repocribro Documentation

Release 0.1

Marek Suchánek

May 19, 2020

Basic Information

Welcome in the **repocribro** documentation. Continue by choosing the desired topic from the list of contents. You can also visit the repository repocribro@GitHub.

CHAPTER 1

Contents

1.1 Introduction

repocribro is Python powered web application allowing users to register their GitHub repositories so they can be managed, searched, browsed, tested, etc. (depends on used extensions) within the app. Main idea is to provide simple but powerful modular tool for building groups of GitHub repositories which are developed by different users and organizations with some common goal.

Cribro means sieve in Italian language (origins in Latin word *cribrum*). This project provides tool for intelligent sifting repositories, information about them and its contents.

This project has been created as final semester work for subject MI-PYT, CTU in Prague (more in *Credits*).

Project is open-source (under MIT license) published @GitHub. Basically you just need to always include the copyright and permission notice with name of author. But it would be great if you let us know that you are using **repocribro** in any way!!!

1.2 Installation

1.2.1 Requirements

- Python 3.5+
- Installed dependencies (automatic with `setup.py`)
- *Configuration* prepared
- DB supported by SQLAlchemy

1.2.2 Installation options

This application can be installed via standard `setuptools`, for more information read Python docs - Installing Python Module. Check the *Requirements* before installation.

PyPi

- https://pypi.python.org/pypi/repocribro

You can use pip tool to install the package **repocribro** from PyPi:

```
$ pip install repocribro
```

setup.py

Or download the repository from GitHub and run:

```
$ python3 setup.py install
```

Check installation

After the successful installation you should be able to run:

```
$ repocribro --version
repocribro v0.1
```

1.2.3 Configuration

You can see example configuration files at:

- `config/app.example.cfg`
- `config/auth.example.cfg`
- `config/db.example.cfg`

  !!! If you are going to publish your configuration somewhere make sure, that it does not contain any secret information like passwords or API tokens!

Syntax of configuration files is standard INI, parsed by ConfigParser. Names of variables are case insensitive. Configuration can be in separate configuration files but if there are same variables within same sections there will overriding depending on the order of files.

Default config file can be also specified with environment variable:

```
$ export REPOCRIBRO_CONFIG_FILE='/path/to/config.cfg'
$ python
Python 3.5.2 (default, Oct 14 2016, 12:54:53)
[GCC 6.2.1 20160916 (Red Hat 6.2.1-2)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from repocribro.app import app
>>> app.run()
```

You can also take advantage of python-dotenv and specify configuration in *.env* file.

Application

You can specify any of the Flask (or extensions) configuration variables that is supposed to be placed to `app.config`. Just use same name (it can be also lowercase). These configurations must be done in `[flask]` section. Mandatory attribute is `SECRET_KEY` used for the session signing, this key is of course **private**.

- http://flask.pocoo.org/docs/0.12/config/#builtin-configuration-values

For example:

```
[flask]
# something is wrong, I want to debug
DEBUG = true
# random secret key (use os.urandom())
SECRET_KEY = VeryPseudoRandomSuchSecret
```

Webroot in subdirectory

If you are running the application in subdirectiory and not whole (sub)domain (e.g. myportal.xyz/repocribro/), then you probably want to use the setting `APPLICATION_ROOT` which will in case of repocribro not only set appropriately session path but also make URLs correct and working.

Listing 1: Repocribro config (part)

```
[flask]
APPLICATION_ROOT = /repocribro
# ... other config
```

Listing 2: NGINX config (part)

```
location ~ /repocribro(/.*)$ {
    proxy_pass http://127.0.0.1:5000/repocribro$1;
            proxy_set_header Host $host;
}
```

Database

Next you need to specify configuration of your database. Flask extension Flask-SQLAlchemy is used so again configuration needs to be done within section `[flask]`.

- http://flask-sqlalchemy.pocoo.org/2.1/config/#configuration-keys

  !!! If file contains DB password and username keep it **private**!

For example:

```
[flask]
# SQLite is enough, just testing
SQLALCHEMY_DATABASE_URI = sqlite:////tmp/test.db
```

GitHub

For communication with GitHub OAuth you are going to need **Client ID** and **Client secret**. Also for working with webhooks secret key must be set-up so every incoming message can be verified. Specify those in `[github]` section of config.

- https://developer.github.com/v3/oauth/
- https://github.com/settings/applications/new
- https://developer.github.com/webhooks/securing/

!!! Always keep file with this configuration **private**!

For example:

```
[github]
# Client ID & secret is obtained by creating OAuth app
CLIENT_ID = myAppClientIdFromGitHub
CLIENT_SECRET = myAppClientSecretFromGitHub
# Webhook secret for signing should be randomly generated
WEBHOOKS_SECRET = someRandomSecretKeyForWebhooks
```

Core customization

You can specify name and logo for your deployment of repocribro within `repocribro-core` section. More options will be added later.

For example:

```
[repocribro-core]
# custom name
NAME = myRepocribro
# custom logo URL
LOGO = https://upload.wikimedia.org/wikipedia/commons/thumb/2/2f/Logo_TV_2015.svg/
↪2000px-Logo_TV_2015.svg.png
# landing page text
LANDING_TEXT = <p>Landing text paragraph number 1</p>
               <p>Landing text paragraph number 2</p>
# landing page picture (defaults to LOGO)
LANDING_PICTURE = https://assets-cdn.github.com/images/modules/logos_page/Octocat.png
# navbar classes (dark/light, defaults to dark)
NAVBAR_STYLE = light
```

1.2.4 Database

In order to create and maintain the database, you can use migrations by Flask-Migrate:

```
$ repocribro db --help
```

Or you can use standard SQLAlchemy procedure `db.create_all()` via:

```
$ repocribro create-db
```

Both will try to create tables into database specified in the *Configuration*.

1.2.5 Become an admin

After first start you should login into web app via GitHub and then you can use `assign-role` command to become an `admin`.

```
$ repocribro assign-role --login MarekSuchanek --role admin
Loaded extensions: core
Role admin not in DB... adding
Role admin added to MarekSuchanek
```

1.2.6 Docker

There is a `Dockerfile` prepared for simpler deployment and use of repocribro. Just prepare the config file and use Docker as you usually do:

```
docker build -t repocribro
docker run repocribro -d -p 5000:5000 repocribro [COMMAND]
```

1.3 Usage

1.3.1 Usage basics

First you need to have prepared config file(s) with at least minimal mandatory configuration and **repocribro** successfully installed(see *Installation* and *Configuration*).

```
$ repocribro --config <config_file> [command] [command options]
$ repocribro -c <config_file> [command] [command options]
$ repocribro -c <config_file_1> -c <config_file_2> [command] [command options]
```

For all commands you can specify configuration file(s) of **repocribro** app, order of arguments matters only if you are overriding same configuration variable in those files. If no config files are specified those from default path will be used.

Commands

After supplying configuration files you can use various commands. Full list of commands with details are described within *CLI commands*.

For starting the web application (server) use:

```
$ repocribro runserver
```

Common options

You can also use standard `-?`, `--help` and `--version`:

```
$ repocribro --help
usage: repocribro [-c CFG_FILES] [-v] [-?] {runserver,db,shell,repocheck} ...

positional arguments:
  {runserver,db,shell,repocheck}
    runserver           Runs the Flask development server i.e. app.run()
    db                  Perform database migrations
    shell               Runs a Python shell inside Flask application context.
    repocheck           Perform check procedure of repository events
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -c CFG_FILES, --config CFG_FILES
  -v, --version         show program's version number and exit
  -?, --help            show this help message and exit

$ repocribro --version
repocribro v0.0
```

1.3.2 CLI commands

There are various command for the app management some are provided by Flask extensions, some by repocribro. You can use option `--help` to get more information.

