FlaskBB Documentation

Release 2.0.2

Peter Justin

Mar 20, 2019

Contents

FlaskBB is a classic forum software in a modern and fresh look. It is written in Python using the web framework Flask. FlaskBB is being distributed under the BSD 3-Clause License.

CHAPTER 1

Links

website | documentation | source code

1.1 User Documentation

1.1.1 Installation

- *Basic Setup*
- *Configuration*
- *Deploying*
- *Deploying to PythonAnywhere*

Basic Setup

We recommend installing FlaskBB in an isolated Python environment. This can be achieved with virtualenv. In our little guide we will use a wrapper around virtualenv - the virtualenvwrapper. In addition to virtualenv, we will also use the package manager pip to install the dependencies for FlaskBB.

Virtualenv Setup

Linux: The easiest way to install virtualenv and virtualenvwrapper is, to use the package manager on your system (if you are running Linux) to install them.

Windows: Take a look at the flask documentation (then skip ahead to dependencies).

For example, on archlinux you can install them with:

```
$ sudo pacman -S python2-virtualenvwrapper
```

or, on macOS, you can install them with:

```
$ sudo pip install virtualenvwrapper
```

It's sufficient to just install the virtualenvwrapper because it depends on virtualenv and the package manager will resolve all the dependencies for you.

After that, you can create your virtualenv with:

```
$ mkvirtualenv -a /path/to/flaskbb -p $(which python2) flaskbb
```

This will create a virtualenv named `flaskbb` using the python interpreter in version 2 and it will set your project directory to `/path/to/flaskbb`. This comes handy when typing `workon flaskbb` as it will change your current directory automatically to `/path/to/flaskbb`. To deactivate it you just have to type `deactivate` and if you want to work on it again, just type `workon flaskbb`.

It is also possible to use `virtualenv` without the `virtualenvwrapper`. For this you have to use the `virtualenv` command and pass the name of the virtualenv as an argument. In our example, the name of the virtualenv is `.venv`.

```
$ virtualenv .venv
```

and finally activate it

```
$ source .venv/bin/activate
```

If you want to know more about those isolated python environments, checkout the virtualenv and virtualenvwrapper docs.

Dependencies

Now that you have set up your environment, you are ready to install the dependencies.

```
$ pip install -r requirements.txt
```

Alternatively, you can use the *make* command to install the dependencies.

```
$ make dependencies
```

The development process requires a few extra dependencies which can be installed with the provided `requirements-dev.txt` file.

```
$ pip install -r requirements-dev.txt
```

Optional Dependencies

We have one optional dependency, redis (the python package is installed automatically). If you want to use it, make sure that a redis-server is running. Redis will be used as the default result and caching backend for celery (celery is a task queue which FlaskBB uses to send non blocking emails). The feature for tracking the *online guests* and *online users* do also require redis (although *online users* works without redis as well). To install redis, just use your distributions package manager. For Arch Linux this is *pacman* and for Debian/Ubuntu based systems this is *apt-get*.

```
# Installing redis using 'pacman':
$ sudo pacman -S redis
# Installing redis using 'apt-get':
$ sudo apt-get install redis-server

# Check if redis is already running.
$ systemctl status redis

# If not, start it.
$ sudo systemctl start redis

# Optional: Lets start redis everytime you boot your machine
$ sudo systemctl enable redis
```

Configuration

Production

FlaskBB already sets some sane defaults, so you shouldn't have to change much. To make this whole process a little bit easier for you, we have created a little wizard which will ask you some questions and with the answers you provide it will generate a configuration for you. You can of course further adjust the generated configuration.

The setup wizard can be started with:

```
flaskbb makeconfig
```

These are the only settings you have to make sure to setup accordingly if you want to run FlaskBB in production:

- `SERVER_NAME = "example.org"`
- `PREFERRED_URL_SCHEME = "https"`
- `SQLALCHEMY_DATABASE_URI = 'sqlite:///path/to/flaskbb.sqlite'`
- `SECRET_KEY = "secret key"`
- `WTF_CSRF_SECRET_KEY = "secret key"`

By default it will try to save the configuration file with the name flaskbb.cfg in FlaskBB's root folder.

Finally to get going – fire up FlaskBB!

```
flaskbb --config flaskbb.cfg run

[+] Using config from: /path/to/flaskbb/flaskbb.cfg
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Development

To get started with development you have to generate a development configuration first. You can use the CLI for this, as explained in *Configuration*:

```
flaskbb makeconfig -d
```

or

flaskbb makeconfig –development

Now you can either use `make` to run the development server:

```
make run
```

or if you like to type a little bit more, the CLI:

```
flaskbb --config flaskbb.cfg run
```

You can either pass an import string to the path to the (python) config file you've just created, or a default config object. (Most users will follow the example above, which uses the generated file). This is how you do it by using an import string. Be sure that it is importable from within FlaskBB:

> flaskbb –config flaskbb.configs.default.DefaultConfig run

Redis

If you have decided to use redis as well, which we highly recommend, then the following services and features can be enabled and configured to use redis.

Before you can start using redis, you have to enable and configure it. This is quite easy just set `REDIS_ENABLE` to `True` and adjust the `REDIS_URL` if needed.

```
REDIS_ENABLED = True
REDIS_URL = "redis://localhost:6379"  # or with a password: "redis://
↪:password@localhost:6379"
REDIS_DATABASE = 0
```

The other services are already configured to use the `REDIS_URL` configuration variable.

Celery

```
CELERY_BROKER_URL = REDIS_URL
CELERY_RESULT_BACKEND = REDIS_URL
```

Caching

```
CACHE_TYPE = "redis"
CACHE_REDIS_URL = REDIS_URL
```

Rate Limiting

```
RATELIMIT_ENABLED = True
RATELIMIT_STORAGE_URL = REDIS_URL
```

Mail Examples

Both methods are included in the example configs.

Google Mail

```
MAIL_SERVER = "smtp.gmail.com"
MAIL_PORT = 465
MAIL_USE_SSL = True
MAIL_USERNAME = "your_username@gmail.com"
MAIL_PASSWORD = "your_password"
MAIL_DEFAULT_SENDER = ("Your Name", "your_username@gmail.com")
```

Local SMTP Server

```
MAIL_SERVER = "localhost"
MAIL_PORT = 25
MAIL_USE_SSL = False
MAIL_USERNAME = ""
MAIL_PASSWORD = ""
MAIL_DEFAULT_SENDER = "noreply@example.org"
```

Installation

MySQL users: Make sure that you create the database using the `utf8` charset:

```
CREATE DATABASE flaskbb CHARACTER SET utf8;
```

Even though the `utf8mb4` charset is prefered today (see this SO answer), we have to create our database using the `utf8` charset. A good explanation about this issue can be found here.

For a guided install, run:

```
$ make install
```

or:

```
flaskbb install
```

During the installation process you are asked about your username, your email address and the password for your administrator user. Using the `make install` command is recommended as it checks that the dependencies are also installed.

Upgrading

If the database models changed after a release, you have to run the `upgrade` command:

```
flaskbb db upgrade
```

Deploying

This chapter will describe how to set up Supervisor + uWSGI + nginx for FlaskBB as well as document how to use the built-in WSGI server (gunicorn) that can be used in a productive environment.

Supervisor

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

To install *supervisor* on Debian, you need to fire up this command:

```
$ sudo apt-get install supervisor
```

There are two ways to configure supervisor. The first one is, you just put the configuration to the end in the `/etc/supervisor/supervisord.conf` file.

The second way would be to create a new file in the `/etc/supervisor/conf.d/` directory. For example, such a file could be named `uwsgi.conf`.

After you have choosen the you way you like, simply put the snippet below in the configuration file.

```
[program:uwsgi]
command=/usr/bin/uwsgi --emperor /etc/uwsgi/apps-enabled
user=apps
stopsignal=QUIT
autostart=true
autorestart=true
redirect_stderr=true
```

uWSGI

uWSGI is a web application solution with batteries included.

To get started with uWSGI, you need to install it first. You'll also need the python plugin to serve python apps. This can be done with:

```
$ sudo apt-get install uwsgi uwsgi-plugin-python
```

For the configuration, you need to create a file in the `/etc/uwsgi/apps-available` directory. In this example, I will call the file `flaskbb.ini`. After that, you can start with configuring it. My config looks like this for *flaskbb.org* (see below). As you might have noticed, I'm using a own user for my apps whose home directory is located at */var/apps/*. In this directory there are living all my Flask apps.

```
[uwsgi]
base = /var/apps/flaskbb
home = /var/apps/.virtualenvs/flaskbb/
pythonpath = %(base)
socket = 127.0.0.1:30002
module = wsgi
callable = flaskbb
uid = apps
gid = apps
logto = /var/apps/flaskbb/logs/uwsgi.log
plugins = python
```

base	/path/to/flaskbb	The folder where your flaskbb application lives
home	/path/to/virtualenv/folder	The virtualenv folder for your flaskbb application
python-path	/path/to/flaskbb	The same as base
socket	socket	This can be either a ip or the path to a socket (don't forget to change that in your nginx config)
module	wsgi.py	This is the file located in the root directory from flaskbb (where manage.py lives).
callable	flaskbb	The callable is application you have created in the `wsgi.py` file
uid	your_user	The user who should be used. **NEVER** use root!
gid	your_group	The group who should be used.
logto	/path/to/log/file	The path to your uwsgi logfile
plugins	python	We need the python plugin

Don't forget to create a symlink to `/etc/uwsgi/apps-enabled`.

```
ln -s /etc/uwsgi/apps-available/flaskbb /etc/uwsgi/apps-enabled/flaskbb
```

gunicorn

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX.

It's a pre-fork worker model ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

This is probably the easiest way to run a FlaskBB instance. Just install gunicorn via pip inside your virtualenv:

```
pip install gunicorn
```

FlaskBB has an built-in command to gunicorn:

```
flaskbb start
```

To see a full list of options either type `flaskbb start --help` or visit the *cli* docs.

nginx

nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server, written by Igor Sysoev.

The nginx config is pretty straightforward. Again, this is how I use it for *FlaskBB*. Just copy the snippet below and paste it to, for example `/etc/nginx/sites-available/flaskbb`. The only thing left is, that you need to adjust the `server_name` to your domain and the paths in `access_log`, `error_log`. Also, don't forget to adjust the paths in the `alias` es, as well as the socket address in `uwsgi_pass`.

