
Flask-Blogging Documentation

Release 0.4.1

Gouthaman Balaraman

September 16, 2015

1	Quick Start Example	3
2	Configuring your Application	5
3	Configuration Variables	7
4	Blog Views	9
5	Permissions	11
6	Screenshots	13
6.1	Blog Page	14
6.2	Blog Editor	14
7	Useful Tips	15
8	Release Notes	17
9	Compatibility Notes	19
10	API Documentation	21
10.1	Module contents	21
10.2	Submodules	21
10.3	flask_blogging.engine module	21
10.4	flask_blogging.processor module	21
10.5	flask_blogging.sqlastorage module	21
10.6	flask_blogging.storage module	21
10.7	flask_blogging.views module	21
10.8	flask_blogging.forms module	21
11	Contributors	23

Flask-Blogging is a Flask extension for adding Markdown based blog support to your site. It provides a flexible mechanism to store the data in the database of your choice. It is meant to work with the authentication provided by packages such as [Flask-Login](#) or [Flask-Security](#).

The philosophy behind this extension is to provide a lean app based on Markdown to provide blog support to your existing web application. This is contrary to some other packages such as [Flask-Blog](#) that are just blogs. If you already have a web app and you need to have a blog to communicate with your user or to promote your site through content based marketing, then Flask-Blogging would help you quickly get a blog up and running.

Out of the box, Flask-Blogging has support for the following:

- Bootstrap based site
- Markdown based blog editor
- Models to store blog
- Authentication of User's choice
- Sitemap, ATOM support
- Disqus support for comments
- Google analytics for usage tracking
- Permissions enabled to control which users can create/edit blogs
- Integrated Flask-Cache based caching for optimization
- Well documented, tested, and extensible design

- *Quick Start Example*
- *Configuring your Application*
- *Configuration Variables*
- *Blog Views*
- *Permissions*
- *Screenshots*
 - *Blog Page*
 - *Blog Editor*
- *Useful Tips*
- *Release Notes*
- *Compatibility Notes*
- *API Documentation*
 - *Module contents*
 - *Submodules*
 - *flask_blogging.engine module*
 - *flask_blogging.processor module*
 - *flask_blogging.sqlastorage module*
 - *flask_blogging.storage module*
 - *flask_blogging.views module*
 - *flask_blogging.forms module*
- *Contributors*

Quick Start Example

```

from flask import Flask, render_template_string, redirect
from sqlalchemy import create_engine, MetaData
from flask.ext.login import UserMixin, LoginManager, \
    login_user, logout_user
from flask.ext.blogging import SQLAlchemyStorage, BloggingEngine

app = Flask(__name__)
app.config["SECRET_KEY"] = "secret" # for WTF-forms and login
app.config["BLOGGING_URL_PREFIX"] = "/blog"
app.config["BLOGGING Disqus SITENAME"] = "test"
app.config["BLOGGING_SITEURL"] = "http://localhost:8000"

# extensions
engine = create_engine('sqlite:///tmp/blog.db')
meta = MetaData()
sql_storage = SQLAlchemyStorage(engine, metadata=meta)
blog_engine = BloggingEngine(app, sql_storage)
login_manager = LoginManager(app)
meta.create_all(bind=engine)

# user class for providing authentication
class User(UserMixin):
    def __init__(self, user_id):
        self.id = user_id

    def get_name(self):
        return "Paul Dirac" # typically the user's name

@login_manager.user_loader
@blog_engine.user_loader
def load_user(user_id):
    return User(user_id)

index_template = """
<!DOCTYPE html>
<html>
  <head> </head>
  <body>
    {% if current_user.is_authenticated() %}
      <a href="/logout/">Logout</a>
    {% else %}
      <a href="/login/">Login</a>
    {% endif %}
  
```

```
        &nbsp;&nbsp;<a href="/blog/">Blog</a>
        &nbsp;&nbsp;<a href="/blog/sitemap.xml">Sitemap</a>
        &nbsp;&nbsp;<a href="/blog/feeds/all.atom.xml">ATOM</a>
    </body>
</html>
"""

@app.route("/")
def index():
    return render_template_string(index_template)

@app.route("/login/")
def login():
    user = User("testuser")
    login_user(user)
    return redirect("/blog")

@app.route("/logout/")
def logout():
    logout_user()
    return redirect("/")

if __name__ == "__main__":
    app.run(debug=True, port=8000, use_reloader=True)
```

The key components required to get the blog hooked is explained below.