You can use both `repocribro` and `flask` commands. The configuration must by specified by environenment variable as described in *Configuration*.

assign_role

Main purpose for this command is to set the initial admin of the app without touching DB directly. Others can be then set within administration zone of web interface.

```
$ repocribro assign_role --login MarekSuchanek --role admin
Loaded extensions: core
Role admin not in DB... adding
Role admin added to MarekSuchanek

$ repocribro assign-role --login MarekSuchanek --role admin
Loaded extensions: core
User MarekSuchanek already has role admin
```

For more information:

```
$ repocribro assign_role --help
```

check_config

Commands for checking configuration currently used by repocribro. There are two styles for printing, same syntax as is in the cfg file (default) or just triples section key value.

```
$ repocribro check_config
[flask]
secret_key = MySecretKey
...

$ repocribro check_config --style triple
flask secret_key MySecretKey
...
```

db (database)

Command supplied by Flask-Migrate extension provides tool to work with database migrations (init, migrate, downgrade, upgrade, etc.).

For more information:

```
$ repocribro db --help
```

repocheck

This command provides simple checking of one or all repositories if there are some uncaught events within specified time. Main idea is to get the missed events (from webhooks) due to app outage.

```
$ repocribro repocheck --help
```

runserver

Runs the web application (`app.run()`), but also some settings can be overriden like hostname, port, debugging, ...

For more information:

```
$ repocribro run --help
```

shell

Runs the Python shell inside the Flask app context. That can be useful for some debugging by hand.

For more information:

```
$ repocribro shell --help
```

1.3.3 Web application

Public usage

Anonymous (unauthenticated) used can browse public content of the web application which includes:

- landing with basic info
- search
- user/organization profiles
- public repositories with their releases and updates

Authentication

User can authenticate via GitHub OAuth with scope:

- `repo` = read and manage user repositories (with private)
- `user` = read user information (with private)
- `admin:webhook` = add/remove webhooks

User management

Every activate (not banned) user can manage which of his/her repositories should be listed within application as:

- public = everyone can see them
- hidden = only people with secret URL can see them
- private = only owner or administrator can see them

Information about user account can be synchronized as well as the repository information. When activating the repository webhook is added. Because webhook can be deleted at GitHub by hand, user can recreate the webhook again (he can't do it by hand because doesn't know the webhooks secret).

Administration

Managing user accounts, roles and repositories (not owned) can be done in administration zone. Same principles as in user management zone.

REST API

There is also REST API (only GET) for all GitHub entities, but it will be reworked soon (because the repo privacy & compatibility issues).

The actual is done by Flask-Restless with collections:

- `user`
- `org`
- `repo`
- `push`
- `commit`
- `release`

1.3.4 Privileges and Roles

In Repocribro, there are defined roles that can be assigned to user accounts. Each account can have several roles and role can be assigned to several accounts. Some pages and actions can be accessible just to some roles.

On top of that, to allow higher granularity of permissions, each role has a list of action privileges. Some actions can be restricted with such privilege instead of whole role.

Roles and privileges can be simply managed in the administration web interface of repocribro. As administrator, you can define new roles with different privileges and assign user to them. Repocribro implements simple wildcarding of action privileges, so `*` for role `admin` means that the role can perform all actions defined now or in the future.

Core roles

It is recommended to have three roles described in the table below - for not-logged users, default for users and for administrators.

Table 1: Core roles

Name	Privileges	Description
admin	`*`	Service administrators
user	TBD	Regular users
anonymous	`search*:login`	Not-logged users

Core action privileges

There action privileges are defined in the core and can be used.

Table 2: Core action privileges

Name	Description
search	use search engine
login	login to Repocribro via GitHub to app
logout	logout from Repocribro
browse	visit basic pages like landing
browse_user	visit detail of user
browse_org	visit detail of organization
browse_repo	visit detail of repository that is public or owned-private
browse_repo_hidden	visit detail of repository that is hidden
manage_dashboard	access to account management dashboard
manage_profile_update	update profile information from GitHub
manage_repos	list GitHub repositories
manage_repo	manage single repository within Repocribro
manage_repo_update	update repository information from GitHub
manage_repo_delete	delete repository from Repocribro
manage_repo_activate	activate repository within Repocribro
manage_repo_deactivate	deactivate webhook for Repocribro
manage_orgs	list GitHub organizations
manage_org	manage single organization
manage_org_update	update organization information from GitHub
manage_org_delete	delete organization from Repocribro

1.4 Credits

This project was created as final semester work for the awesome subject MI-PYT (Advanced Python) taught at the Faculty of Information Technology, Czech Technical University in Prague (FIT CTU) by @hroncok and @encukou.

Thanks goes to the Python community, to developers, contributors and other people around projects that are used within **repocribro**:

- requests
- Flask (and extensions)
- Jinja
- pytest (and extensions)
- Sphinx
- SQLAlchemy

Also many thanks to GitHub, Travis CI, coveralls.io, readthedocs.org, requires.io and PyPi for being here for all of us.

1.5 Testing

This project uses the most fabulous testing tools for Python:

- pytest
- pytest-cov
- pytest-pep8
- betamax

1.5.1 Run tests

Run tests simply by:

```
python setup.py test
```

or (if you have installed dependencies):

```
python -m pytest [options]
pytest [options]
```

You can also see the tests logs at Travis CI.

1.5.2 Betamax cassettes

Betamax cassettes are stored in `tests/fixtures/cassettes` directory. If you are not connected to theInternet, GitHub API is not working and/or you don't want to create own GitHub token you will use (replay) them in order to test API client.

If you want to run your own cassettes, you need to setup system variable `GITHUB_TOKEN` which will contain the GitHub personal token (must have privileges to create/delete webhooks). You also must change variables within `tests/test_github.py` specifying some of your existing repository and also non-existing repository. Token can be created at:

- https://github.com/settings/tokens

Your test command then might look like:

```
$ GITHUB_TOKEN=<YOUR_TOKEN> python setup.py test
```

or use `export` and `unset`:

```
$ export GITHUB_TOKEN=<YOUR_TOKEN>
$ python setup.py test
...
$ unset GITHUB_TOKEN
```

For more information, enjoy reading Betamax documentation.

1.6 TODO list

Todo: handle pagination of GitHub events

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/commands/re
of repocribro.commands.repocheck.RepocheckCommand._do_check, line 8.)

Todo: implement 410 (org deleted/archived/renamed)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/co
of repocribro.controllers.core.org_detail, line 3.)

Todo: implement 410 (repo deleted/archived/renamed)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/co
of repocribro.controllers.core.repo_detail_common, line 3.)

Todo: more attrs, limits & pages

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/co
of repocribro.controllers.core.search, line 3.)

Todo: implement 410 (user deleted/archived/renamed)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/co
of repocribro.controllers.core.user_detail, line 3.)

Todo: move somewhere else, check registered events

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/ma
of repocribro.controllers.manage.has_good_webhook, line 10.)

Todo: protect from updating too often

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/ma
of repocribro.controllers.manage.profile_update, line 3.)

Todo: protect from activating too often

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/ma
of repocribro.controllers.manage.repository_activate, line 3.)

Todo: consider deleting org repository if there are more members

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/ma
of repocribro.controllers.manage.repository_delete, line 3.)

Todo: protect from updating too often

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/ma
of repocribro.controllers.manage.repository_update, line 3.)

Todo: move somewhere else

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/controllers/ma
of repocribro.controllers.manage.update_webhook, line 10.)

Todo: Consider loading/asking order not by priority but by dependencies

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/extending/ext
of repocribro.extending.Extension, line 10.)

Todo: There might be some problem with ordering of extensions

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/extending/ext
of repocribro.extending.ExtensionsMaster.__init__, line 6.)

Todo: handle if GitHub is out of service, custom errors, better abstraction, work with extensions

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/github.py:doc
of repocribro.github.GitHubAPI, line 6.)