```
server {
    listen 80;
    server_name forums.flaskbb.org;

    access_log /var/log/nginx/access.forums.flaskbb.log;
    error_log /var/log/nginx/error.forums.flaskbb.log;

    location / {
        try_files $uri @flaskbb;
    }

    # Static files
    location /static {
       alias /var/apps/flaskbb/flaskbb/static/;
    }

    location ~ ^/_themes/([^/]+)/(.*)$ {
        alias /var/apps/flaskbb/flaskbb/themes/$1/static/$2;
    }

    # robots.txt
    location /robots.txt {
        alias /var/apps/flaskbb/flaskbb/static/robots.txt;
    }

    location @flaskbb {
        uwsgi_pass 127.0.0.1:30002;
```

(continues on next page)

(continued from previous page)

```
        include uwsgi_params;
    }
}
```

If you wish to use gunicorn instead of uwsgi just replace the `location @flaskbb` with this:

```
location @flaskbb {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header Host $http_host;
    #proxy_set_header SCRIPT_NAME /forums;  # This line will make flaskbb available␣
↪on /forums;
    proxy_redirect off;
    proxy_buffering off;

    proxy_pass http://127.0.0.1:8000;
}
```

Don't forget to adjust the `proxy_pass` address to your socket address.

Like in the *uWSGI* chapter, don't forget to create a symlink to `/etc/nginx/sites-enabled/`.

User Contributed Deployment Guides

We do not maintain these deployment guides. They have been submitted by users and we thought it is nice to include them in docs. If something is missing, or doesn't work - please open a new pull request on GitHub.

Deploying to PythonAnywhere

PythonAnywhere is a platform-as-a-service, which basically means they have a bunch of servers pre-configured with Python, nginx and uWSGI. You can run a low-traffic website with them for free, so it's an easy way to get quickly FlaskBB running publicly.

Here's what to do:

- Sign up for a PythonAnywhere account at https://www.pythonanywhere.com/.
- On the "Consoles" tab, start a Bash console and install/configure FlaskBB like this

```
git clone https://github.com/sh4nks/flaskbb.git
cd flaskbb
pip3.5 install --user -r requirements.txt
pip3.5 install --user -e .
flaskbb makeconfig
flaskbb install
```

- Click the PythonAnywhere logo to go back to the dashboard, then go to the "Web" tab, and click the "Add a new web app" button.
- Just click "Next" on the first page.
- On the next page, click "Flask"
- On the next page, click "Python 3.5"
- On the next page, just accept the default and click next
- Wait while the website is created.

- Click on the "Source code" link, and in the input that appears, replace the *mysite* at the end with *flaskbb*
- Click on the "WSGI configuration file" filename, and wait for an editor to load.
- Change the line that sets *project_home* to replace *mysite* with *flaskbb* again.
- Change the line that says

```
from flask_app import app as application
```

to say

```
from flaskbb import create_app
application = create_app("/path/to/your/configuration/file")
```

- Click the green "Save" button near the top right.
- Go back to the "Web" tab.
- Click the green "Reload…" button.
- Click the link to visit the site – you'll have a new FlaskBB install!

1.1.2 Command Line Interface

Here you can find the documentation about FlaskBB's Command Line Interface.

To get help for a commands, just type `flaskbb COMMAND --help`. If no command options or arguments are used it will display all available commands.

```
Usage: flaskbb [OPTIONS] COMMAND [ARGS]...

  This is the commandline interface for flaskbb.

Options:
  --config CONFIG  Specify the config to use in dotted module notation e.g.
                   flaskbb.configs.default.DefaultConfig
  --version        Show the FlaskBB version.
  --help           Show this message and exit.

Commands:
  celery           Preconfigured wrapper around the 'celery' command.
  db               Perform database migrations.
  download-emojis  Downloads emojis from emoji-cheat-sheet.com.
  install          Installs flaskbb.
  makeconfig       Generates a FlaskBB configuration file.
  plugins          Plugins command sub group.
  populate         Creates the necessary tables and groups for FlaskBB.
  reindex          Reindexes the search index.
  run              Runs a development server.
  shell            Runs a shell in the app context.
  start            Starts a production ready wsgi server.
  themes           Themes command sub group.
  translations     Translations command sub group.
  upgrade          Updates the migrations and fixtures.
  urls             Show routes for the app.
  users            Create, update or delete users.
```

Commands

Here you will find a detailed description of every command including all of their options and arguments.

`flaskbb install`
Installs flaskbb. If no arguments are used, an interactive setup will be run.

`--welcome, -w`
Disables the generation of the welcome forum.

`--force, -f`
Doesn't ask for confirmation if the database should be deleted or not.

`--username USERNAME, -u USERNAME`
The username of the user.

`--email EMAIL, -e EMAIL`
The email address of the user.

`--password PASSWORD, -p PASSWORD`
The password of the user.

`--group GROUP, -g GROUP`
The primary group of the user. The group `GROUP` has to be one of `admin`, `super_mod`, `mod` or `member`.

`flaskbb upgrade`
Updates the migrations and fixtures.

`--all, -a`
Upgrades migrations AND fixtures to the latest version.

`--fixture FIXTURE, -f FIXTURE`
The fixture which should be upgraded or installed. All fixtures have to be places inside flaskbb/fixtures/

`--force-fixture, -ff`
Forcefully upgrades the fixtures. WARNING: This will also overwrite any settings.

`flaskbb populate`
Creates the necessary tables and groups for FlaskBB.

`--test-data, -t`
Adds some test data.

`--bulk-data, -b`
Adds a lot of test data. Has to be used in combination with `--posts` and `--topics`.

`--posts`
Number of posts to create in each topic (default: 100).

`--topics`
Number of topics to create (default: 100).

`--force, -f`
Will delete the database without asking before populating it.

`--initdb, -i`
Initializes the database before populating it.

`flaskbb runserver`
Starts the development server

`flaskbb start`
Starts a production ready wsgi server. Other versions of starting FlaskBB are still supported!

`--server SERVER, -s SERVER`

Defaults to `gunicorn`. The following WSGI Servers are supported:

- gunicorn (default)
- gevent

`--host HOST, -h HOST`
The interface to bind FlaskBB to. Defaults to `127.0.0.1`.

`--port PORT, -p PORT`
The port to bind FlaskBB to. Defaults to `8000`.

`--workers WORKERS, -w WORKERS`
The number of worker processes for handling requests. Defaults to `4`.

`--daemon, -d`
Starts gunicorn in daemon mode.

`--config, -c`
The configuration file to use for the FlaskBB WSGI Application.

`flaskbb celery CELERY_ARGS`
Starts celery. This is just a preconfigured wrapper around the `celery` command. Additional arguments are directly passed to celery.

`--help-celery`
Shows the celery help message.

`flaskbb shell`
Creates a python shell with an app context.

`flaskbb urls`
Lists all available routes.

`--route, -r`
Order by route.

`--endpoint, -e`
Order by endpoint

`--methods, -m`
Order by methods

`flaskbb makeconfig`
Generates a FlaskBB configuration file.

`--development, -d`
Creates a development config with DEBUG set to True.

`--output, -o`
The path where the config file will be saved at. Defaults to the flaskbb's root folder.

`--force, -f`
Overwrites any existing config file, if one exsits, WITHOUT asking.

`flaskbb reindex`
Reindexes the search index.

`flaskbb translations`
Translations command sub group.

`new LANGUAGE_CODE`
Adds a new language to FlaskBB's translations. The `LANGUAGE_CODE` is the short identifier for the language i.e. '`en`', '`de`', '`de_AT`', etc.

`--plugin PLUGIN_NAME, --p PLUGIN_NAME`
Adds a new language to a plugin.

`update`
Updates the translations.

`--all, -a`
Updates all translations, including the ones from the plugins.

`--plugin PLUGIN_NAME, --p PLUGIN_NAME`
Update the language of the given plugin.

`compile`
Compiles the translations.

`--all, -a`
Compiles all translations, including the ones from the plugins.

`--plugin PLUGIN_NAME, --p PLUGIN_NAME`
Compiles only the given plugin translation.

`flaskbb plugins`
Plugins command sub group.

`new PLUGIN_IDENTIFIER`
Creates a new plugin based on the cookiecutter plugin template. Defaults to this template: https://github.com/sh4nks/cookiecutter-flaskbb-plugin. It will either accept a valid path on the filesystem or a URL to a Git repository which contains the cookiecutter template.

`install PLUGIN_IDENTIFIER`
Installs a plugin by using the plugin's identifier.

`uninstall PLUGIN_IDENTIFIER`
Uninstalls a plugin by using the plugin's identifier.

`remove PLUGIN_IDENTIFIER`
Removes a plugin from the filesystem by using the plugin's identifier.

describe:: –force, -f

Removes the plugin without asking for confirmation first.

`list`
Lists all installed plugins.

`flaskbb themes`
Themes command sub group.

`new THEME_IDENTIFIER`
Creates a new theme based on the cookiecutter theme template. Defaults to this template: https://github.com/sh4nks/cookiecutter-flaskbb-theme. It will either accept a valid path on the filesystem or a URL to a Git repository which contains the cookiecutter template.

`remove THEME_IDENTIFIER`
Removes a theme from the filesystem by the theme's identifier.

`list`
Lists all installed themes.

`flaskbb users`

Creates a new user. If an option is missing, you will be interactivly prompted to type it.

`new`

Creates a new user.

`--username USERNAME, -u USERNAME`

The username of the user.

`--email EMAIL, -e EMAIL`

The email address of the user.

`--password PASSWORD, -p PASSWORD`

The password of the user.

`--group GROUP, -g GROUP`

The primary group of the user. The group `GROUP` has to be one of `admin`, `super_mod`, `mod` or `member`.

`update`

Updates an user.

`--username USERNAME, -u USERNAME`

The username of the user.

`--email EMAIL, -e EMAIL`

The email address of the user.

`--password PASSWORD, -p PASSWORD`

The password of the user.

`--group GROUP, -g GROUP`

The primary group of the user. The group `GROUP` has to be one of `admin`, `super_mod`, `mod` or `member`.

`delete`

`--username USERNAME, -u USERNAME`

The username of the user.

`--force, -f`

Removes the user without asking for confirmation first.

1.1.3 Plugins

FlaskBB provides a full featured plugin system. This system allows you to easily extend or modify FlaskBB without touching any FlaskBB code. Under the hood it uses the pluggy plugin system which does most of the heavy lifting for us. A list of available plugins can be found at the GitHub Wiki. A proper index for FlaskBB Plugins and Themes still have to be built.

If you are interested in creating new plugins, checkout out the *Developing new Plugins* page.

Management

Before plugins can be used in FlaskBB, they have to be downloaded, installed and activated. Plugins can be very minimalistic with nothing to install at all (just enabling and disabling) to be very complex where you have to run migrations and add some additional settings.

Download

Downloading a Plugin is as easy as:

```
$ pip install flaskbb-plugin-MYPLUGIN
```

if the plugin has been uploaded to PyPI. If you haven't uploaded your plugin to PyPI or are in the middle of developing one, you can just:

```
$ pip install -e .
```

in your plugin's package directory to install it.

Remove

Removing a plugin is a little bit more tricky. By default, FlaskBB does not remove the settings of a plugin by itself because this could lead to some unwanted dataloss.

Disable and *Uninstall* the plugin first before continuing.

After taking care of this and you are confident that you won't need the plugin anymore you can finally remove it:

```
$ pip uninstall flaskbb-plugin-MYPLUGIN
```

There is a setting in FlaskBB which lets you control the deletion of settings of a plugin. If `REMOVE_DEAD_PLUGINS` is set to `True`, all not available plugins (not available on the filesystem) are constantly removed. Only change this if you know what you are doing.

Install

In our context, by installing a plugin, we mean, to install the settings and apply the migrations. Personal Note: I can't think of a better name and I am open for suggestions.

The plugin can be installed via the Admin Panel (in tab 'Plugins') or by running:

```
flaskbb plugins install <plugin_name>
```

Make sure to to apply the migrations of the plugin as well (**if any**, check the plugins docs):

```
flaskbb db upgrade <plugin_name>@head
```

Uninstall

Removing a plugin involves two steps. The first one is to check if the plugin has applied any migrations on FlaskBB and if so you can undo them via:

```
$ flaskbb db downgrade <plugin_name>@base
```

The second step is to wipe the settings from FlaskBB which can be done in the Admin Panel or by running:

```
$ flaskbb plugins uninstall <plugin_name>
```

Disable

Disabling a plugin has the benefit of keeping all the data of the plugin but not using the functionality it provides. A plugin can either be deactivated via the Admin Panel or by running:

```
flaskbb plugins disable <plugin_name>
```

Important: Restart the server.

You must restart the wsgi/in-built server in order to make the changes effect your forum.

Enable

All plugins are activated by default. To activate a deactivated plugin you either have to activate it via the Admin Panel again or by running the activation command:

```
flaskbb plugins enable <plugin_name>
```

Database

Upgrading, downgrading and generating database revisions is all handled via alembic. We make use of alembic's branching feature to manage seperate migrations for the plugins. Each plugin will have it's own branch in alembic where migrations can be managed. Following commands are used for generaring, upgrading and downgrading your plugins database migrations:

- **(Auto-)Generating revisions** `flaskbb db revision --branch <plugin_name> "<YOUR_MESSAGE>"`

  Replace <YOUR_MESSAGE> with something like "initial migration" if it's the first migration or with just a few words that will describe the changes of the revision.

- **Applying revisions** `flaskbb db upgrade <plugin_name>@head`

  If you want to upgrade to specific revision, replace `head` with the revision id.

- **Downgrading revisions** `flaskbb db downgrade <plugin_name>@-1`

  If you just want to revert the latest revision, just use `-1`. To downgrade all database migrations, use `base`.

1.1.4 FAQ - Frequently Asked Questions

Here we try to answer some common questions and pitfalls about FlaskBB.

- Why do I get a `AttributeError: 'NullTranslations' object has no attribute 'add'` exception?

  This usually happens when you forgot to compile the translations. To compile them, just run:

  ```
  $ flaskbb translations compile
  ```

  Relevant issue: #389

- Why isn't the cache (Flask-Caching) using the configured `REDIS_URL`?

  You have to set the `CACHE_REDIS_HOST` to the `REDIS_URL`. This is inconvenience is caused because you are not limited to redis as the caching backend. See the Flask-Caching documentation for a full list of caching backends.

  Relevant issue: #372

1.2 Developer Documentation

1.2.1 Theming

FlaskBB uses the Flask-Themes2 extension for theming.

Quickstart

1. Create a new folder within the `themes/` folder and give it the name of your theme.
2. Copy the content of the `aurora/` folder into your folder theme's folder.
3. Create **2** new folders called `static/` and `templates/` in your themes folder.
4. Copy `layout.html` from FlaskBB's `templates/` into your themes `templates/` folder and modified to your liking. Feel free to copy other templates over into your themes. Just make sure that they have the same name and directory structure to overwrite them.
5. Add some information about your theme using the `info.json` file.
6. Edit the `package.json` to your needs.
7. Happy theming!

In the end your folder structure should look like this:

```
── example_theme/
    ├── node_modules
    │   └── ...
    ├── src
    │   ├── img
    │   │   └── ...
    │   ├── js
    │   │   └── ...
    │   └── scss
    │       └── ...
    ├── static
    │   ├── img
    │   ├── css
    │   ├── fonts
    │   └── js
    ├── templates
    │   ├── ...
    │   └── layout.html
    ├── tools
    │   ├── build_css
    │   ├── build_fonts
    │   └── build_js
    ├── info.json
```

(continues on next page)

(continued from previous page)

```
    ├── LICENSE
    ├── package.json
    └── README.md
```

Getting Started

A theme is simply a folder containing static media (like CSS files, images, and JavaScript) and Jinja2 templates, with some metadata. A theme folder should look something like this:

```
my_theme
├── info.json
├── LICENSE
├── static
│   └── style.css
└── templates
    └── layout.html
```

Every theme needs to have a file called **info.json**. The info.json file contains the theme's metadata, so that the application can provide a nice switching interface if necessary. For example, the info.json file for the aurora theme looks like this:

```
{
    "application": "flaskbb",
    "identifier": "aurora",
    "name": "Aurora",
    "author": "Peter Justin",
    "license": "BSD 3-Clause",
    "website": "https://flaskbb.org",
    "description": "The default theme for FlaskBB.",
    "preview": "preview.png",
    "version": "1.0.0"
}
```

Field Explanation

application The name of the application, in our case this should always be **flaskbb**.

identifier The unique name of your theme. This identifier should match the themes folder name!

name Human readable name of the theme.

author The name of the author.

license A short phrase describing the license, like "GPL", "BSD", "Public Domain", or "Creative Commons BY-SA 3.0". Every theme should define a license under which terms the theme can be used. You should also put a copy of the license in your themes directory (e.g. in a LICENSE file).

description A short description about your theme. For example: "A minimalistic blue theme".

website The URL of the theme's Web site. This can be a Web site specifically for this theme, Web site for a collection of themes that includes this theme, or just the author's Web site.

preview The theme's preview image, within the static folder.

version The version of the theme.

Templates

Flask and therefore also FlaskBB uses the Jinja2 templating engine, so you should read its documentation to learn about the actual syntax of the templates.

All templates are by default loaded from FlaskBB's `templates/` folder. In order to create your own theme, you have to create a `templates/` folder in your themes directory and optionally also copy the `layout.html` file from FlaskBB's template folder over to yours. This `layout.html` file is your starting point. Every template will extend it. If you want to overwrite other templates, just copy them over from the templates folder and modify them to your liking.

Each loaded template will have a global function named *theme* available to look up the theme's templates. For example, if you want to extend, import, or include another template from your theme, you can use `theme(template_name)`, like this:

```
{% extends theme('layout.html') %}
{% from theme('macros.html') import horizontal_field %}
```

Note: If the template you requested **doesn't** exist within the theme, it will **fallback** to using the application's template.

If you pass *false* as the second parameter, it will only return the theme's template.

```
{# This template, for example, does not exist in FlaskBB #}
{% include theme('header.html', false) %}
```

You can also explicitly import/include templates from FlaskBB. Just use the tag without calling *theme*.

```
{% from 'macros.html' import topnav %}
```

You can also get the URL for the theme's media files with the *theme_static* function:

```
<link rel=stylesheet href="{{ theme_static('style.css') }}">
```

To include the static files that FlaskBB ships with, you just proceed as normal:

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/pygments.css') }}">
```

If you want to get information about the currently active theme, you can do that with the *theme_get_info* function:

```
This theme is <a href="{{ theme_get_info('website')}}">
  <b>{{ theme_get_info('name') }}</b>
</a>
```

Advanced Example

A more advanced example of a theme, is our own default theme called **Aurora**. We do not have a `layout.html` file because we want to avoid code duplication and are just falling back to the one that FlaskBB ships with in its `templates/` folder. In order to use your own stylesheets you have to create a `layout.html` file. It's probably the easiest to just copy the `layout.html` from FlaskBB's `templates/` folder into your themes `templates/` folder.

For example, the forums on FlaskBB are using a slightly modified version of the Aurora theme. It is available on GitHub here: Aurora Mod. The modified version just adds a top navigation and uses a different footer.

Prerequisites

To use the same build tools, which we also use to develop the Aurora theme, you have to make sure that you have npm installed. You can install npm by following the official installation guide.

The theme also uses SASS, a CSS preprocessor, to make development easier. If you are not familar with SASS but want to use it, which I can really recommend, follow this guide to get a basic understanding of it.

As explained in *Field Explanation*, each theme must have a unique theme **identifier** - so open up `info.json` (from your themes folder) with your favorite editor and adjust all the fields properly.

Next, do the same thing for the `package.json` file. This file is used by npm to install some libraries like Bootstrap. A detailed explanation about all the fields is available from package.json documentation page.

To install the stated requirements in `package.json` just run the `npm install` command in the directory where the `package.json` file is located. Now you have set up the toolchain which is used for the Aurora theme.

Toolchain Commands

For the build, minify, etc. process we use npm's task runner. Just hit up `npm run` to get a list with all available commands. Following commands are used:

```
Usage
  npm run [TASK]

Available tasks
  clean
    rm -f node_modules
  autoprefixer
    postcss -u autoprefixer -r static/css/*
  scss
    ./tools/build_css
  uglify
    ./tools/build_js
  imagemin
    imagemin src/img/* -o static/img
  fonts
    ./tools/build_fonts
  build:css
    npm run scss && npm run autoprefixer
  build:js
    npm run uglify
  build:images
    npm run imagemin && npm run fonts
  build:all
    npm run build:css && npm run build:js && npm run build:images
  watch:css
    onchange 'src/scss' -- npm run build:css
  watch:js
    onchange 'src/js' -- npm run build:js
  watch:all
    npm-run-all -p watch:css watch:js
```

For example, to watch for changes in our JS and SCSS files, you just have to run:

```
npm run watch:all
```

and upon changes it will automatically rebuild the files.

1.2.2 Hooks

In FlaskBB we distinguish from *Python Hooks* and *Template Hooks*. Python Hooks are prefixed with `flaskbb_` and called are called in Python files whereas Template Hooks have to be prefixed with `flaskbb_tpl_` and are executed in the templates.

If you miss a hook, feel free to open a new issue or create a pull request. The pull request should always contain a entry in this document with a small example.

A hook needs a hook specification which are defined in `flaskbb.plugins.spec`. All hooks have to be prefixed with `flaskbb_` and template hooks with `flaskbb_tpl_`.

Be sure to also check out the *API* documentation for interfaces that interact with these plugins in interesting ways.

Application Startup Hooks

Application startup hooks are called when the application is created, either through a WSGI server (uWSGI or gunicorn for example) or by the `flaskbb` command.

Unless noted, all FlaskBB hooks are called after the relevant builtin FlaskBB setup has run (e.g. `flaskbb_load_blueprints` is called after all standard FlaskBB blueprints have been loaded).

The hooks below are listed in the order they are called.

