breeze

Komeil Parseh

Jul 01, 2022

CONTENTS

CHAPTER

ONE

DEVELOPMENT HAS STOPPED FOR NOW AND MAY BE DONE AGAIN IN THE FUTURE.

the breeze is a fun web application for practising with Flask, SQLalchemy and a few other things that try to be similar to **Twitter**!

1.1 Run the breeze

- breeze need to python 3.7 or upper

first your need to install breeze requirements:

```
pip install -e .
```

then your add breeze to your environment and run breeze:

- bash:

```
export FLASK_APP=breeze
flask run
```

- fish

```
set -x FLASK_APP breeze
flask run
```

- cmd

```
set FLASK_APP=breeze
flask run
```

- powershell

```
$env:FLASK_APP = "breeze"
flask run
```

now you can visit your breeze on your browser: http://localhost:5000/

1.2 TODOs

this section is breeze todos. we use this section to keep track of all the todos in breeze.

- **tags**:

we use tags to categorize todos. on the future, maybe we will use tools to help you to categorize todos but for now, you can just add custom tags to your todos.

- ~[async]: todos that are related to async programming.~
- [auth]: authentication related like login, logout, register, etc.
- [db]: database related like migration, seeding, etc.
- [frontend]: frontend related like jinja2, pages, etc.
- [mention]: like twitter mention
- [notification]: notification related like notification sys, etc.
- [post]: post(or comment) related like post, post edit, etc.
- [profile]: profile related like profile, profile edit, etc.
- [search]: search related like search engine, etc.
- [utils]: utils related like check time, etc.

  issues after review may be add to here.

- [X] use flask-wtf for form validation
- [x] add user authentication
  - [X] auth with other services like Google, GitHub, Twitter, etc.[auth]
    * [X] github(OAuth2)[auth]
    * [ ] Google(OAuth2)[auth]
    * [X] Discord(OAuth2)[auth]
  - [ ] add page for check user OAuth is valid or not[auth][frontend]
  - [ ] user can change their data like password, email, etc.[setting]
  - [ ] user can delete their data[setting]
  - [X] add check for duplicate username, email, etc.[db]
  - [ ] add verification for emails with link, send code, etc.[auth]
  - [X] use Captcha for verification like reCaptcha, etc.[auth]

    flask-wtf by defult use recaptcha v2
- [x] add user profile
  - [X] support gravatar[profile]
  - [ ] support cutsom avatar[profile][setting]
  - [ ] support custom background[profile][setting]
  - [ ] user can edit their profile and add their bio and info[profile]
    * [ ] bio: short description about yourself[profile]
    * [ ] info: many datas like location, birthday, etc.[profile]

- [X] create a post or comment[post]
  - [X] delete post or comment[post]
  - [ ] like post or comment[post][db]
  - [ ] edit post or comment[post]
    * [ ] time limit for edit[post][utils]
  - [ ] retweet post or comment[post]
    * [ ] add description for retweet[post]
  - [X] add comment for post[post]
  - [ ] add comment for comment[post]
  - [ ] mention user in post or comment[post][mention]
- [ ] ~use `Asynchronous` in project[async]~

  ~for more information, please visit flask docs (new in 2.x version)~ why this won't do? because flask for now is not support async fully.(see issue #13)
- [ ] tag system
  - [ ] add tag for post or comment[post]
  - [ ] add tag for user bio[profile]
  - [ ] search posts(or other things) by tag[post][tag][search]
- [ ] add a search system[search]
  - [ ] use tag system for search[search]
  - [ ] search by username[search]
  - [ ] search by text[search]

  we can use algolia or elasticsearch but if we use flask, we need to use wtf-forms to handle the form.
- [ ] add a notification system[notification]
  - [ ] user can allow or disallow notifications for a post(or other things)[notification]

  when user like a post or comment, we can send a notification to the user who post or comment the post or comment.
- [ ] add a message system(like twitter DM)[message]
  - [ ] user can send a message to another user[message]
  - [ ] ban and onban user[message][auth]
  - [ ] user can delete/edit their message[message]
- **frontend**:

  I'm not sure if we need to use react or vue for frontend. we can use pure flask and bootstrap for frontend. of course, we can use flask-admin for admin panel…
- [ ] fix post and comment form[frontend]
  - [ ] fix post form[frontend]
  - [ ] fix comment form[frontend]
  - bootstrap or other css framework

- [ ] fix user profile[frontend]
  - [ ] use bootstrap and other css framework(if need) for improve user profile page[frontend]
- [ ] fix auth page[frontend]
- [ ] add buttom or otherthings
  - [ ] for remove a post or comment[frontend]
- **docs and api**:
- [X] add docs for api[docs]
- [ ] fix problem on show TOODs on the documention page[docs]

1.2.1 Configuration of the breeze

breeze has a configuration class in `breeze.config.Config` but we can't edit this file because in that case our keys are placed in the code and this is not a good practice. So we have to create a file *.env* at the root of the project and we can edit it.

`.env` file is a file that contains environment variables. It is a file that is used to store configuration variables. python dotenv is a library that allows us to read and write environment variables from a file for more information visit here.