Todo: check granted scope vs GH_SCOPES

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/github.py:doc
of repocribro.github.GitHubAPI.login, line 8.)

Todo: verify, there are some conflict in GitHub docs

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/models.py:do
of repocribro.models.Commit.create_from_dict, line 10.)

Todo: How about some past events before adding to app?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/repocribro/checkouts/latest/repocribro/models.py:do
of repocribro.models.Repository.events_updated, line 3.)

1.7 Writing extensions

1.7.1 Hooks for extension

Table 3: Extension hooks

Name	Description	Return type
`get_gh_event_processors`	Get GitHub events processors	`dict of str: list of function`
`get_gh_webhook_processors`	Get GitHub webhook processors	`dict of str: list of function`
`init_business`	Init business layer of the extension	`None`
`init_security`	Init security of the extensions (define roles and privileges)	`None`
`init_blueprints`	Init Flask blueprints (register them)	`None`
`init_container`	Put whatever you need into DI container of the app	`None`
`init_filters`	Init Jinja2 filters for views (register them)	`None`
`init_models`	Init models (data) layer of the extension	`array`
`introduce`	Introduce yourself (just name) for the list of extensions	`str`
`view_admin_extensions`	Return view object of extension for the admin list	`ExtensionView`
`view_admin_index_tabs`	Add/edit tabs for admin index page	`dict of str:ViewTab`
`view_core_search_tabs`	Add/edit tabs for search results page	`dict of str:ViewTab`
`view_core_user_detail_tabs`	Add/edit tabs for public user page	`dict of str:ViewTab`
`view_core_org_detail_tabs`	Add/edit tabs for public organization page	`dict of str:ViewTab`
`view_core_repo_detail_tabs`	Add/edit tabs for repository detail page	`dict of str:ViewTab`
`view_manage_dashboard_tabs`	Add/edit tabs for user management zone dashboard	`dict of str: `ViewTab`

1.7.2 Privileges and Roles

Again, good example is `repocribro.ext_core`. Within your extension you can define new roles and action privileges by methods `provide_roles` and `provide_actions`. It uses flask-principal wrapped in `flask.security.permissions` object. After registration of roles and action privileges, you can simply import the object and use it to decorate controller function like this:

```
import flask
from ..security import permissions

showcase = flask.Blueprint('showcase', __name__, url_prefix='/showcase')

@showcase.route('test1')
@permissions.roles.my_new_role.require(403)
def test1():
    ...

@showcase.route('test2')
@permissions.actions.test_it.require(403)
def test2():
    ...
```

For basics about privileges, see *Privileges and Roles*.

You can write your own **repocribro** extension. It's very simple, all you need is extend the `Extension` class from `repocribro.extending`, make function returning instance of this class and direct entrypoint in the group `[repocribro.ext]` on that function. Extending is done via implementing actions on *Hooks for extension* which can return something.

While writing new plugin use please the same model, so even your extension is also easily extensible. Big part of core repocribro is extension itself see the module `repocribro.ext_core`.

1.7.3 my_ext.py

```
from repocribro.extending import Extension


class MyNewExtension(Extension):
    ...


def make_my_new_extension():
    ...
    return MyNewExtension()
```

1.7.4 setup.py

```
from setuptools import setup

...
setup(
    ...
    entry_points={
        'repocribro.ext': [
            'repocribro-my_ext = my_ext:make_my_new_extension'
        ]
    },
    ...
)
...
```

1.8 API

repocribro consists of following package(s) and it's modules:

1.8.1 repocribro.manage

1.8.2 repocribro.commands

assign_role

`repocribro.commands.assign_role.`**`_assign_role`**(*login*, *role_name*)

check_config

repocribro.commands.check_config.**_check_config**(*style*)

db_create

repocribro.commands.db_create.**_db_create**()

repocheck

class repocribro.commands.repocheck.**RepocheckCommand**

__dict__ = mappingproxy({'__module__': 'repocribro.commands.repocheck', 'event2webhoo

__module__ = 'repocribro.commands.repocheck'

__weakref__
list of weak references to the object (if defined)

_do_check(*repo*)
Perform single repository check for new events

Parameters **repo** (repocribro.models.Repository) – Repository to be checked

Raises **SystemExit** – if GitHub API request fails

Todo: handle pagination of GitHub events

_process_event(*repo*, *event*)
Process potentially new event for repository

Parameters

- **repo** (repocribro.models.Repository) – Repository related to event
- **event** (*dict*) – GitHub event data

Returns If the event was new or already registered before

Return type bool

event2webhook = {'PushEvent': 'push', 'ReleaseEvent': 'release', 'RepositoryEvent':

run(*full_name=None*)
Run the repocheck command to check repo(s) new events

Obviously this procedure can check events only on public repositories. If name of repository is not specified, then procedure will be called on all registered public repositories in DB.

Parameters **full_name** (*str*) – Name of repository to be checked (if None -> all)

Raises **SystemExit** – If repository with given full_name does not exist

repocribro.commands.repocheck.**_repocheck**(*full_name=None*)

1.8.3 repocribro.config

class repocribro.config.**Config**(**args*, ***kwargs*)
Repocribro app configuration container

check()
Check and correct missing mandatory options and sections

Returns List of errors (string with section and option)

Return type list of str

default
Access defaults as property

Returns Default configuration options

Return type dict

mark_mandatory(*section*, *option*)
Mark some option within section as mandatory

Parameters

- **section** (*str*) – Section with mandatory option
- **option** (*str*) – Option to be mandatory

read_env(*section*, *option*, *env_name*)
Shorthand for reading ENV variable into config

Parameters

- **section** (*str*) – Target config section
- **option** (*str*) – Target config option
- **env_name** (*str*) – Name of ENV variable

read_envs(*prefix*)
Shorthand for reading ENV variables with given prefix into config

This will load all ENV variables with given prefix, the section name is after prefix and next underscore so section name can not contain undercore. For example `REPOCRIBRO_FLASK_SECRET_KEY` will be loaded to section `FLASK` and option `SECRET_KEY` (`REPOCRIBRO` is the prefix).

Parameters **prefix** (*str*) – ENV variable name prefix

update_flask_cfg(*app*)
All options from flask section will be inserted to config of the given Flask app

Parameters **app** (`flask.Flask`) – Flask application to be configured

repocribro.config.**check_config**(*config*, *exit_code=1*)
Procedure for checking mandatory config

If there are some missing mandatory configurations this procedure prints info on stderr and exits program with specified exit code.

Parameters

- **config** (`repocribro.config.Config`) – Configuration object
- **exit_code** (*int*) – Exit code on fail

Raises SystemExit

repocribro.config.**create_config**(*cfg_files*, *env_prefix='REPOCRIBRO'*)
Factory for making Repocribro config object

Parameters **cfg_files** – Single or more config file(s)

Returns Constructed config object

Return type repocribro.config.Config

1.8.4 repocribro.controllers

repocribro.controllers.admin

repocribro.controllers.admin.**account_ban**(*login*)
Ban (make inactive) account (POST handler)

repocribro.controllers.admin.**account_delete**(*login*)
Delete account (POST handler)

repocribro.controllers.admin.**account_detail**(*login*)
Account administration (GET handler)

repocribro.controllers.admin.**admin = <flask.blueprints.Blueprint object>**
Admin controller blueprint

repocribro.controllers.admin.**index**()
Administration zone dashboard (GET handler)

repocribro.controllers.admin.**repo_delete**(*login*, *reponame*)
Delete repository (POST handler)

repocribro.controllers.admin.**repo_detail**(*login*, *reponame*)
Repository administration (GET handler)

repocribro.controllers.admin.**repo_visibility**(*login*, *reponame*)
Change repository visibility (POST handler)

repocribro.controllers.admin.**role_assignment_add**(*name*)
Assign role to user (POST handler)

repocribro.controllers.admin.**role_assignment_remove**(*name*)
Remove assignment of role to user (POST handler)

repocribro.controllers.admin.**role_create**()
Create new role (POST handler)

repocribro.controllers.admin.**role_delete**(*name*)
Delete role (POST handler)

repocribro.controllers.admin.**role_detail**(*name*)
Role administration (GET handler)

repocribro.controllers.admin.**role_edit**(*name*)
Edit role (POST handler)

repocribro.controllers.auth

repocribro.controllers.auth.**auth = <flask.blueprints.Blueprint object>**
Auth controller blueprint

repocribro.controllers.auth.**github**()
Redirect to GitHub OAuth gate (GET handler)

repocribro.controllers.auth.**github_callback**()
Callback gate for GitHub OAUTH (GET handler)

repocribro.controllers.auth.**github_callback_get_account**(*db*, *gh_api*)
Processing GitHub callback action