`flaskbb.plugins.spec.`**`flaskbb_extensions`**`(`*app*`)`
Hook for initializing any plugin loaded extensions.

`flaskbb.plugins.spec.`**`flaskbb_load_blueprints`**`(`*app*`)`
Hook for registering blueprints.

Parameters **app** – The application object.

`flaskbb.plugins.spec.`**`flaskbb_jinja_directives`**`(`*app*`)`
Hook for registering jinja filters, context processors, etc.

Parameters **app** – The application object.

`flaskbb.plugins.spec.`**`flaskbb_request_processors`**`(`*app*`)`
Hook for registering pre/post request processors.

Parameters **app** – The application object.

`flaskbb.plugins.spec.`**`flaskbb_errorhandlers`**`(`*app*`)`
Hook for registering error handlers.

Parameters **app** – The application object.

`flaskbb.plugins.spec.`**`flaskbb_load_migrations`**`()`
Hook for registering additional migrations.

`flaskbb.plugins.spec.`**`flaskbb_load_translations`**`()`
Hook for registering translation folders.

`flaskbb.plugins.spec.`**`flaskbb_load_post_markdown_class`**`(`*app*`)`
Hook for loading a mistune renderer child class in order to render markdown on posts and user signatures. All classes returned by this hook will be composed into a single class to render markdown for posts.

Since all classes will be composed together, child classes should call super as appropriate and not add any new arguments to *__init__* since the class will be insantiated with predetermined arguments.

Example:

```
class YellingRenderer(mistune.Renderer):
    def paragraph(self, text):
        return super(YellingRenderer, self).paragraph(text.upper())

@impl
def flaskbb_load_post_markdown_class():
    return YellingRenderer
```

Parameters **app** (*Flask*) – The application object associated with the class if needed

`flaskbb.plugins.spec.`**`flaskbb_load_nonpost_markdown_class`**(*app*)

Hook for loading a mistune renderer child class in order to render markdown in locations other than posts, for example in category or forum descriptions. All classes returned by this hook will be composed into a single class to render markdown for nonpost content (e.g. forum and category descriptions).

Since all classes will be composed together, child classes should call super as appropriate and not add any new arguments to *__init__* since the class will be insantiated with predetermined arguments.

Example:

```
class YellingRenderer(mistune.Renderer):
    def paragraph(self, text):
        return super(YellingRenderer, self).paragraph(text.upper())

@impl
def flaskbb_load_nonpost_markdown_class():
    return YellingRenderer
```

Parameters **app** (*Flask*) – The application object associated with the class if needed

`flaskbb.plugins.spec.`**`flaskbb_additional_setup`**(*app*, *pluggy*)

Hook for any additional setup a plugin wants to do after all other application setup has finished.

For example, you could apply a WSGI middleware:

```
@impl
def flaskbb_additional_setup(app):
    app.wsgi_app = ProxyFix(app.wsgi_app)
```

Parameters

- **app** – The application object.
- **pluggy** – The pluggy object.

FlaskBB CLI Hooks

These hooks are only invoked when using the `flaskbb` CLI.

`flaskbb.plugins.spec.`**`flaskbb_cli`**(*cli*, *app*)

Hook for registering CLI commands.

For example:

```
@impl
def flaskbb_cli(cli):
    @cli.command()
    def testplugin():
        click.echo("Hello Testplugin")

    return testplugin
```

Parameters

- **app** – The application object.
- **cli** – The FlaskBBGroup CLI object.

`flaskbb.plugins.spec.`**`flaskbb_shell_context`**`()`
Hook for registering shell context handlers Expected to return a single callable function that returns a dictionary or iterable of key value pairs.

FlaskBB Event Hooks

`flaskbb.plugins.spec.`**`flaskbb_event_post_save_before`**`(`*post*`)`
Hook for handling a post before it has been saved.

Parameters **post** (`flaskbb.forum.models.Post`) – The post which triggered the event.

`flaskbb.plugins.spec.`**`flaskbb_event_post_save_after`**`(`*post*, *is_new*`)`
Hook for handling a post after it has been saved.

Parameters

- **post** (`flaskbb.forum.models.Post`) – The post which triggered the event.
- **is_new** (*bool*) – True if the post is new, False if it is an edit.

`flaskbb.plugins.spec.`**`flaskbb_event_topic_save_before`**`(`*topic*`)`
Hook for handling a topic before it has been saved.

Parameters **topic** (`flaskbb.forum.models.Topic`) – The topic which triggered the event.

`flaskbb.plugins.spec.`**`flaskbb_event_topic_save_after`**`(`*topic*, *is_new*`)`
Hook for handling a topic after it has been saved.

Parameters

- **topic** (`flaskbb.forum.models.Topic`) – The topic which triggered the event.
- **is_new** (*bool*) – True if the topic is new, False if it is an edit.

`flaskbb.plugins.spec.`**`flaskbb_event_user_registered`**`(`*username*`)`
Hook for handling events after a user is registered

Parameters **username** – The username of the newly registered user.

`flaskbb.plugins.spec.`**`flaskbb_authenticate`**`(`*identifier*, *secret*`)`
Hook for authenticating users in FlaskBB. This hook should return either an instance of *`flaskbb.user.models.User`* or None.

If a hook decides that all attempts for authentication should end, it may raise a `flaskbb.core.exceptions.StopAuthentication` and include a reason why authentication was stopped.

Only the first User result will used and the default FlaskBB authentication is tried last to give others an attempt to authenticate the user instead.

See also: `AuthenticationProvider`

Example of alternative auth:

```
def ldap_auth(identifier, secret):
    "basic ldap example with imaginary ldap library"
    user_dn = "uid={},ou=flaskbb,dc=flaskbb,dc=org"
    try:
        ldap.bind(user_dn, secret)
        return User.query.join(
            UserLDAP
        ).filter(
            UserLDAP.dn==user_dn
        ).with_entities(User).one()
    except:
        return None

@impl
def flaskbb_authenticate(identifier, secret):
    return ldap_auth(identifier, secret)
```

Example of ending authentication:

```
def prevent_login_with_too_many_failed_attempts(identifier):
    user = User.query.filter(
        db.or_(
            User.username == identifier,
            User.email == identifier
        )
    ).first()

    if user is not None:
        if has_too_many_failed_logins(user):
            raise StopAuthentication(_(
                "Your account is temporarily locked due to too many login attempts
↪"
            ))

@impl(tryfirst=True)
def flaskbb_authenticate(user, identifier):
    prevent_login_with_too_many_failed_attempts(identifier)
```

`flaskbb.plugins.spec.`**`flaskbb_post_authenticate`**`(`*user*`)`

Hook for handling actions that occur after a user is authenticated but before setting them as the current user.

This could be used to handle MFA. However, these calls will be blocking and should be taken into account.

Responses from this hook are not considered at all. If a hook should need to prevent the user from logging in, it should register itself as tryfirst and raise a `flaskbb.core.exceptions.StopAuthentication` and include why the login was prevented.

See also: `PostAuthenticationHandler`

Example:

```
def post_auth(user):
    today = utcnow()
    if is_anniversary(today, user.date_joined):
        flash(_("Happy registerversary!"))
```

(continues on next page)

(continued from previous page)

```
@impl
def flaskbb_post_authenticate(user):
    post_auth(user)
```

`flaskbb.plugins.spec.`**`flaskbb_authentication_failed`**`(`*identifier*`)`

Hook for handling authentication failure events. This hook will only be called when no authentication providers successfully return a user or a `flaskbb.core.exceptions.StopAuthentication` is raised during the login process.

See also: `AuthenticationFailureHandler`

Example:

```
def mark_failed_logins(identifier):
    user = User.query.filter(
        db.or_(
            User.username == identifier,
            User.email == identifier
        )
    ).first()

    if user is not None:
        if user.login_attempts is None:
            user.login_attempts = 1
        else:
            user.login_attempts += 1
        user.last_failed_login = utcnow()
```

`flaskbb.plugins.spec.`**`flaskbb_reauth_attempt`**`(`*user*, *secret*`)`

Hook for handling reauth in FlaskBB

These hooks receive the currently authenticated user and the entered secret. Only the first response from this hook is considered – similar to the authenticate hooks. A successful attempt should return True, otherwise None for an unsuccessful or untried reauth from an implementation. Reauth will be considered a failure if no implementation return True.

If a hook decides that a reauthenticate attempt should cease, it may raise StopAuthentication.

See also: `ReauthenticateProvider`

Example of checking secret or passing to the next implementer:

```
@impl
def flaskbb_reauth_attempt(user, secret):
    if check_password(user.password, secret):
        return True
```

Example of forcefully ending reauth:

```
@impl
def flaskbb_reauth_attempt(user, secret):
    if user.login_attempts > 5:
        raise StopAuthentication(_("Too many failed authentication attempts"))
```

`flaskbb.plugins.spec.`**`flaskbb_post_reauth`**`(`*user*`)`

Hook called after successfully reauthenticating.

These hooks are called a user has passed the flaskbb_reauth_attempt hooks but before their reauth is confirmed so a post reauth implementer may still force a reauth to fail by raising StopAuthentication.

Results from these hooks are not considered.

See also: `PostReauthenticateHandler`

`flaskbb.plugins.spec.`**`flaskbb_reauth_failed`**`(`*user*`)`

Hook called if a reauth fails.

These hooks will only be called if no implementation for flaskbb_reauth_attempt returns a True result or if an implementation raises StopAuthentication.

If an implementation raises ForceLogout it should register itself as trylast to give other reauth failed handlers an opprotunity to run first.

See also: `ReauthenticateFailureHandler`

FlaskBB Form Hooks

`flaskbb.plugins.spec.`**`flaskbb_form_new_post_save`**`(`*form*`)`

Hook for modifying the `ReplyForm`.

This hook is called while populating the post object with the data from the form. The post object will be saved after the hook call.

Parameters

- **`form`** – The form object.
- **`post`** – The post object.

`flaskbb.plugins.spec.`**`flaskbb_form_new_post`**`(`*form*`)`

Hook for modifying the `ReplyForm`.

For example:

```
@impl
def flaskbb_form_new_post(form):
    form.example = TextField("Example Field", validators=[
        DataRequired(message="This field is required"),
        Length(min=3, max=50)])
```

Parameters **`form`** – The `ReplyForm` class.

`flaskbb.plugins.spec.`**`flaskbb_form_new_topic`**`(`*form*`)`

Hook for modifying the `NewTopicForm`

Parameters **`form`** – The `NewTopicForm` class.

`flaskbb.plugins.spec.`**`flaskbb_form_new_topic_save`**`(`*form*, *topic*`)`

Hook for modifying the `NewTopicForm`.

This hook is called while populating the topic object with the data from the form. The topic object will be saved after the hook call.

Parameters

- **`form`** – The form object.
- **`topic`** – The topic object.

`flaskbb.plugins.spec.`**`flaskbb_form_registration`**(*form*)
Hook for modifying the `RegisterForm`.