Configuring your Application

The *BloggingEngine* class is the gateway to configure blogging support to your web app. You should create the *BloggingEngine* instance like this:

```
blogging_engine = BloggingEngine()
blogging_engine.init_app(app, storage)
```

You also need to pick the storage for blog. That can be done as:

```
from sqlalchemy import create_engine, MetaData

engine = create_engine("sqlite:///tmp/sqlite.db")
meta = MetaData()
storage = SQLAStorage(engine, metadata=meta)
meta.create_all(bind=engine)
```

Here we have created the storage, and created all the tables in the metadata. Once you have created the blogging engine, storage, and all the tables in the storage, you can connect with your app using the *init_app* method as shown below:

```
blogging_engine.init_app(app, storage)
```

If you are using Flask-Sqlalchemy, you can do the following:

```
from flask.ext.sqlalchemy import SQLAlchemy

db = SQLAlchemy(app)
storage = SQLAStorage(db=db)
db.create_all()
```

One of the changes in version 0.3.1 is the ability for the user to provide the *metadata* object. This has the benefit of the table creation being passed to the user. Also, this gives the user the ability to use the common *metadata* object, and hence helps with the tables showing up in migrations while using Alembic.

As of version 0.4.0, Flask-Cache integration is supported. In order to use caching in the blogging engine, you need to pass the *Cache* instance to the *BloggingEngine* as:

```
from flask.ext.cache import Cache
from flask.ext.blogging import BloggingEngine

blogging_engine = BloggingEngine(app, storage, cache)
```

Flask-Blogging lets the developer pick the authentication that is suitable, and hence requires her to provide a way to load user information. You will need to provide a *BloggingEngine.user_loader* callback. This callback is used to load

the user from the *user_id* that is stored for each blog post. Just as in Flask-Login, it should take the *unicode user_id* of a user, and return the corresponding user object. For example:

```
@blogging_engine.user_loader
def load_user(userid):
    return User.get(userid)
```

For the blog to have a readable display name, the `User` class must implement either the `get_name` method or the `__str__` method.

The `BloggingEngine` accepts an optional `extensions` argument. This is a list of Markdown extensions objects to be used during the markdown processing step.

The `BloggingEngine` also accepts `post_processor` argument, which can be used to provide a custom post processor object to handle the processing of Markdown text. An ideal way to do this would be to inherit the default `PostProcessor` object and override custom methods. There is a `custom_process` method that can be overridden to add extra functionality to the post processing step.

In version 0.4.1 and onwards, the `BloggingEngine` object can be accessed from your app as follows:

```
engine = app.extensions["blogging"]
```

The engine method also exposes a `get_posts` method to get the recent posts for display of posts in other views.

In earlier versions the same can be done using the key `FLASK_BLOGGING_ENGINE` instead of `blogging`. The use of `FLASK_BLOGGING_ENGINE` key will be deprecated moving forward.

Configuration Variables

The Flask-Blogging extension can be configured by setting the following app config variables. These arguments are passed to all the views. The keys that are currently supported include:

- `BLOGGING_SITENAME` (*str*): The name of the blog to be used as the brand name. This is also used in the feed heading. (default “Flask-Blogging”)
- `BLOGGING_SITEURL` (*str*): The url of the site.
- `BLOGGING_RENDER_TEXT` (*bool*): Value to specify if the raw text should be rendered or not. (default `True`)
- `BLOGGING_DISQUS_SITENAME` (*str*): Disqus sitename for comments. A `None` value will disable comments. (default `None`)
- `BLOGGING_GOOGLE_ANALYTICS` (*str*): Google analytics code for usage tracking. A `None` value will disable google analytics. (default `None`)
- `BLOGGING_URL_PREFIX` (*str*) : The prefix for the URL of blog posts. A `None` value will have no prefix (default `None`).
- `BLOGGING_FEED_LIMIT` (*int*): The number of posts to limit to in the feed. If `None`, then all are shown, else will be limited to this number. (default `None`)
- `BLOGGING_PERMISSIONS` (*bool*): if `True`, this will enable permissions for the blogging engine. With permissions enabled, the user will need to have “blogger” `Role` to edit or create blog posts. Other authenticated users will not have blog editing permissions. The concepts here derive from `Flask-Principal` (default `False`)
- `BLOGGING_POSTS_PER_PAGE` (*int*): This sets the default number of pages to be displayed per page. (default 10)
- `BLOGGING_CACHE_TIMEOUT` (*int*): The timeout in seconds used to cache the blog pages. (default 60)