Parameters

- **db** (flask_sqlalchemy.SQLAlchemy) – Database for storing GitHub user info
- **gh_api** (repocribro.github.GitHubAPI) – GitHub API client ready for the communication

Returns User account and flag if it's new one

Return type tuple of repocribro.models.UserAccount, bool

repocribro.controllers.auth.**logout**()
Logout currently logged user (GET handler)

repocribro.controllers.core

repocribro.controllers.core.**core = <flask.blueprints.Blueprint object>**
Core controller blueprint

repocribro.controllers.core.**index**()
Landing page (GET handler)

repocribro.controllers.core.**org_detail**(*login*)
Organization detail (GET handler)

Todo: implement 410 (org deleted/archived/renamed)

---

repocribro.controllers.core.**repo_detail**(*login*, *reponame*)
Repo detail (GET handler)

repocribro.controllers.core.**repo_detail_common**(*db*, *ext_master*, *repo*, *has_secret=False*)
Repo detail (for GET handlers)

Todo: implement 410 (repo deleted/archived/renamed)

---

repocribro.controllers.core.**repo_detail_hidden**(*secret*)
Hidden repo detail (GET handler)

repocribro.controllers.core.**repo_redir**(*login*)

repocribro.controllers.core.**search**(*query=''*)
Search page (GET handler)

Todo: more attrs, limits & pages

---

repocribro.controllers.core.**user_detail**(*login*)
User detail (GET handler)

Todo: implement 410 (user deleted/archived/renamed)

repocribro.controllers.errors

repocribro.controllers.errors.**err_forbidden**(*error*)
Error handler for HTTP 403 - Unauthorized

repocribro.controllers.errors.**err_gone**(*error*)
Error handler for HTTP 410 - Gone

repocribro.controllers.errors.**err_internal**(*error*)
Error handler for HTTP 501 - Not Implemented

repocribro.controllers.errors.**err_not_found**(*error*)
Error handler for HTTP 403 - Not Found

repocribro.controllers.errors.**errors = <flask.blueprints.Blueprint object>**
Errors controller blueprint

repocribro.controllers.manage

repocribro.controllers.manage.**dashboard**()
Management zone dashboard (GET handler)

repocribro.controllers.manage.**get_repo_if_admin**(*db*, *full_name*)

Retrieve repository from db and return if current user is admin (owner or member)

Parameters

- **db** (flask_sqlalchemy.SQLAlchemy) – database connection where are repos stored
- **full_name** (*str*) – full name of desired repository

Returns repository if found, None otherwise

Return type repocribro.models.Repository or None

repocribro.controllers.manage.**has_good_webhook**(*gh_api*, *repo*)
Check webhook at GitHub for repo

Parameters

- **gh_api** (repocribro.github.GitHubAPI) – GitHub API client for communication
- **repo** (repocribro.models.Repository) – Repository which webhook should be checked

Returns If webhook is already in good shape

Return type bool

Todo: move somewhere else, check registered events

repocribro.controllers.manage.**manage = <flask.blueprints.Blueprint object>**
Manage controller blueprint

repocribro.controllers.manage.**organization**(*login*)
List organization repositories for activation

repocribro.controllers.manage.**organization_delete**(*login*)
Delete organization (if no repositories)

repocribro.controllers.manage.**organization_update**(*login*)
Update organization

repocribro.controllers.manage.**organizations**()
List user organizations from GitHub (GET handler)

repocribro.controllers.manage.**profile_update**()
Update user info from GitHub (GET handler)

Todo: protect from updating too often

repocribro.controllers.manage.**repositories**()
List user repositories from GitHub (GET handler)

repocribro.controllers.manage.**repository_activate**()
Activate repo in app from GitHub (POST handler)

Todo: protect from activating too often

repocribro.controllers.manage.**repository_deactivate**()
Deactivate repo in app from GitHub (POST handler)

repocribro.controllers.manage.**repository_delete**()
Delete repo (in app) from GitHub (POST handler)

Todo: consider deleting org repository if there are more members

repocribro.controllers.manage.**repository_detail**(*full_name*)
Repository detail (GET handler)

repocribro.controllers.manage.**repository_update**()
Update repo info from GitHub (POST handler)

Todo: protect from updating too often

repocribro.controllers.manage.**update_webhook**(*gh_api*, *repo*)
Update webhook at GitHub for repo if needed

Parameters

- **gh_api** (repocribro.github.GitHubAPI) – GitHub API client for communication
- **repo** (repocribro.models.Repository) – Repository which webhook should be updated

Returns If webhook is now in good shape

Return type bool

Todo: move somewhere else

repocribro.controllers.webhooks

repocribro.controllers.webhooks.**gh_webhook**()
Point for GitHub webhook msgs (POST handler)

1.8.5 repocribro.ext_core

CoreExtension

class repocribro.ext_core.**CoreExtension**(*master*, *app*, *db*)
Bases: repocribro.extending.extension.Extension

ADMIN_URL = None

AUTHOR = 'Marek Suchánek'
Author of core extension

CATEGORY = 'basic'
Category of core extension

GH_URL = 'https://github.com/MarekSuchanek/repocribro'
GitHub URL of core extension

HOME_URL = None

NAME = 'core'
Name of core extension

PRIORITY = 0
Priority of core extension

__init__(*master*, *app*, *db*)
Inits the basic two parts of repocribro - flask app and DB

Parameters

- **master** (ExtensionsMaster) – Master for this extension
- **app** (flask.Flask) – Flask application of repocribro
- **db** (flask_sqlalchemy.SQLAlchemy) – SQLAlchemy database of repocribro
- **args** – not used
- **kwargs** – not used

call(*hook_name*, *default*, **args*, ***kwargs*)
Call the operation via hook name

Parameters

- **hook_name** (*str*) – Name of hook to be called
- **default** – Default return value if hook operation not found
- **args** – Positional args to be passed to the hook operation
- **kwargs** – Keywords args to be passed to the hook operation

Returns Result of the operation on the requested hook

static get_gh_event_processors()
Get all GitHub events processors

static get_gh_webhook_processors()
Get all GitHub webhooks processory

init_blueprints()
Hook operation for initiating the blueprints and registering them within repocribro Flask app

init_business()
Init business layer (other extensions, what is needed)

init_container()
Init service DI container of the app

init_filters()
Hook operation for initiating the Jinja filters and registering them within Jinja env of repocribro Flask app

init_models()
Hook operation for initiating the models and registering them within db

init_security()
Hook operation to setup privileges (roles and actions)

init_template_vars()

introduce()
Hook operation for getting short introduction of extension (mostly for debug/log purpose)

Returns Name of the extension

Return type str

static provide_actions()
Extension can define actions for privileges

Returns List of action names

Return type list of str

static provide_blueprints()
Extension can provide Flask blueprints to the app by this method

Returns List of Flask blueprints provided by extension

Return type list of `flask.blueprint`

static provide_dropdown_menu_items()

static provide_filters()
Extension can provide Jinja filters to the app by this method

Returns Dictionary with name + function/filter pairs

Return type dict of str: function

static provide_models()
Extension can provide (DB) models to the app by this method