Parameters **`form`** – The form class

Template Hooks

Note: Template hooks, which are used in forms, are usually rendered after the hidden CSRF token field and before an submit field.

`flaskbb.plugins.spec.`**`flaskbb_tpl_navigation_before`**()
Hook for registering additional navigation items.

in `templates/layout.html`.

`flaskbb.plugins.spec.`**`flaskbb_tpl_navigation_after`**()
Hook for registering additional navigation items.

in `templates/layout.html`.

`flaskbb.plugins.spec.`**`flaskbb_tpl_user_nav_loggedin_before`**()
Hook for registering additional user navigational items which are only shown when a user is logged in.

in `templates/layout.html`.

`flaskbb.plugins.spec.`**`flaskbb_tpl_user_nav_loggedin_after`**()
Hook for registering additional user navigational items which are only shown when a user is logged in.

in `templates/layout.html`.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_registration_before`**(*form*)
This hook is emitted in the Registration form **before** the first input field but after the hidden CSRF token field.

in `templates/auth/register.html`.

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_registration_after`**(*form*)
This hook is emitted in the Registration form **after** the last input field but before the submit field.

in `templates/auth/register.html`.

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_user_details_before`**(*form*)
This hook is emitted in the Change User Details form **before** an input field is rendered.

in `templates/user/change_user_details.html`.

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_user_details_after`**(*form*)
This hook is emitted in the Change User Details form **after** the last input field has been rendered but before the submit field.

in `templates/user/change_user_details.html`.

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_new_post_before`**(*form*)
Hook for inserting a new form field before the first field is rendered.

For example:

```
@impl
def flaskbb_tpl_form_new_post_after(form):
    return render_template_string(
        """
        <div class="form-group">
            <div class="col-md-12 col-sm-12 col-xs-12">
                <label>{{ form.example.label.text }}</label>

                {{ form.example(class="form-control",
                                placeholder=form.example.label.text) }}

                {%- for error in form.example.errors -%}
                <span class="help-block">{{error}}</span>
                {%- endfor -%}
            </div>
        </div>
        """
```

in `templates/forum/new_post.html`

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_new_post_after`**(*form*)

Hook for inserting a new form field after the last field is rendered (but before the submit field).

in `templates/forum/new_post.html`

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_new_topic_before`**(*form*)

Hook for inserting a new form field before the first field is rendered (but before the CSRF token).

in `templates/forum/new_topic.html`

Parameters **`form`** – The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_form_new_topic_after`**(*form*)

Hook for inserting a new form field after the last field is rendered (but before the submit button).

in `templates/forum/new_topic.html` :param form: The form object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_profile_settings_menu`**()

This hook is emitted on the user settings page in order to populate the side bar menu. Implementations of this hook should return a list of tuples that are view name and display text. The display text will be provided to the translation service so it is unnecessary to supply translated text.

A plugin can declare a new block by setting the view to None. If this is done, consider marking the hook implementation with *trylast=True* to avoid capturing plugins that do not create new blocks.

For example:

```
@impl(trylast=True)
def flaskbb_tpl_profile_settings_menu():
    return [
        (None, 'Account Settings'),
        ('user.settings', 'General Settings'),
        ('user.change_user_details', 'Change User Details'),
        ('user.change_email', 'Change E-Mail Address'),
        ('user.change_password', 'Change Password')
    ]
```

Hookwrappers for this spec should not be registered as FlaskBB supplies its own hookwrapper to flatten all the lists into a single list.

in `templates/user/settings_layout.html`

`flaskbb.plugins.spec.`**`flaskbb_tpl_profile_sidebar_stats`**`(`*user*`)`

This hook is emitted on the users profile page below the standard information. For example, it can be used to add additional items such as a link to the profile.

in `templates/user/profile_layout.html`

Parameters **`user`** – The user object for whom the profile is currently visited.

`flaskbb.plugins.spec.`**`flaskbb_tpl_post_author_info_before`**`(`*user*, *post*`)`

This hook is emitted before the information about the author of a post is displayed (but after the username).

in `templates/forum/topic.html`

Parameters

- **`user`** – The user object of the post's author.
- **`post`** – The post object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_post_author_info_after`**`(`*user*, *post*`)`

This hook is emitted after the information about the author of a post is displayed (but after the username).

in `templates/forum/topic.html`

Parameters

- **`user`** – The user object of the post's author.
- **`post`** – The post object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_post_content_before`**`(`*post*`)`

Hook to do some stuff before the post content is rendered.

in `templates/forum/topic.html`

Parameters **`post`** – The current post object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_post_content_after`**`(`*post*`)`

Hook to do some stuff after the post content is rendered.

in `templates/forum/topic.html`

Parameters **`post`** – The current post object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_post_menu_before`**`(`*post*`)`

Hook for inserting a new item at the beginning of the post menu.

in `templates/forum/topic.html`

Parameters **`post`** – The current post object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_post_menu_after`**`(`*post*`)`

Hook for inserting a new item at the end of the post menu.

in `templates/forum/topic.html`

Parameters **`post`** – The current post object.

`flaskbb.plugins.spec.`**`flaskbb_tpl_topic_controls`**`(`*topic*`)`

Hook for inserting additional topic moderation controls.

in `templates/forum/topic_controls.html`

Parameters **topic** – The current topic object.

flaskbb.plugins.spec.**flaskbb_tpl_admin_settings_menu**(*user*)

This hook is emitted in the admin panel and used to add additional navigation links to the admin menu.

Implementations of this hook should return a list of tuples that are view name, display text and optionally an icon. The display text will be provided to the translation service so it is unnecessary to supply translated text.

For example:

```
@impl(trylast=True)
def flaskbb_tpl_admin_settings_menu():
    # only add this item if the user is an admin
    if Permission(IsAdmin, identity=current_user):
        return [
            ("myplugin.foobar", "Foobar", "fa fa-foobar")
        ]
```

Hookwrappers for this spec should not be registered as FlaskBB supplies its own hookwrapper to flatten all the lists into a single list.

in templates/management/management_layout.html

Parameters **user** – The current user object.

1.2.3 Plugin Development

Developing new Plugins

If you want to write a plugin, it's a very good idea to checkout existing plugins. A good starting point for example is the Portal Plugin.

Also make sure to check out the cookiecutter-flaskbb-plugin project, which is a cookiecutter template which helps you to create new plugins.

For example, the structure of a plugin could look like this:

```
your_package_name
|-- setup.py
|-- my_plugin
    |-- __init__.py
    |-- views.py
    |-- models.py
    |-- forms.py
    |-- static
    |   |-- style.css
    |-- templates
        |-- myplugin.html
    |-- migrations
        |-- 59f7c49b6289_init.py
```

Metadata

FlaskBB Plugins are usually following the naming scheme of flaskbb-plugin-YOUR_PLUGIN_NAME which should make them better distinguishable from other PyPI distributions.

A proper plugin should have at least put the following metadata into the setup.py file.

```
setup(
    name="flaskbb-plugin-YOUR_PLUGIN_NAME",   # name on PyPI
    packages=["your_package_name"],  # name of the folder your plugin is located in
    version='1.0',
    url=<url to your project>,
    license=<your license>,
    author=<you>,
    author_email=<your email>,
    description=<your short description>,
    long_description=__doc__,
    include_package_data=True,
    zip_safe=False,
    platforms='any',

    entry_points={
        'flaskbb_plugin': [
            'unique_name_of_plugin = your_package_name.pluginmodule',  # most␣
↪important part
        ]
    }
)
```

The most important part here is the `entry_point`. Here you tell FlaskBB the unique name of your plugin and where your plugin module is located inside your project. Entry points are a feature that is provided by setuptools. FlaskBB looks up the `flaskbb_plugin` entrypoint to discover its plugins. Have a look at the setup script documentation and the sample setup.py file to get a better idea what the `setup.py` file is all about it.

For a full example, checkout the Portal Plugin.

Settings

Plugins can create settings which integrate with the 'Settings' tab of the Admin Panel.

The settings are stored in a dictionary with a given structure. The name of the dictionary must be `SETTINGS` and be placed in the plugin module.

The structure of the `SETTINGS` dictionary is best explained via an example:

```
SETTINGS = {
    # This key has to be unique across FlaskBB.
    # Using a prefix is recommended.
    'forum_ids': {

        # Default Value. The type of the default value depends on the
        # SettingValueType.
        'value': [1],

        # The Setting Value Type.
        'value_type': SettingValueType.selectmultiple,

        # The human readable name of your configuration variable
        'name': "Forum IDs",

        # A short description of what the settings variable does
        'description': ("The forum ids from which forums the posts "
                        "should be displayed on the portal."),
```

(continues on next page)

(continued from previous page)

```
        # extra stuff like the 'choices' in a select field or the
        # validators are defined in here
        'extra': {"choices": available_forums, "coerce": int}
    }
}
```

Table 1: Available Setting Value Types

Setting Value Type	Parsed & Saved As
SettingValueType.string	str
SettingValueType.integer	int
SettingValueType.float	float
SettingValueType.boolean	bool
SettingValueType.select	list
SettingValueType.selectmultiple	list

Table 2: Available Additional Options via the `extra` Keyword

Options	Applicable Types	Description
`min`	string, integer, float	**Optional.** The minimum required length of the setting value. If used on a numeric type, it will check the minimum value.
`max`	string, integer, float	**Optional.** The maximum required length of the setting value. If used on a numeric type, it will check the maximum value.
`choices`	select, selectmultiple	**Required.** A callable which returns a sequence of (value, label) pairs.
`coerce`	select, selectmultiple	**Optional.** Coerces the select values to the given type.

Validating the size of the integer/float and the length of the string fields is also possible via the `min` and `max` keywords:

```
'recent_topics': {
    ...
    'extra': {"min": 1},
},
```

The `select` and `selectmultiple` fields have to provide a callback which lists all the available choices. This is done via the `choices` keyword. In addition to that they can also specify the `coerce` keyword which will coerce the input value into the specified type.:

```
'forum_ids': {
    ...
    'extra': {"choices": available_forums, "coerce": int}
}
```

For more information see the *Settings* chapter.

Using Hooks

Hooks are invoked based on an event occurring within FlaskBB. This makes it possible to change the behavior of certain actions without modifying the actual source code of FlaskBB.

For your plugin to actually do something useful, you probably want to 'hook' your code into FlaskBB. This can be done throughout a lot of places in the code. FlaskBB loads and calls the hook calls hook functions from registered plugins for any given hook specification.

Each hook specification has a corresponding hook implementation. By default, all hooks that are prefix with `flaskbb_` will be marked as a standard hook implementation. It is possible to modify the behavior of hooks. For example, default hooks are called in LIFO registered order. Although, registration order might not be deterministic. A hookimpl can influence its call-time invocation position using special attributes. If marked with a "tryfirst" or "trylast" option it will be executed first or last respectively in the hook call loop:

```
hookimpl = HookimplMarker('flaskbb')

@hookimpl(trylast=True)
def flaskbb_additional_setup(app):
    return "save the best for last"
```

In order to extend FlaskBB with your Plugin you will need to connect your callbacks to the hooks.