Blog Views

There are various views that are exposed through Flask-Blogging. The URL for the various views are:

- `url_for('blogging.index')` (GET): The index blog posts with the first page of articles.
- `url_for('blogging.page_by_id', post_id=<post_id>)` (GET): The blog post corresponding to the `post_id` is retrieved.
- `url_for('blogging.posts_by_tag', tag=<tag_name>)` (GET): The list of blog posts corresponding to `tag_name` is returned.
- `url_for('blogging.posts_by_author', user_id=<user_id>)` (GET): The list of blog posts written by the author `user_id` is returned.
- `url_for('blogging.editor')` (GET, POST): The blog editor is shown. This view needs authentication and permissions (if enabled).
- `url_for('blogging.delete', post_id=<post_id>)` (POST): The blog post given by `post_id` is deleted. This view needs authentication and permissions (if enabled).
- `url_for('blogging.sitemap')` (GET): The sitemap with a link to all the posts is returned.
- `url_for('blogging.feed')` (GET): Returns ATOM feed URL.

The view can be easily customised by the user by overriding with their own templates. The template pages that need to be customized are:




- `blogging/index.html`: The blog index page used to serve index of posts, posts by tag, and posts by author
- `blogging/editor.html`: The blog editor page.
- `blogging/page.html`: The page that shows the given article.
- `blogging/sitemap.xml`: The sitemap for the blog posts.

Permissions

In version 0.3.0 Flask-Blogging, enables permissions based on Flask-Principal. This addresses the issue of controlling which of the authenticated users can have access to edit or create blog posts. Permissions are enabled by setting `BLOGGING_PERMISSIONS` to `True`. Only users that have access to Role “blogger” will have permissions to create or edit blog posts.

Screenshots

6.1 Blog Page

 Delete
  Edit
  New


Dirac Equation


Posted by *Paul Dirac* on 03 Jun, 2015



In particle physics, the Dirac equation is a relativistic wave equation derived by British physicist Paul Dirac in 1928. In its free form, or including electromagnetic interactions, it describes all spin-1/2 massive particles, for which parity is a symmetry, such as electrons and quarks, and is consistent with both the principles of quantum mechanics and the theory of special relativity,[1] and was the first theory to account fully for special relativity in the context of quantum mechanics.


Dirac's Equation is given as:

$$(\beta mc^2 + c(\alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3)) \psi(x, t) = i\hbar \frac{\partial \psi(x, t)}{\partial t}$$

 PHYSICS

0 Comments
 test
  Gouthaman Balar...

 Recommend
  Share
 Sort by Best



Be the first to comment.

6.2 Blog Editor

Title

Dirac Equation

Useful Tips

- **Postgres using psycopg2:** If you use psycopg2 driver for Postgres while using the SQLAlchemy you would need to have autocommit turned on while creating the engine:

```
create_engine("postgresql+psycopg2://postgres:@localhost/flask_blogging",
              isolation_level="AUTOCOMMIT")
```

- **Migrations with Alembic:** (Applies to versions 0.3.0 and earlier) If you have migrations part of your project using Alembic, or extensions such as Flask-Migrate which uses Alembic, then you have to modify the Alembic configuration in order for it to ignore the Flask-Blogging related tables. If you don't set these modifications, then every time you run migrations, Alembic will not recognize the tables and mark them for deletion. And if you happen to upgrade by mistake then all your blog tables will be deleted. What we will do here is ask Alembic to exclude the tables used by Flask-Blogging. In your alembic.ini file, add a line:

```
[alembic:exclude]
tables = tag, post, tag_posts, user_posts
```

If you have a value set for table_prefix argument while creating the SQLAlchemy, then the table names will contain that prefix in their names. In which case, you have to use appropriate names in the table names.