Returns List of models provided by extension

Return type list of `db.Model`

static provide_roles()
Extension can define roles for user accounts

Returns Dictionary with name + Role entity

Return type dict of str: `repocribro.models.Role`

static provide_template_loader()

register_blueprints_from_list(*blueprints*)
Registering Flask blueprints to the app

Parameters **blueprints** (`list` of `flask.blueprint`) – List of Flask blueprints to be registered

register_filters_from_dict(*filters*)
Registering functions as Jinja filters

Parameters **filters** (*dict of str: function*) – Dictionary where key is name of filter and value is the function serving as filter

setup_config()
Setup necessary configuration attributes

view_admin_extensions()
Hook operation for getting view model of the extension in order to show it in the administration of app

Returns Extensions view for this extension

Return type `repocribro.extending.helpers.ExtensionView`

view_admin_index_tabs(*tabs_dict*)
Prepare tabs for index view of admin controller

Parameters **tabs_dict** (dict of str: `repocribro.extending.helpers.ViewTab`) – Target dictionary for tabs

view_core_org_detail_tabs(*org*, *tabs_dict*)
Prepare tabs for org detail view of core controller

Parameters

- **org** (`repocribro.models.Organization`) – Organization which details should be shown
- **tabs_dict** (dict of str: `repocribro.extending.helpers.ViewTab`) – Target dictionary for tabs

view_core_repo_detail_tabs(*repo*, *tabs_dict*)
Prepare tabs for repo detail view of core controller

Parameters

- **repo** (`repocribro.models.Repository`) – Repository which details should be shown
- **tabs_dict** (dict of str: `repocribro.extending.helpers.ViewTab`) – Target dictionary for tabs

view_core_search_tabs(*query*, *tabs_dict*)
Prepare tabs for search view of core controller

Parameters

- **query** (*str*) – Fulltext query for the search
- **tabs_dict** (dict of str: `repocribro.extending.helpers.ViewTab`) – Target dictionary for tabs

view_core_user_detail_tabs(*user*, *tabs_dict*)
Prepare tabs for user detail view of core controller

Parameters

- **user** (repocribro.models.User) – User which details should be shown
- **tabs_dict** (dict of str: repocribro.extending.helpers.ViewTab) – Target dictionary for tabs

view_manage_dashboard_tabs(*tabs_dict*)
Prepare tabs for dashboard view of manage controller

Parameters **tabs_dict** (dict of str: repocribro.extending.helpers.ViewTab) – Target dictionary for tabs

make_extension

repocribro.ext_core.**make_extension**(**args*, ***kwargs*)
Alias for instantiating the extension

Actually not needed, just example that here can be something more complex to do before creating the extension.

1.8.6 repocribro.extending

Extension

class repocribro.extending.**Extension**(*master*, *app*, *db*)
Bases: object

Generic **repocribro** extension class

It serves as base extension which does nothing but has prepared all the attributes and methods needed. Particular real extensions can override those attributes and methods to make so behavior and extend repocribro. It also provides some useful methods to those subclasses.

Todo: Consider loading/asking order not by priority but by dependencies

---

ADMIN_URL = None
Administration URL within site (best via url_for)

AUTHOR = ''
Author(s) of extension

CATEGORY = ''
Category of extension (basic, security, data, . . .)

GH_URL = None
GitHub url of extension project

HOME_URL = None
Homepage url of extension (rtd, pocoo, . . .)

NAME = 'unknown'
Name of extension

PRIORITY = 1000
Priority (lower will be loaded/asked sooner)

__init__ (*master*, *app*, *db*)
Inits the basic two parts of repocribro - flask app and DB

Parameters

- **master** (`ExtensionsMaster`) – Master for this extension
- **app** (`flask.Flask`) – Flask application of repocribro
- **db** (`flask_sqlalchemy.SQLAlchemy`) – SQLAlchemy database of repocribro
- **args** – not used
- **kwargs** – not used

call (*hook_name*, *default*, **args*, ***kwargs*)
Call the operation via hook name

Parameters

- **hook_name** (*str*) – Name of hook to be called
- **default** – Default return value if hook operation not found
- **args** – Positional args to be passed to the hook operation
- **kwargs** – Keywords args to be passed to the hook operation

Returns Result of the operation on the requested hook

init_blueprints()
Hook operation for initiating the blueprints and registering them within repocribro Flask app

init_filters()
Hook operation for initiating the Jinja filters and registering them within Jinja env of repocribro Flask app

init_models()
Hook operation for initiating the models and registering them within db

init_security()
Hook operation to setup privileges (roles and actions)

init_template_vars()

introduce()
Hook operation for getting short introduction of extension (mostly for debug/log purpose)

Returns Name of the extension

Return type str

static provide_actions()
Extension can define actions for privileges

Returns List of action names

Return type list of str

static provide_blueprints()
Extension can provide Flask blueprints to the app by this method

Returns List of Flask blueprints provided by extension

Return type list of `flask.blueprint`

static provide_dropdown_menu_items()

static provide_filters()
Extension can provide Jinja filters to the app by this method

Returns Dictionary with name + function/filter pairs

Return type dict of str: function

static provide_models()
Extension can provide (DB) models to the app by this method

Returns List of models provided by extension

Return type list of `db.Model`

static provide_roles()
Extension can define roles for user accounts

Returns Dictionary with name + Role entity

Return type dict of str: `repocribro.models.Role`

static provide_template_loader()

register_blueprints_from_list(*blueprints*)
Registering Flask blueprints to the app

Parameters **blueprints** (`list` of `flask.blueprint`) – List of Flask blueprints to be registered

register_filters_from_dict(*filters*)
Registering functions as Jinja filters

Parameters **filters** (*dict of str: function*) – Dictionary where key is name of filter and value is the function serving as filter

view_admin_extensions()
Hook operation for getting view model of the extension in order to show it in the administration of app

Returns Extensions view for this extension

Return type `repocribro.extending.helpers.ExtensionView`

ExtensionsMaster

class repocribro.extending.**ExtensionsMaster**(**args*, ***kwargs*)
Bases: `object`

Collector & master of Extensions

Extension master finds and holds all the **repocribro** extensions and is used for calling operations on them and collecting the results.

ENTRYPOINT_GROUP = 'repocribro.ext'
String used for looking up the extensions

LOAD_ERROR_MSG = 'Extension "{}" ({}) is not making an Extension (sub)class instance.
Error message mask for extension load error

__init__(**args*, ***kwargs*)
Collects all the extensions to be mantained by this object

Parameters

- **args** – positional args to be passed to extensions

- **kwargs** – keywords args to be passed to extensions

Todo: There might be some problem with ordering of extensions

classmethod _collect_extensions(*name=None*)
Method for selecting extensions within ENTRYPOINT_GROUP

Parameters **name** (*str*) – Can be used to select single entrypoint/extension

Returns Generator of selected entry points

Return type pkg_resources.WorkingSet.iter_entry_points

call(*hook_name*, *default=None*, **args*, ***kwargs*)
Call the hook on all extensions registered

Parameters

- **hook_name** (*str*) – Name of hook to be called
- **default** – Default return value if hook operation not found
- **args** – Positional args to be passed to the hook operation
- **kwargs** – Keywords args to be passed to the hook operation

Returns Result of the operation on the requested hook

repocribro.extending.helpers.views

repocribro.extending.helpers.views.ViewTab

class repocribro.extending.helpers.views.**ViewTab**(*id*, *name*, *priority=100*, *content=''*, *octicon=None*, *badge=None*)

Bases: object

Tab for the tabbed view at pages

__init__(*id*, *name*, *priority=100*, *content=''*, *octicon=None*, *badge=None*)
Initialize self. See help(type(self)) for accurate signature.