Let's look at an actually piece of used code.

```
def flaskbb_load_blueprints(app):
    app.register_blueprint(portal, url_prefix="/portal")
```

By defining a function called `flaskbb_load_blueprints`, which has a corresponding hook specification under the same name. FlaskBB will pass in an `app` object as specified in the hook spec, which we will use to register a new blueprint. It is also possible to completely omit the `app` argument from the function where it is **not possible** to add new arguments to the hook implemention.

For a complete list of all available hooks in FlaskBB see the *Hooks* section.

pytest and pluggy are good resources to get better understanding on how to write hook functions using pluggy.

Plugin Management

FlaskBB provides a couple of helpers for helping with plugin management.

Plugin Registry

The plugin registry holds all available plugins. It shows the plugin's status whether it is enabled or disabled, installable or installed. The registry also holds a reference to the plugin's instance, provides an interface to access the plugins metadata and stores its settings.

You can query it like any SQLAlchemy Model:

```
plugin = PluginRegistry.query.filter_by(name="portal").first()
```

class `flaskbb.plugins.models.`**`PluginRegistry`**(***kwargs*)

`settings`
Returns a dict with contains all the settings in a plugin.

`info`
Returns some information about the plugin.

`is_installable`
Returns True if the plugin has settings that can be installed.

is_installed
Returns True if the plugin is installed.

get_settings_form()
Generates a settings form based on the settings.

update_settings(*settings*)
Updates the given settings of the plugin.

Parameters **settings** – A dictionary containing setting items.

add_settings(*settings*, *force=False*)
Adds the given settings to the plugin.

Parameters

- **settings** – A dictionary containing setting items.
- **force** – Forcefully overwrite existing settings.

Plugin Manager

FlaskBB overrides the PluginManager from pluggy to provide some additional functionality like accessing the information stored in a setup.py file. The plugin manager will only list the currently enabled plugins and can be used to directly access the plugins instance by its name.

Accessing a plugins instance is as easy as:

```
plugin_instance = current_app.pluggy.get_plugin(name)
```

class flaskbb.plugins.manager.**FlaskBBPluginManager**(*project_name*, *implprefix=None*)
Overwrites `pluggy.PluginManager` to add FlaskBB specific stuff.

register(*plugin*, *name=None*, *internal=False*)
Register a plugin and return its canonical name or None if the name is blocked from registering. Raise a ValueError if the plugin is already registered.

unregister(*plugin=None*, *name=None*)
Unregister a plugin object and all its contained hook implementations from internal data structures.

set_blocked(*name*)
Block registrations of the given name, unregister if already registered.

is_blocked(*name*)
Return True if the name blockss registering plugins of that name.

get_plugin(*name*)
Return a plugin or None for the given name.

get_name(*plugin*)
Return name for registered plugin or None if not registered.

load_setuptools_entrypoints(*entrypoint_name*)
Load modules from querying the specified setuptools entrypoint name. Return the number of loaded plugins.

get_metadata(*name*)
Returns the metadata for a given name.

list_name()
Returns only the enabled plugin names.

list_internal_name_plugin()
: Returns a list of internal name/plugin pairs.

list_plugin_metadata()
: Returns the metadata for all plugins

list_disabled_plugins()
: Returns a name/distinfo tuple pairs of disabled plugins.

get_disabled_plugins()
: Returns a list with disabled plugins.

get_internal_plugins()
: Returns a set of registered internal plugins.

get_external_plugins()
: Returns a set of registered external plugins.

add_hookcall_monitoring(*before*, *after*)
: add before/after tracing functions for all hooks and return an undo function which, when called, will remove the added tracers.

  `before(hook_name, hook_impls, kwargs)` will be called ahead of all hook calls and receive a hookcaller instance, a list of HookImpl instances and the keyword arguments for the hook call.

  `after(outcome, hook_name, hook_impls, kwargs)` receives the same arguments as `before` but also a `_Result`` object which represents the result of the overall hook call.

add_hookspecs(*module_or_class*)
: add new hook specifications defined in the given module_or_class. Functions are recognized if they have been decorated accordingly.

check_pending()
: Verify that all hooks which have not been verified against a hook specification are optional, otherwise raise PluginValidationError

enable_tracing()
: enable tracing of hook calls and return an undo function.

get_canonical_name(*plugin*)
: Return canonical name for a plugin object. Note that a plugin may be registered under a different name which was specified by the caller of register(plugin, name). To obtain the name of an registered plugin use `get_name(plugin)` instead.

get_hookcallers(*plugin*)
: get all hook callers for the specified plugin.

get_plugins()
: return the set of registered plugins.

has_plugin(*name*)
: Return True if a plugin with the given name is registered.

is_registered(*plugin*)
: Return True if the plugin is already registered.

list_name_plugin()
: return list of name/plugin pairs.

list_plugin_distinfo()
: return list of distinfo/plugin tuples for all setuptools registered plugins.

subset_hook_caller(*name*, *remove_plugins*)
Return a new _HookCaller instance for the named method which manages calls to all registered plugins except the ones from remove_plugins.

1.2.4 API

This is the software API for FlaskBB, such as interfaces, models, exceptions and provided implementations where appropriate.

Core Exceptions

These are exceptions that aren't specific to any one part of FlaskBB and are used ubiquitously.

exception `flaskbb.core.exceptions.`**BaseFlaskBBError**
Root exception for FlaskBB.

exception `flaskbb.core.exceptions.`**ValidationError**(*attribute*, *reason*)
Used to signal validation errors for things such as token verification, user registration, etc.

Parameters

- **attribute** (`str`) – The attribute the validation error applies to, if the validation error applies to multiple attributes or to the entire object, this should be set to None
- **reason** (`str`) – Why the attribute, collection of attributes or object is invalid.

exception `flaskbb.core.exceptions.`**StopValidation**(*reasons*)
Raised from validation handlers to signal that validation should end immediately and no further processing should be done.

Can also be used to communicate all errors raised during a validation run.

Parameters **reasons** – A sequence of *(attribute, reason)* pairs explaining why the object is invalid.

Models

FlaskBB uses SQLAlchemy as it's ORM. The models are split in three modules which are covered below.

Forum Models

This module contains all related models for the forums.

The hierarchy looks like this: Category > Forum > Topic > Post. In the Report model are stored the reports and the TopicsRead and ForumsRead models are used to store the status if the user has read a specific forum or not.

class `flaskbb.forum.models.`**Category**(***kwargs*)

slug
Returns a slugified version from the category title

url
Returns the slugified url for the category

delete(*users=None*)
Deletes a category. If a list with involved user objects is passed, it will also update their post counts

Parameters **users** – A list with user objects

classmethod get_all(*user*)

Get all categories with all associated forums. It returns a list with tuples. Those tuples are containing the category and their associated forums (whose are stored in a list).

For example:

```
[(<Category 1>, [(<Forum 2>, <ForumsRead>), (<Forum 1>, None)]),
 (<Category 2>, [(<Forum 3>, None), (<Forum 4>, None)])]
```

Parameters **user** – The user object is needed to check if we also need their forumsread object.

classmethod get_forums(*category_id*, *user*)

Get the forums for the category. It returns a tuple with the category and the forums with their forumsread object are stored in a list.

A return value can look like this for a category with two forums:

```
(<Category 1>, [(<Forum 1>, None), (<Forum 2>, None)])
```

Parameters

- **category_id** – The category id
- **user** – The user object is needed to check if we also need their forumsread object.

class flaskbb.forum.models.**Forum**(***kwargs*)

slug

Returns a slugified version from the forum title

url

Returns the slugified url for the forum

last_post_url

Returns the url for the last post in the forum

update_last_post(*commit=True*)

Updates the last post in the forum.

update_read(*user*, *forumsread*, *topicsread*)

Updates the ForumsRead status for the user. In order to work correctly, be sure that *topicsread is **not** 'None*.

Parameters

- **user** – The user for whom we should check if he has read the forum.
- **forumsread** – The forumsread object. It is needed to check if if the forum is unread. If *forumsread* is *None* and the forum is unread, it will create a new entry in the *ForumsRead* relation, else (and the forum is still unread) we are just going to update the entry in the *ForumsRead* relation.
- **topicsread** – The topicsread object is used in combination with the forumsread object to check if the forumsread relation should be updated and therefore is unread.

recalculate(*last_post=False*)

Recalculates the post_count and topic_count in the forum. Returns the forum with the recounted stats.

Parameters **last_post** – If set to `True` it will also try to update the last post columns in the forum.

save(*groups=None*)
Saves a forum

Parameters

- **moderators** – If given, it will update the moderators in this forum with the given iterable of user objects.
- **groups** – A list with group objects.

delete(*users=None*)
Deletes forum. If a list with involved user objects is passed, it will also update their post counts

Parameters **users** – A list with user objects

move_topics_to(*topics*)
Moves a bunch a topics to the forum. Returns `True` if all topics were moved successfully to the forum.

Parameters **topics** – A iterable with topic objects.

classmethod get_forum(*forum_id*, *user*)
Returns the forum and forumsread object as a tuple for the user.

Parameters

- **forum_id** – The forum id
- **user** – The user object is needed to check if we also need their forumsread object.

classmethod get_topics(*forum_id*, *user*, *page=1*, *per_page=20*)
Get the topics for the forum. If the user is logged in, it will perform an outerjoin for the topics with the topicsread and forumsread relation to check if it is read or unread.

Parameters

- **forum_id** – The forum id
- **user** – The user object
- **page** – The page whom should be loaded
- **per_page** – How many topics per page should be shown

class flaskbb.forum.models.**Topic**(*title=None*, *user=None*)

second_last_post
Returns the second last post or None.

slug
Returns a slugified version of the topic title.

url
Returns the slugified url for the topic.

first_unread(*topicsread*, *user*, *forumsread=None*)
Returns the url to the first unread post. If no unread posts exist it will return the url to the topic.

Parameters

- **topicsread** – The topicsread object for the topic
- **user** – The user who should be checked if he has read the last post in the topic
- **forumsread** – The forumsread object in which the topic is. If you also want to check if the user has marked the forum as read, than you will also need to pass an forumsread object.

tracker_needs_update(*forumsread*, *topicsread*)
Returns True if the topicsread tracker needs an update. Also, if the `TRACKER_LENGTH` is configured, it will just recognize topics that are newer than the `TRACKER_LENGTH` (in days) as unread.

Parameters

- **forumsread** – The ForumsRead object is needed because we also need to check if the forum has been cleared sometime ago.
- **topicsread** – The topicsread object is used to check if there is a new post in the topic.

update_read(*user*, *forum*, *forumsread*)
Updates the topicsread and forumsread tracker for a specified user, if the topic contains new posts or the user hasn't read the topic. Returns True if the tracker has been updated.