And in your env.py, we have to mark these tables as the ones to be ignored.

```
def exclude_tables_from_config(config_):
    tables_ = config_.get("tables", None)
    if tables_ is not None:
        tables = tables_.split(",")
    return tables

exclude_tables = exclude_tables_from_config(config.get_section('alembic:exclude'))

def include_object(object, name, type_, reflected, compare_to):
    if type_ == "table" and name in exclude_tables:
        return False
    else:
        return True

def run_migrations_online():
    """Run migrations in 'online' mode.

    In this scenario we need to create an Engine
    and associate a connection with the context.

    """
```

```
engine = engine_from_config(
    config.get_section(config.config_ini_section),
    prefix='sqlalchemy.',
    poolclass=pool.NullPool)

connection = engine.connect()
context.configure(
    connection=connection,
    target_metadata=target_metadata,
    include_object=include_object,
    compare_type=True
)

try:
    with context.begin_transaction():
        context.run_migrations()
finally:
    connection.close()
```

In the above, we are using `include_object` in `context.configure(...)` to be specified based on the `include_object` function.

Release Notes

- **Version 0.4.1**

Released September 16, 2015

- Added javascript to center images in blog page
- Added method in blogging engine to render post and fetch post.

- **Version 0.4.0**

Released July 26, 2015

- Integrated Flask-Cache to optimize blog page rendering
- Fixed a bug where anonymous user was shown the new blog button

- **Version 0.3.2:**

Released July 20, 2015

- Fixed a bug in the edit post routines. The edited post would end up as a new one instead.

- **Version 0.3.1:**

Released July 17, 2015

- The `SQLAlchemyStorage` accepts metadata, and `SQLAlchemy` object as inputs. This adds the ability to keep the blogging table metadata synced up with other models. This feature adds compatibility with `Alchemic` autogenerate.
- Update docs to reflect the correct version number.

- **Version 0.3.0:**

Released July 11, 2015

- `Permissions` is a new feature introduced in this version. By setting `BLOGGING_PERMISSIONS` to `True`, one can restrict which of the users can create, edit or delete posts.
- Added `BLOGGING_POSTS_PER_PAGE` configuration variable to control the number of posts in a page.
- Documented the url construction procedure.

- **Version 0.2.1:**

Released July 10, 2015

- `BloggingEngine` `init_app` method can be called without having to pass a `storage` object.
- Hook tests to `setup.py` script.

- **Version 0.2.0:**

Released July 6, 2015

- BloggingEngine configuration moved to the app config setting. This breaks backward compatibility. See compatibility notes below.
- Added ability to limit number of posts shown in the feed through app configuration setting.
- The `setup.py` reads version from the module file. Improves version consistency.

- **Version 0.1.2:**

Released July 4, 2015

- Added Python 3.4 support

- **Version 0.1.1:**

Released June 15, 2015

- Fixed PEP8 errors
- Expanded SQLAStorage to include Postgres and MySQL flavors
- Added `post_date` and `last_modified_date` as arguments to the `Storage.save_post(...)` call for general compatibility

- **Version 0.1.0:**

Released June 1, 2015

- Initial Release
- Adds detailed documentation
- Supports Markdown based blog editor
- Has 90% code coverage in unit tests

Compatibility Notes

- **Version 0.4.1:**

The documented way to get the blogging engine from app is using the key `blogging` from `app.extensions`.

- **Version 0.3.1:**

The `SQLAStorage` will accept metadata and set it internally. The database tables will not be created automatically. The user would need to invoke `create_all` in the metadata or `SQLAlchemy` object in `Flask-SQLAlchemy`.

- **Version 0.3.0:**

- In this release, the templates folder was renamed from `blog` to `blogging`. To override the existing templates, you will need to create your templates in the `blogging` folder.
- The blueprint name was renamed from `blog_api` to `blogging`.

- **Version 0.2.0:**

In this version, `BloggingEngine` will no longer take `config` argument. Instead, all configuration can be done through app config variables. Another `BloggingEngine` parameter, `url_prefix` is also available only through config variable.

API Documentation

10.1 Module contents

10.2 Submodules

10.3 flask_blogging.engine module

10.4 flask_blogging.processor module

10.5 flask_blogging.sqlastorage module

10.6 flask_blogging.storage module

10.7 flask_blogging.views module

10.8 flask_blogging.forms module

Contributors

- Gouthaman Balaraman
- adilosa