__lt__(*other*)
Return self<value.

repocribro.extending.helpers.views.Badge

class repocribro.extending.helpers.views.**Badge**(*content*)
Bases: object

Simple Twitter Bootstrap badge representation

__init__(*content*)
Initialize self. See help(type(self)) for accurate signature.

repocribro.extending.helpers.views.ExtensionView

class repocribro.extending.helpers.views.**ExtensionView**(*name*, *category*, *author*, *admin_url=None*, *home_url=None*, *gh_url=None*)

Bases: object

View object for extensions

__init__(*name*, *category*, *author*, *admin_url=None*, *home_url=None*, *gh_url=None*)
Initialize self. See help(type(self)) for accurate signature.

static from_class(*cls*)
Make view from Extension class

1.8.7 repocribro.github

GitHubAPI

class repocribro.github.**GitHubAPI**(*client_id*, *client_secret*, *webhooks_secret*, *session=None*, *token=None*)

Bases: object

Simple GitHub API communication wrapper

It provides simple way for getting the basic GitHub API resources and special methods for working with webhooks.

Todo: handle if GitHub is out of service, custom errors, better abstraction, work with extensions

API_URL = 'https://api.github.com'
URL to GitHub API

AUTH_URL = 'https://github.com/login/oauth/authorize?scope={}&client_id={}'
URL for OAuth request at GitHub

CONNECTIONS_URL = 'https://github.com/settings/connections/applications/{}'
URL for checking connections within GitHub

SCOPES = ['user', 'repo', 'admin:repo_hook']
Scopes for OAuth request

TOKEN_URL = 'https://github.com/login/oauth/access_token'
URL for OAuth token at GitHub

WEBHOOKS = ['push', 'release', 'repository']
Required webhooks to be registered

WEBHOOK_CONTROLLER = 'webhooks.gh_webhook'
Controller for incoming webhook events

__init__(*client_id*, *client_secret*, *webhooks_secret*, *session=None*, *token=None*)
Initialize self. See help(type(self)) for accurate signature.

_get_headers()
Prepare auth header fields (empty if no token provided)

Returns Headers for the request

Return type dict

app_connections_link

get(*what*, *page=0*)
Perform GET request on GitHub API

Parameters

- **what** (*str*) – URI of requested resource
- **page** (*int*) – Number of requested page

Returns Response from the GitHub

Return type repocribro.github.GitHubResponse

get_auth_url()
Create OAuth request URL

Returns OAuth request URL

Return type str

login(*session_code*)
Authorize via OAuth with given session code

Parameters **session_code** (*str*) – The session code for OAuth

Returns If the auth procedure was successful

Return type bool

Todo: check granted scope vs GH_SCOPES

---

webhook_create(*full_name*, *hook_url*, *events=None*)
Create new webhook for specified repository

Parameters

- **full_name** (*str*) – Full name of the repository
- **hook_url** (*str*) – URL where the webhook data will be sent
- **events** (*list of str*) – List of requested events for that webhook

Returns The created webhook data

Return type dict or None

webhook_delete(*full_name*, *hook_id*)
Perform DELETE request for repo's webhook

Parameters

- **full_name** (*str*) – Full name of repository that contains the hook
- **hook_id** (*int*) – GitHub ID of hook to be deleted

Returns If request was successful

Return type bool

webhook_get(*full_name*, *hook_id*)
Perform GET request for repo's webhook

Parameters

- **full_name** (*str*) – Full name of repository that contains the hook
- **hook_id** (*int*) – GitHub ID of hook to be get

Returns Data of the webhook

Return type repocribro.github.GitHubResponse

webhook_tests(*full_name*, *hook_id*)
Perform test request for repo's webhook

Parameters

- **full_name** (*str*) – Full name of repository that contains the hook
- **hook_id** (*int*) – GitHub ID of hook to be tested

Returns If request was successful

Return type bool

webhook_verify_signature(*data*, *signature*)
Verify the content with signature

Parameters

- **data** – Request data to be verified
- **signature** (*str*) – The signature of data

Returns If the content is verified

Return type bool

webhooks_get(*full_name*)
GET all webhooks of the repository

Parameters **full_name** (*str*) – Full name of repository

Returns List of returned webhooks

Return type repocribro.github.GitHubResponse

GitHubResponse

class repocribro.github.**GitHubResponse**(*response*)
Bases: object

Wrapper for GET request response from GitHub

__init__(*response*)
Initialize self. See help(type(self)) for accurate signature.

actual_page
Actual page number

Returns actual page number

Return type int

data
Response data as dict/list

Returns data of response

Return type dict|list

is_first_page
Check if this is the first page of data

Returns if it is the first page of data

Return type bool

is_last_page
Check if this is the last page of data

Returns if it is the last page of data

Return type bool

is_ok
Check if request has been successful

Returns if it was OK

Return type bool

is_only_page
Check if this is the only page of data

Returns if it is the only page page of data

Return type bool

links
Response header links

Returns URL origin

Return type dict

static parse_page_number(*url*)
Parse page number from GitHub GET URL

Parameters **url** (*str*) – URL used for GET request

Returns page number

Return type int

total_pages
Number of pages

Returns number of pages

Return type int

url
URL of the request leading to this response

Returns URL origin

Return type str

1.8.8 repocribro.models

Mixins

Anonymous

class repocribro.models.**Anonymous**

Bases: flask_login.mixins.AnonymousUserMixin, *repocribro.models.UserMixin*

Anonymous (not logged) user representation

_roles = []

has_role(*role*)

Check whether has the role

Parameters **role** (repocribro.models.RoleMixin) – Role to be checked

Returns False, anonymous has no roles

Return type bool

is_active

Check whether is user active

Returns False, anonymous is not active

Return type bool

owns_repo(*repo*)

Check if user owns the repository

Parameters **repo** (repocribro.models.Repository) – Repository which shoudl be tested

Returns False, anonymous can not own repository

Return type bool

rolename = 'anonymous'

rolenames

Get names of all roles of that user

Returns Empty list, anonymous has no roles

Return type list of str

roles

sees_repo(*repo*, *has_secret=False*)

Check if user is allowed to see the repo

Anonymous can see only public repos

Parameters

- **repo** (repocribro.models.Repository) – Repository which user want to see
- **has_secret** (*bool*) – If current user knows the secret URL

Returns If user can see repo

Return type bool

classmethod set_role(*role*)

RoleMixin

class repocribro.models.**RoleMixin**
Bases: object

Mixin for models representing roles

__eq__(*other*)
Equality of roles is based on names

Parameters **other** (repocribro.models.RoleMixin or str) – Role or its name to be compared with

Returns If names are equal

Return type bool

__hash__()
Standard hashing via name

Returns Hash of role

Return type int

__ne__(*other*)
Inequality of roles is based on names

Parameters **other** (repocribro.models.RoleMixin or str) – Role or its name to be compared with

Returns If names are not equal

Return type bool

permits(*privilege*)
Check if action priviledge is permitted in this role

Parameters **privilege** – privilege to be tested

Type str

Returns if it is permitted

Return type bool

priv_regex = re.compile('[a-z_\\?*]+')

valid_privileges()
Checks if privileges string is valid

Returns if privileges string is valid

Return type bool

SearchableMixin

class repocribro.models.**SearchableMixin**
Bases: object

Mixin for models that support fulltext query

classmethod fulltext_query(*query_str*, *db_query*)
Add fulltext filter to the DB query

Parameters

- **query_str** (*str*) – String to be queried
- **db_query** (sqlalchemy.orm.query.Query) – Database query object

Returns Query with fulltext filter added

Return type sqlalchemy.orm.query.Query

UserMixin

class repocribro.models.**UserMixin**

Bases: flask_login.mixins.UserMixin

has_role(*role*)

Check whether has the role

Parameters **role** (*str*) – Role to be checked

Returns If user has a role

Return type bool

is_active

Check whether is user active

Returns If user is active (can login)