Parameters

- **user** – The user for whom the readstracker should be updated.
- **forum** – The forum in which the topic is.
- **forumsread** – The forumsread object. It is used to check if there is a new post since the forum has been marked as read.

recalculate()
Recalculates the post count in the topic.

move(*new_forum*)
Moves a topic to the given forum. Returns True if it could successfully move the topic to forum.

Parameters **new_forum** – The new forum for the topic

save(*user=None*, *forum=None*, *post=None*)
Saves a topic and returns the topic object. If no parameters are given, it will only update the topic.

Parameters

- **user** – The user who has created the topic
- **forum** – The forum where the topic is stored
- **post** – The post object which is connected to the topic

delete(*users=None*)
Deletes a topic with the corresponding posts. If a list with user objects is passed it will also update their post counts

Parameters **users** – A list with user objects

hide(*user*, *users=None*)
Soft deletes a topic from a forum

unhide(*users=None*)
Restores a hidden topic to a forum

involved_users()
Returns a query of all users involved in the topic

class flaskbb.forum.models.**Post**(*content=None*, *user=None*, *topic=None*)

url
Returns the url for the post.

save (*user=None*, *topic=None*)
Saves a new post. If no parameters are passed we assume that you will just update an existing post. It returns the object after the operation was successful.

Parameters

- **user** – The user who has created the post
- **topic** – The topic in which the post was created

delete ()
Deletes a post and returns self.

class flaskbb.forum.models.**TopicsRead** (***kwargs*)

class flaskbb.forum.models.**ForumsRead** (***kwargs*)

class flaskbb.forum.models.**Report** (***kwargs*)

save (*post=None*, *user=None*)
Saves a report.

Parameters

- **post** – The post that should be reported
- **user** – The user who has reported the post
- **reason** – The reason why the user has reported the post

User Models

The user modules contains all related models for the users.

class flaskbb.user.models.**User** (***kwargs*)

is_active
Returns the state of the account. If the `ACTIVATE_ACCOUNT` option has been disabled, it will always return `True`. Is the option activated, it will, depending on the state of the account, either return `True` or `False`.

last_post
Returns the latest post from the user.

url
Returns the url for the user.

permissions
Returns the permissions for the user.

groups
Returns the user groups.

days_registered
Returns the amount of days the user is registered.

topic_count
Returns the thread count.

posts_per_day
Returns the posts per day count.

topics_per_day
Returns the topics per day count.

password
Returns the hashed password.

check_password(*password*)
Check passwords. If passwords match it returns true, else false.

classmethod authenticate(*login*, *password*)
A classmethod for authenticating users. It returns the user object if the user/password combination is ok. If the user has entered too often a wrong password, he will be locked out of his account for a specified time.

Parameters

- **login** – This can be either a username or a email address.
- **password** – The password that is connected to username and email.

recalculate()
Recalculates the post count from the user.

all_topics(*page*, *viewer*)
Returns a paginated result with all topics the user has created.

Parameters

- **page** – The page which should be displayed.
- **viewer** – The user who is viewing this user. It will return a list with topics that the *viewer* has access to and thus it will not display all topics from the requested user.

all_posts(*page*, *viewer*)
Returns a paginated result with all posts the user has created.

Parameters

- **page** – The page which should be displayed.
- **viewer** – The user who is viewing this user. It will return a list with posts that the *viewer* has access to and thus it will not display all posts from the requested user.

track_topic(*topic*)
Tracks the specified topic.

Parameters **topic** – The topic which should be added to the topic tracker.

untrack_topic(*topic*)
Untracks the specified topic.

Parameters **topic** – The topic which should be removed from the topic tracker.

is_tracking_topic(*topic*)
Checks if the user is already tracking this topic.

Parameters **topic** – The topic which should be checked.

add_to_group(*group*)
Adds the user to the *group* if he isn't in it.

Parameters **group** – The group which should be added to the user.

remove_from_group(*group*)
Removes the user from the *group* if he is in it.

Parameters **group** – The group which should be removed from the user.

in_group(*group*)
Returns True if the user is in the specified group.

Parameters **group** – The group which should be checked.

get_groups(***kwargs*)
Returns all the groups the user is in.

get_permissions(***kwargs*)
Returns a dictionary with all permissions the user has

invalidate_cache()
Invalidates this objects cached metadata.

ban()
Bans the user. Returns True upon success.

unban()
Unbans the user. Returns True upon success.

save(*groups=None*)
Saves a user. If a list with groups is provided, it will add those to the secondary groups from the user.

Parameters **groups** – A list with groups that should be added to the secondary groups from user.

delete()
Deletes the User.

class flaskbb.user.models.**Group**(***kwargs*)

classmethod get_member_group()
Returns the first member group.

Management Models

The management module contains all related models for the management of FlaskBB.

class flaskbb.management.models.**SettingsGroup**(***kwargs*)

class flaskbb.management.models.**Setting**(***kwargs*)

classmethod get_form(*group*)
Returns a Form for all settings found in *SettingsGroup*.

Parameters **group** – The settingsgroup name. It is used to get the settings which are in the specified group.

classmethod update(*settings*, *app=None*)
Updates the cache and stores the changes in the database.

Parameters **settings** – A dictionary with setting items.

classmethod get_settings(*from_group=None*)
This will return all settings with the key as the key for the dict and the values are packed again in a dict which contains the remaining attributes.

Parameters **from_group** – Optionally - Returns only the settings from a group.

classmethod as_dict(***kwargs*)
Returns all settings as a dict. This method is cached. If you want to invalidate the cache, simply execute `self.invalidate_cache()`.

Parameters

- **from_group** – Returns only the settings from the group as a dict.
- **upper** – If upper is `True`, the key will use upper-case letters. Defaults to `False`.

classmethod invalidate_cache()
Invalidates this objects cached metadata.

Registration

These interfaces and implementations control the user registration flow in FlaskBB.

Registration Interfaces

class flaskbb.core.auth.registration.**UserRegistrationInfo**(*username*, *password*, *email*, *language*, *group*)
User registration object, contains all relevant information for validating and creating a new user.

class flaskbb.core.auth.registration.**UserValidator**
Used to validate user registrations and stop the registration process by raising a *ValidationError*.

validate(*user_info*)
This method is abstract.

Parameters user_info (*UserRegistrationInfo*) – The provided registration information.

class flaskbb.core.auth.registration.**UserRegistrationService**
Used to manage the registration process. A default implementation is provided however, this interface is provided in case alternative flows are needed.

register(*user_info*)
This method is abstract.

Parameters user_info (*UserRegistrationInfo*) – The provided user registration information.

Registration Provided Implementations

class flaskbb.auth.services.registration.**UsernameRequirements**(*min*, *max*, *blacklist*)
Configuration for username requirements, minimum and maximum length and disallowed names.

class flaskbb.auth.services.registration.**UsernameValidator**(*requirements*)
Validates that the username for the registering user meets the minimum requirements (appropriate length, not a forbidden name).

class flaskbb.auth.services.registration.**UsernameUniquenessValidator**(*users*)
Validates that the provided username is unique in the application.

class flaskbb.auth.services.registration.**EmailUniquenessValidator**(*users*)
Validates that the provided email is unique in the application.

class `flaskbb.auth.services.registration.`**RegistrationService**(*validators*, *user_repo*)

Default registration service for FlaskBB, runs the registration information against the provided validators and if it passes, creates the user.

If any of the provided *UserValidators* raise a *ValidationError* then the register method will raise a *StopValidation* with all reasons why the registration was prevented.

Authentication

FlaskBB exposes several interfaces and hooks to customize authentication and implementations of these. For details on the hooks see *Hooks*

Authentication Interfaces

class `flaskbb.core.auth.authentication.`**AuthenticationManager**

Used to handle the authentication process. A default is implemented, however this interface is provided in case alternative flows are needed.

If a user successfully passes through the entire authentication process, then it should be returned to the caller.

authenticate(*identifier*, *secret*)

This method is abstract.

Parameters

- **identifier** (*str*) – An identifer for the user, typically this is either a username or an email.
- **secret** (*str*) – A secret to verify the user is who they say they are

Returns A fully authenticated but not yet logged in user

Return type *User*

class `flaskbb.core.auth.authentication.`**AuthenticationProvider**

Used to provide an authentication service for FlaskBB.

For example, an implementer may choose to use LDAP as an authentication source:

```
class LDAPAuthenticationProvider(AuthenticationProvider):
    def __init__(self, ldap_client):
        self.ldap_client = ldap_client

    def authenticate(self, identifier, secret):
        user_dn = "uid={},ou=flaskbb,ou=org".format(identifier)
        try:
            self.ldap_client.bind_user(user_dn, secret)
            return User.query.join(
                    UserLDAP
                ).filter(
                    UserLDAP.dn==user_dn
                ).with_entities(User).one()
        except Exception:
            return None
```

During an authentication process, a provider may raise a *StopAuthentication* exception to completely, but safely halt the process. This is most useful when multiple providers are being used.

authenticate(*identifier*, *secret*)
This method is abstract.

Parameters

- **identifier** (*str*) – An identifer for the user, typically this is either a username or an email.
- **secret** (*str*) – A secret to verify the user is who they say they are

Returns An authenticated user.

Return type *User*

class flaskbb.core.auth.authentication.**PostAuthenticationHandler**
Used to post process authentication success. Post authentication handlers recieve the user instance that was returned by the successful authentication rather than the identifer.

Postprocessors may decide to preform actions such as flashing a message to the user, clearing failed login attempts, etc.

Alternatively, a postprocessor can decide to fail the authentication process anyways by raising *StopAuthentication*, for example a user may successfully authenticate but has not yet activated their account.

Cancelling a successful authentication will cause registered *AuthenticationFailureHandler* instances to be run.

Success handlers should not return a value as it will not be considered.

handle_post_auth(*user*)
This method is abstact.

Parameters **user** (User) – An authenticated but not yet logged in user

class flaskbb.core.auth.authentication.**AuthenticationFailureHandler**
Used to post process authentication failures, such as no provider returning a user or a provider raising *StopAuthentication*.

Postprocessing may take many forms, such as incrementing the login attempts locking an account if too many attempts are made, forcing a reauth if the user is currently authenticated in a different session, etc.

Failure handlers should not return a value as it will not be considered.

handle_authentication_failure(*identifier*)
This method is abstract.

Parameters **identifier** (*str*) – An identifer for the user, typically this is either a username or an email.

Authentication Provided Implementations

class flaskbb.auth.services.authentication.**DefaultFlaskBBAuthProvider**
This is the default username/email and password authentication checker, locates the user based on the identifer passed – either username or email – and compares the supplied password to the hash connected to the matching user (if any).

