userid
python manage.py usermod -u 2 -p abcde

# Set the password for a user identified by their email
python manage.py usermod -e foo@bar -p abcde

# Rebuild the entire search index
python manage.py rebuild_index
```

```
# Reindex only what has changed in the last hour
python manage.py update_index --age 1

# Import 100 posts from a mbox file into biostar
python manage.py import_mbox -f filename -l 100

# Create a postgres database dump
python manage.py biostar_pg_dump
```

Merging Users

Create a space separated text file that contains the emails in the form::

```
master_email alias_email1 alias_email2 ...
```

Then run the command::

```
python manage.py patch --merge_users yourfile.txt
```

The command will move all content, votes and accounts associated with users identified by the aliases into the master email. It then deletes the alias users. The effect of this command cannot be reverted other than loading up a backup database dump.

Deploy

Getting started

Thanks to the modular structure of its design Biostar is able to integrate with a wide variety of backends and provides a number of configuration scripts and helper methods to different deployment options.

The choices made when deploying Biostar depend on the expected levels of traffic and number of posts that the site needs to manage. The examples that we provide are the two extremes, some deployments may use a combination of settings from both.

Example files can be found in the `live` folder named `deploy.env` and `deploy.py`.

The basic rule is to create a settings file based on the default settings. This means that the customized settings file will start with::

```
from biostar.settings.base import *
```

Then subsequently override the various settings for the current deployment. For example::

```
from biostar.settings.base import *
SITE_DOMAIN = "mysite.com"
SERVER_EMAIL = "myemail@mysite.com"
```

etc.

Technically a django deployment needs only a settings file, but in practice we use an environment file to populate a shell environment and a settings file that pulls some of these variables out of the environment.

We recommend that you start with the files in `live/deploy*` and copy them another name. The `deploy.env` and `deploy.py` files show the minimally necessary variables that need to be set.

```
source live/deploy.env
./biostar.sh test
```

The `deploy.env` must specify the correct django settings module in this case `live.deploy` that will load the `live/deploy.py` python module.

To run periodic scripts make sure that they load up the enviroment variables before executing the script.

Low traffic deployment

Suited to websites that distribute information to smaller organizations. It can be achieved with just python based solutions. Install the dependencies with::

```
pip install -r conf/requirements/deploy.txt
```

Copy the `live/deploy.env` and `live/deploy.py` files to a different name/location. For example `simple.env` and `simple.py`. Customize these as needed. To run the site invoke the waitress server that was installed above::

```
source live/simple.env
waitress-serve --port 8080 live.deploy.simple_wsgi:application
```

Create a crontab entry that updates the index every 30 minutes::

```
source live/simple.env
biostar.sh update_index
```

You are done.

High traffic deployment

While not required to be turned on the site supports compressing and precompiling the site assets. To make use of this functionality you will need to have `lessc` to be installed and you will need to set the `USE_COMPRESSOR=True` in your settings file.

To deploy the site with `postgresql` and `elasticsearch` install the requirements::

```
pip install --upgrade -r conf/requirements/deploy.txt
```

Start with the `conf/defaults.env` and files unde `conf/deploy/*` and customize them. We typically copy these into the `live` folder. Rember to add an `__init__.py` file in this folder if you want to import your settings from it.

For high performance installation we recommend deploying the production servers with the following stack:

- Front end webserver with `nginx`
- Biostar WSGI running via `gunicorn`
- `Postgresql` as the database
- `Redis` as the job queue
- `Celery` for running the asynchronous jobs

- `Supervisord` keeping everything running
- `Elasticsearch` as the search engine

The `conf/server` folder has configuration files for `nginx`, `gunicorn` and `supervisord`. The `conf/fabs` folder has Fabric files to automate a large number of site deployment operations.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search