Return type bool

owns_repo(*repo*)

Check if user owns the repository

Parameters **repo** (repocribro.models.Repository) – Repository which shoudl be tested

Returns If user owns repo

Return type bool

privileges(*all_privileges=frozenset()*)

Filter given privileges if are applicable for the user

Parameters **all_privileges** (*set of str*) – set of all privileges to be filtered

Returns set of applicable privileges

Return type set of str

rolenames

Get names of all roles of that user

Returns List of names of roles of user

Return type list of str

sees_repo(*repo*, *has_secret=False*)

Check if user is allowed to see the repo

Must be admin or owner to see not public repo

Parameters

- **repo** (repocribro.models.Repository) – Repository which user want to see
- **has_secret** (*bool*) – If current user knows the secret URL

Returns If user can see repo

Return type bool

Models

Commit

class repocribro.models.**Commit**(*sha*, *message*, *author_name*, *author_email*, *distinct*, *push*)

Bases: flask_sqlalchemy.Model, *repocribro.models.SearchableMixin*, repocribro.models.SerializableMixin

Commit from GitHub

__init__(*sha*, *message*, *author_name*, *author_email*, *distinct*, *push*)

__repr__()

Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'author_email': <sqlalchemy.orm.attributes.InstrumentedAttribute

author_email

The git author's email address.

author_name

The git author's name.

static create_from_dict(*commit_dict*, *push*)

Create new commit from GitHub and additional data

Parameters

- **commit_dict** (*dict*) – GitHub data containing commit
- **push** (repocribro.models.Push) – Push where this commit belongs

Returns Created new commit

Return type repocribro.models.Commit

Todo: verify, there are some conflict in GitHub docs

distinct

Whether this commit is distinct from any that have been pushed before.

id

Unique identifier of the commit

message

The commit message.

push

Push where the commit belongs to

push_id

ID of push where the commit belongs to

sha
The SHA of the commit.

Organization

class repocribro.models.**Organization**(*github_id*, *login*, *email*, *name*, *company*, *location*, *description*, *blog_url*, *avatar_url*)

Bases: repocribro.models.RepositoryOwner, *repocribro.models.SearchableMixin*, repocribro.models.SerializableMixin

Organization from GitHub

__init__(*github_id*, *login*, *email*, *name*, *company*, *location*, *description*, *blog_url*, *avatar_url*)

__repr__()
Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'avatar_url': <sqlalchemy.orm.attributes.InstrumentedAttribute o

avatar_url

blog_url

company

static create_from_dict(*org_dict*)
Create new organization from GitHub data

Parameters **org_dict** (*dict*) – GitHub data containing organization

Returns Created new organization

Return type repocribro.models.Organization

description

email

github_id

id

location

login

name

repositories

type

Push

class repocribro.models.**Push**(*github_id*, *ref*, *after*, *before*, *size*, *distinct_size*, *timestamp*, *sender_login*, *sender_id*, *repository*)

Bases: flask_sqlalchemy.Model, *repocribro.models.SearchableMixin*, repocribro.models.SerializableMixin

Push from GitHub

__init__(*github_id*, *ref*, *after*, *before*, *size*, *distinct_size*, *timestamp*, *sender_login*, *sender_id*, *repository*)

__repr__()
Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'after': <sqlalchemy.orm.attributes.InstrumentedAttribute object:

after
The SHA of the most recent commit on ref after the push. (HEAD)

before
The SHA of the most recent commit on ref before the push.

commits
Commits within this push

static create_from_dict(*push_dict*, *sender_dict*, *repo*, *timestamp=None*)
Create new push from GitHub and additional data

This also creates commits of this push

Parameters

- **push_dict** (*dict*) – GitHub data containing push
- **sender_dict** (*dict*) – GitHub data containing sender
- **repo** (repocribro.models.Repository) – Repository where this push belongs

Returns Created new push

Return type repocribro.models.Push

distinct_size
The number of distinct commits in the push.

github_id
GitHub Push ID

id
Unique identifier of the push

ref
The full Git ref that was pushed.

repository
Repository where push belongs to

repository_id
ID of the repository where push belongs to

sender_id
ID of the sender

sender_login
Login of the sender

size
The number of commits in the push.

timestamp
Timestamp of push (when it was registered)

Role

class repocribro.models.**Role**(*name*, *privileges*, *description*)

Bases: flask_sqlalchemy.Model, *repocribro.models.RoleMixin*

User account role in the application

__init__(*name*, *privileges*, *description*)

__repr__()

Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'description': <sqlalchemy.orm.attributes.InstrumentedAttribute

description

Description (purpose, notes, ...) of the role

id

Unique identifier of the role

name

Unique name of the role

privileges

Serialized list of privileges

user_accounts

User accounts assigned to the role

Release

class repocribro.models.**Release**(*github_id*, *tag_name*, *created_at*, *published_at*, *url*, *prerelease*, *draft*, *name*, *body*, *author_id*, *author_login*, *sender_login*, *sender_id*, *repository*)

Bases: flask_sqlalchemy.Model, *repocribro.models.SearchableMixin*, repocribro.models.SerializableMixin

Release from GitHub

__init__(*github_id*, *tag_name*, *created_at*, *published_at*, *url*, *prerelease*, *draft*, *name*, *body*, *author_id*, *author_login*, *sender_login*, *sender_id*, *repository*)

__repr__()

Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'author_id': <sqlalchemy.orm.attributes.InstrumentedAttribute ob

author_id

ID of author

author_login

Login of author

body

Body with some description

static create_from_dict(*release_dict*, *sender_dict*, *repo*)
Create new release from GitHub and additional data

Parameters

- **release_dict** (*dict*) – GitHub data containing release
- **sender_dict** (*dict*) – GitHub data containing sender
- **repo** (repocribro.models.Repository) – Repository where this release belongs

Returns Created new release

Return type repocribro.models.Release

created_at
Timestamp when the release was created

draft
Flag if it's just a draft

github_id
GitHub unique identifier

id
Unique identifier of the release

name
Name

prerelease
Flag if it's just a prerelease

published_at
Timestamp when the release was published

repository
Repository where release belongs to

repository_id
ID of the repository where release belongs to

sender_id
ID of sender

sender_login
Login of sender

tag_name
Tag of the release

url
URL to release page

Repository

class repocribro.models.**Repository**(*github_id*, *parent_name*, *full_name*, *name*, *languages*, *url*, *description*, *topics*, *private*, *webhook_id*, *owner*, *visibility_type*, *secret=None*)
Bases: flask_sqlalchemy.Model, *repocribro.models.SearchableMixin*, repocribro.models.SerializableMixin

Repository from GitHub

VISIBILITY_HIDDEN = 2
Constant representing hidden visibility within app

VISIBILITY_PRIVATE = 1
Constant representing private visibility within app

VISIBILITY_PUBLIC = 0
Constant representing public visibility within app

__init__(*github_id*, *parent_name*, *full_name*, *name*, *languages*, *url*, *description*, *topics*, *private*, *webhook_id*, *owner*, *visibility_type*, *secret=None*)

__repr__()
Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'description': <sqlalchemy.orm.attributes.InstrumentedAttribute

static create_from_dict(*repo_dict*, *owner*, *webhook_id=None*, *visibility_type=0*, *secret=None*)
Create new repository from GitHub and additional data

Parameters

- **repo_dict** (*dict*) – GitHub data containing repository
- **owner** (repocribro.model.RepositoryOwner) – Owner of this repository
- **webhook_id** (*int*) – ID of registered webhook (if available)
- **visibility_type** (*int*) – Visibility type within app (default: public)
- **secret** (*str*) – Secret for hidden URL (if available)

Returns Created new repository

Return type repocribro.models.Repository

description

events_updated()
Set that now was performed last events update of repo

Todo: How about some past events before adding to app?