Offers protection against timing attacks that would rely on the difference in response time from not matching a password hash.

class flaskbb.auth.services.authentication.**MarkFailedLogin**
Failure handler that marks the login attempt on the user and sets the last failed date when it happened.

class `flaskbb.auth.services.authentication.`**`BlockUnactivatedUser`**

Post auth handler that will block a user that has managed to pass the authentication check but has not actually activated their account yet.

class `flaskbb.auth.services.authentication.`**`ClearFailedLogins`**

Post auth handler that clears all failed login attempts from a user's account.

class `flaskbb.auth.services.authentication.`**`PluginAuthenticationManager`**(*plugin_manager*, *ses-sion*)

Authentication manager relying on plugin hooks to manage the authentication process. This is the default authentication manager for FlaskBB.

Reauthentication Interfaces

class `flaskbb.core.auth.authentication.`**`ReauthenticateManager`**

Used to handle the reauthentication process in FlaskBB. A default implementation is provided, however this is interface exists in case alternative flows are desired.

Unlike the AuthenticationManager, there is no need to return the user to the caller.

`reauthenticate`(*user*, *secret*)

This method is abstract.

Parameters

- **user** (*User*) – The current user instance
- **secret** (*str*) – The secret provided by the user

class `flaskbb.core.auth.authentication.`**`ReauthenticateProvider`**

Used to reauthenticate a user that is already logged into the system, for example when suspicious activity is detected in their session.

ReauthenticateProviders are similiar to `AuthenticationProvider` except they receive a user instance rather than an identifer for a user.

A successful reauthentication should return True while failures should return None in order to give other providers an attempt run.

If a ReauthenticateProvider determines that reauthentication should immediately end, it may raise `StopAuthentication` to safely end the process.

An example:

```python
class LDAPReauthenticateProvider(ReauthenticateProvider):
    def __init__(self, ldap_client):
        self.ldap_client = ldap_client

    def reauthenticate(self, user, secret):
        user_dn = "uid={},ou=flaskbb,ou=org".format(user.username)
        try:
            self.ldap_client.bind_user(user_dn, secret)
            return True
        except Exception:
            return None
```

`reauthenticate`(*user*, *secret*)

This method is abstract.

Parameters

- **user** (*User*) – The current user instance
- **secret** (*str*) – The secret provided by the user

Returns True for a successful reauth, otherwise None

class flaskbb.core.auth.authentication.**PostReauthenticateHandler**
Used to post process successful reauthentication attempts.

PostAuthenticationHandlers are similar to *PostAuthenticationHandler*, including their ability to cancel a successful attempt by raising *StopAuthentication*

handle_post_reauth(*user*)
This method is abstract.

Parameters **user** (*User*) – The current user instance that passed the reauth attempt

class flaskbb.core.auth.authentication.**ReauthenticateFailureHandler**
Used to manager reauthentication failures in FlaskBB.

ReauthenticateFailureHandlers are similiar to *AuthenticationFailureHandler* except they receive the user instance rather than an indentifier for a user

handle_reauth_failure(*user*)
This method is abstract.

Parameters **user** (*User*) – The current user instance that failed the reauth attempt

Reauthentication Provided Implementations

class flaskbb.auth.services.reauthentication.**DefaultFlaskBBReauthProvider**
This is the default reauth provider in FlaskBB, it compares the provided password against the current user's hashed password.

class flaskbb.auth.services.reauthentication.**ClearFailedLoginsOnReauth**
Handler that clears failed login attempts after a successful reauthentication.

class flaskbb.auth.services.reauthentication.**MarkFailedReauth**
Failure handler that marks the failed reauth attempt as a failed login and when it occurred.

class flaskbb.auth.services.reauthentication.**PluginReauthenticationManager**(*plugin_manager*, *session*)
Default reauthentication manager for FlaskBB, it relies on plugin hooks to manage the reauthentication flow.

Exceptions

exception flaskbb.core.auth.authentication.**StopAuthentication**(*reason*)
Used by Authentication providers to halt any further attempts to authenticate a user.

Parameters **str** (*reason*) – The reason why authentication was halted

exception flaskbb.core.auth.authentication.**ForceLogout**(*reason*)
Used to forcefully log a user out.

Parameters **str** (*reason*) – The reason why the user was force logged out

Account Management

class flaskbb.core.auth.password.**ResetPasswordService**

Interface for managing the password reset experience in FlaskBB.

initiate_password_reset(*email*)

This method is abstract.

Used to send a password reset token to a user.

This method may raise a `ValidationError` when generating the token, such as when the user requests a reset token be sent to an email that isn't registered in the application.

Parameters **email** (`str`) – The email to send the reset request to.

reset_password(*token*, *email*, *new_password*)

This method is abstract.

Used to process a password reset token and handle resetting the user's password to the newly desired one. The token passed to this message is the raw, serialized token sent to the user.

This method may raise `TokenError` or `ValidationError` to communicate failures when parsing or consuming the token.

Parameters

- **token** (`str`) – The raw serialized token sent to the user
- **email** (`str`) – The email entered by the user at password reset
- **new_password** (`str`) – The new password to assign to the user

class flaskbb.core.auth.activation.**AccountActivator**

Interface for managing account activation in installations that require a user to activate their account before using it.

initiate_account_activation(*user*)

This method is abstract.

Used to extend an offer of activation to the user. This may take any form, but is recommended to take the form of a permanent communication such as email.

This method may raise `ValidationError` to communicate a failure when creating the token for the user to activate their account with (such as when a user has requested a token be sent to an email that is not registered in the installation or the account associated with that email has already been activated).

Parameters **user** (`User`) – The user that the activation request applies to.

activate_account(*token*)

This method is abstract.

Used to handle the actual activation of an account. The token passed in is the serialized token communicated to the user to use for activation. This method may raise `TokenError` or `ValidationError` to communicate failures when parsing or consuming the token.

Parameters **token** (`str`) – The raw serialized token sent to the user

Tokens

class flaskbb.core.tokens.**TokenActions**

Collection of token actions.

Note: This is just a class rather than an enum because enums cannot be extended at runtime which would limit the number of token actions to the ones implemented by FlaskBB itself and block extension of tokens by plugins.

class `flaskbb.core.tokens.`**`Token`**(*user_id*, *operation*)

Parameters

- **`user_id`** (*int*) –
- **`operation`** (*str*) – An operation taken from *TokenActions*

class `flaskbb.core.tokens.`**`TokenSerializer`**

`dumps`(*token*)
This method is abstract.

Used to transform a token into a string representation of it.

Parameters **`token`** (*Token*) –

Returns str

`loads`(*raw_token*)
This method is abstract

Used to transform a string representation of a token into an actual *Token* instance

Parameters **`raw_token`** (*str*) –

Returns token The parsed token

Return type `Token<flaskbb.core.tokens.Token>`

class `flaskbb.core.tokens.`**`TokenVerifier`**
Used to verify the validatity of tokens post deserialization, such as an email matching the user id in the provided token.

Should raise a *ValidationError* if verification fails.

`verify_token`(*token*, ***kwargs*)
This method is abstract.

Parameters

- **`token`** (*Token*) – The parsed token to verify
- **`kwargs`** – Arbitrary context for validation of the token

exception `flaskbb.core.tokens.`**`TokenError`**(*reason*)
Raised when there is an issue with deserializing a token. Has helper classmethods to ensure consistent verbiage.

Parameters **`reason`** (*str*) – An explanation of why the token is invalid

classmethod **`invalid`**()
Used to raise an exception about a token that is invalid due to being signed incorrectly, has been tampered with, is unparsable or contains an inappropriate action.

classmethod **`expired`**()
Used to raise an exception about a token that has expired and is no longer usable.

1.2.5 Settings

This part covers which setting fields are available. This is especially useful if you plan on develop a plugin a want to contribute to FlaskBB.

The available fields are shown below.

Note: For a full list of available methods, visit Settings Model.

class `flaskbb.utils.forms.`**`SettingValueType`**

Value Type	Rendered As	Parsed & Saved as
`string`	`wtforms.fields.StringField`	`str`
`integer`	`wtforms.fields.IntegerField`	`int`
`float`	`wtforms.fields.FloatField`	*`float`*
`boolean`	`wtforms.fields.BooleanField`	`bool`
`select`	`wtforms.fields.SelectField`	`list`
`selectmultiple`	`wtforms.fields.SelectMultipleField`	`list`

TODO

`string = 0`

`integer = 1`

`float = 3`

`boolean = 4`

`select = 5`

`selectmultiple = 6`

`flaskbb.management.`**`models`**

alias of *`flaskbb.management.models`*

1.3 Additional Information

1.3.1 Changelog

Here you can see the full list of changes between each release.

Version 2.0.2

Released July 15th, 2018

- Fix issue with declaring log config file path

Version 2.0.1

Released June 21st, 2018

- Fix issue where activation tokens would fail with an exception

Version 2.0.0

Released on May 16th, 2018.

- Improved management panel load time by requesting celery status async (PR #429)
- Migrated FlaskBB internal behavior to use plugin hook system (PRs #369, #413, #419, #423, #426, #435, #436)
- Migrated behavior in flaskbb.auth from living in route handlers and into services (PRs #421, #424)
- Improved emoji support (PR #417)
- Migrated private messages into a [plugin](https://github.com/sh4nks/flaskbb-plugins/tree/master/conversations) (PR #414)
- Fixed issue where user could not re-select having the default theme (PR #387)
- Fixed issue where a reinstall would attempt to drop the entire database instead of just the tables associated with FlaskBB (PR #364)
- Added ability to hide and unhide topics and posts, preventing unprivileged users from viewing them (PR #325)
- Fixed issue where password was not required when editing a user in the admin panel (PR #321)
- Migrated from Flask-Plugins to Pluggy as plugin system for plugins. Plugins are now loaded via entry points and thus have to be installed into the same environment as FlaskBB. During this migration we also moved the [portal plugin](https://github.com/sh4nks/flaskbb-plugins) into its own python package which can be installed via `pip install flaskbb-plugin-portal`. (PR #311)
- Adds the functionality to "soft delete" posts and topics. (PR #325)
- Improve logging. (PR #327)
- Prefixes migrations with a timestamp which will sort them by creation date. (PR #353)
- Transform views into Class-based Views. (PR #324)
- Drop the tables instead of dropping the database when uninstalling FlaskBB. (PR #364)
- Create the database using SQLAlchemy's `db.create_all` and stamp it as 'latest' instead of going through all migrations.
- Fixes a bug that prevented users to activate their accounts via the form input.
- Fixes a translations bug that wouldn't take the territory code into account when choosing a language (#299).
- Fixes a bug which would not show all conversations in the conversations view.
- Fixes a bug that made a forum section inaccessible when the `last_post_id` of a topic was set to None.
- Various translations updated.
- Multiple permission fixes and various other fixes.

Version 1.0

Released on May 5th, 2017.

- First release of FlaskBB

1.3.2 License

- search

Python Module Index

Index