full_name
Full name (owner login + repository name)

generate_secret()
Generate new unique secret code for repository

github_id
GitHub unique identifier

id
Unique identifier of the repository

is_hidden
Check if repository is hidden within app

is_private
Check if repository is private within app

is_public
Check if repository is public within app

languages

last_event

static make_full_name(*login*, *reponame*)
Create full name from owner login name and repository name

Parameters

- **login** (*str*) – Owner login
- **reponame** (*str*) – Name of repository (without owner login)

Returns Full name of repository

Return type str

members
Members of org repo within app

name

owner
Owner of repository

owner_id

owner_login
Get owner login from full name of repository

Returns Owner login

Return type str

parent_name
Full name of repository which this is fork of

private

pushes
Registered pushes to repository

releases
Registered releases for repository

secret

static serialize_topics(*topics*)
Make string from topics list from GitHub

Parameters **topics** (*list of str*) – List of topics (strings without whitespaces)

Returns Serialized list of topics

Return type str

topics

update_from_dict(*repo_dict*)
Update repository attributes from GitHub data dict

Parameters **repo_dict** (*dict*) – GitHub data containing repository

update_languages(*languages_dict*)
Set languages field from GitHub dict

Parameters **languages_dict** (*dict*) – language - bytes dict

url

visibility_type

webhook_id

User

class repocribro.models.**User**(*github_id*, *login*, *email*, *name*, *company*, *location*, *bio*, *blog_url*, *avatar_url*, *hireable*, *user_account*)
Bases: repocribro.models.RepositoryOwner, *repocribro.models.SearchableMixin*, repocribro.models.SerializableMixin

User from GitHub

__init__(*github_id*, *login*, *email*, *name*, *company*, *location*, *bio*, *blog_url*, *avatar_url*, *hireable*, *user_account*)

__repr__()
Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'avatar_url': <sqlalchemy.orm.attributes.InstrumentedAttribute ob

avatar_url

blog_url

company

static create_from_dict(*user_dict*, *user_account*)
Create new user from GitHub data and related user account

Parameters

- **user_dict** (*dict*) – GitHub data containing user
- **user_account** (repocribro.models.UserAccount) – User account in app for GH user

Returns Created new user

Return type repocribro.models.User

description

email

github_id

hireable
Flag whether is user hireable

id

location

login

name

org_repositories
Members of org repo within app

repositories

type

update_from_dict(*user_dict*)
Update user from GitHub data

Parameters **user_dict** (*dict*) – GitHub data containing user

user_account
User's account within app

user_account_id
ID of user's account within app

UserAccount

class repocribro.models.**UserAccount**(***kwargs*)
Bases: flask_sqlalchemy.Model, *repocribro.models.UserMixin*, *repocribro.models.SearchableMixin*

UserAccount in the repocribro app

__init__(***kwargs*)
A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

__repr__()
Standard string representation of DB object

Returns Unique string representation

Return type str

_sa_class_manager = {'active': <sqlalchemy.orm.attributes.InstrumentedAttribute objec

active
Flag if the account is active or banned

created_at
Timestamp where account was created

default_rolename = 'user'

github_user
Relation to the GitHub user connected to account

id
Unique identifier of the user account

login
Get login name for user account from related GH user

Returns Login name

Return type str

roles
Roles assigned to the user account

1.8.9 repocribro.repocribro

repocribro.repocribro.**AUTHOR = 'Marek Suchánek'**
Author of the application

repocribro.repocribro.**DEFAULT_CONFIG_FILES = ['config/app.cfg', 'config/auth.cfg', 'config/**
Paths to default configuration files

class repocribro.repocribro.**DI_Container**
Simple container of services for web app

Variables

- **factories** – Factories of services
- **singletons** – Singletons (shared objects) of services

__init__()
Prepare dict for storing services and factories

get(*what*, **args*, ***kwargs*)
Retrieve service from the container

Parameters

- **what** (*str*) – Name of the service to get
- **args** – Positional arguments passed to factory
- **kwargs** – Keyword arguments passed to factory

Returns The service or None

set_factory(*name*, *factory*)
Set service factory (callable for creating instances)

Parameters

- **name** (*str*) – Name of the service
- **factory** (*callable*) – Function or callable object creating service instance

set_singleton(*name*, *singleton*)
Set service as singleton (shared object)

Parameters

- **name** (*str*) – Name of the service
- **singleton** (*object*) – The object to be shared as singleton

repocribro.repocribro.**PROG_NAME = 'repocribro'**
Name of the application

repocribro.repocribro.**RELEASE = '0.1'**
Actual release tag

class repocribro.repocribro.**Repocribro**
Repocribro is Flask web application

Variables **container** – Service container for the app

__init__()
Setup Flask app and prepare service container

ext_call(*what_to_call*)
Call hook on all extensions

Parameters **what_to_call** (*str*) – name of hook to call

Returns result of the call

repocribro.repocribro.**VERSION = '0.1'**
Actual version

repocribro.repocribro.**create_app**(*cfg_files=['DEFAULT']*)
Factory for making the web Flask application

Parameters **cfg_files** – Single or more config file(s)

Returns Constructed web application

Return type repocribro.repocribro.Repocribro

1.8.10 repocribro.security

class repocribro.security.**Permissions**
Class for prividing various permissions

__dict__ = mappingproxy({'__module__': 'repocribro.security', '__doc__': 'Class for

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'repocribro.security'

__weakref__
list of weak references to the object (if defined)

all_actions
All registered action privileges

Returns set of str

all_roles
All registered roles

Returns set of str

register_action(*priv_name*)
Register new action privilege by name

Parameters **priv_name** (*str*) – name of action privilege to register

register_role(*role_name*)
Register new role by name

Parameters **role_name** (*str*) – name of role to register

class repocribro.security.**PermissionsContainer**(*name*)
Container for permission to be used for decorators

__dict__ = mappingproxy({'__module__': 'repocribro.security', '__doc__': 'Container

__getattr__(*key*)

__init__(*name*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'repocribro.security'

__weakref__
list of weak references to the object (if defined)

repocribro.security.**clear_session**(**args*)
Simple helper for clearing variables from session

Parameters **args** – names of session variables to remove

repocribro.security.**create_default_role**(*app*, *db*, *role*)
Create default role for the app

Parameters

- **app** (repocribro.repocribro.Repocribro) – Current flask application
- **db** (flask_sqlalchemy.SQLAlchemy) – Database connection
- **role** (repocribro.models.Role) – Role to be created

repocribro.security.**get_default_user_role**(*app*, *db*)
Get special default role for registered users

Parameters

- **app** (repocribro.repocribro.Repocribro) – Current flask application
- **db** (flask_sqlalchemy.SQLAlchemy) – Database connection

repocribro.security.**init_login_manager**(*db*)
Init security extensions (login manager and principal)

Parameters **db** (flask_sqlalchemy.SQLAlchemy) – Database which stores user accounts and roles

Returns Login manager and principal extensions

Return type (flask_login.LoginManager, flask_principal.Principal

repocribro.security.**login**(*user_account*)
Login desired user into the app

Parameters **user_account** (repocribro.models.UserAccount) – User account to be logged in

repocribro.security.**logout**()
Logout the current user from the app

repocribro.security.**on_identity_loaded**(*sender*, *identity*)
Principal helper for loading the identity of logged user

Parameters

- **sender** – Sender of the signal
- **identity** (flask_principal.Identity) – Identity container

repocribro.security.**permissions = <repocribro.security.Permissions object>**
All permissions in the app

repocribro.security.**reload_anonymous_role**(*app*, *db*)
Reload special role for anonymous users

Parameters

- **app** (repocribro.repocribro.Repocribro) – Current flask application
- **db** (flask_sqlalchemy.SQLAlchemy) – Database connection

CHAPTER 2

Indices and tables

- genindex
- search

Python Module Index

r

Index

Symbols

A

B

C

D

E

F

G

H

I

L

R

S

T

U

V

W