Warning: `.env` file not commited to the repository because it's name in the *.gitignore* file. we not suggest to commit this file because it's not a good practice.

for example *.env* file contains the following:

```
RECAPTCHA_PUBLIC_KEY=<your_recaptcha_public_key>
RECAPTCHA_PRIVATE_KEY=<your_recaptcha_private_key>
GITHUB_CLIENT_ID=<your_github_client_id>
GITHUB_CLIENT_SECRET=<your_github_client_secret>
DISCORD_CLIENT_ID=<your_discord_client_id>
DISCORD_CLIENT_SECRET=<your_discord_client_secret>
```

1.2.2 API

Authentication

Auth is a module that provides a simple authentication system for Flask.

auth base class

Auth with other ways(e.g. github, etc)

Blueprints

This section lists our Blueprint APIs. A Blueprint is a way to organize a group of related views and other codes. Rather than registering views and other code directly with an application, they are registered with a blueprint. Then the blueprint is registered with the application when it is available in the factory function.

For more information see Modular Applications with Blueprints

Authentication

Index

Posts

Config

Exception

exception breeze.exc.**BreezeException**(*message*)

Base class for all Breeze exceptions inherited from `Exception`

Attr

`message` (str): the exception message

exception breeze.exc.**EmptyError**(*message*)

Exception raised when a object is empty. inherited from *breeze.exc.BreezeException*

exception breeze.exc.**InvalidUsage**(*message*)

Exception raised when a user does not have permission to perform an action. inherited from *breeze.exc.BreezeException*

exception breeze.exc.**PermissionError**(*message*)

Exception raised when a user does not have permission to perform an action. inherited from *breeze.exc.BreezeException*

Models

class breeze.models.**Post**(***kwargs*)

Posts table on db

inherited from `flask_sqlalchemy.SQLAlchemy`

Methods

save(): save post to db *delete()*: delete post from db `update()`: update post from db

delete()

Delete post from db

save()

Save post to db

class breeze.models.**Tag**(***kwargs*)

class breeze.models.**User**(***kwargs*)

Users table on db

inherited from `flask_sqlalchemy.SQLAlchemy`

Raises

breeze.exc.EmptyError: if password is empty then raise this exception *breeze.exc.PermissionError*: if user not have permission to perform an action then raise this exception

Methods

save(): save user to db *delete()*: delete user from db `update()`: update user from db

Attributes
`username` (*str*): user username `email` (*str*): user email `password` (*str*): user password `profile_image` (*str*): user profile image

this is a url to the image and by default it is a gravatar image

check_password(*password*)

Check user password

Args
`password` (*hash*): User password to confirm delete

Returns
bool: True if password is correct, False otherwise

delete(*confirm_password*)

Delete user from db

Args
`confirm_password` (*str*): user password to confirm delete

Raises
breeze.exc.PermissionError

save()

Save user to db

Raises
breeze.exc.EmptyError

Utils

breeze.utils.**check_hash**(*hash*, *password*)

Checks if a password matches a hash.

Args
`password` (*str*): password to check `hash` (*bytes*): hash to check

Returns
bool: True if the password matches the hash, False otherwise

Parameters

- **hash** (*bytes*) –
- **password** (*str*) –

Return type
Optional[bool]

breeze.utils.**gen_random_string**(*length*)

Generates a random string.

Args
`length` (*int*): length of the string

Returns
str: random string

Parameters
length (*int*) –

Return type
str

`breeze.utils.`**`get_current_time`**`()`

Returns the current time in UTC.

Returns
`datetime.datetime`

Return type
datetime

`breeze.utils.`**`get_image_from_gravatar`**`(`*email*`)`

Gets an image from gravatar.

first, this function Generate md5 of the user's email and next returned this hash with gravatar URL

if this URL is not found(404) gravatar shows the default avatar else show self users avatar from gravatar

Args
`email` (*str*): email to get the image from gravatar

Returns
str: url to the image

Parameters
`email` (*`str`*) –

Return type
str

`breeze.utils.`**`normalise`**`(`*string*`)`

Normalise a string.

Args
`string` (*str*): input string

Returns
str: normalised string

Parameters
`string` (*`Optional[Union[str, bytes]]`*) –

Return type
str

`breeze.utils.`**`normalise_email`**`(`*email*`)`

check and normalise user emails

pattern:

a regex pattern to check for normalised email,

part of the *pattern* is the email domain:

1. **username(**username*@domain.ex*)**

   can be digested or litter::
   breeze1234

2. **domain(*username@*domain*.ex*)**

   can be only litters::
   email

3. **extension(*username@domain.*extension**)

    can only litter with 2, 3 or 4 length::
    com or *io* or *wiki*

if the user's email matches the pattern return email

Args
`email` (*str*): input email

Returns
str: normalized email *bool*: flase if email is invalid or None

Parameters
`email` (*`Optional[str]`*) –

Return type
Union[str, bool]

`breeze.utils.`**`string_to_hash`**(*string*)

Converts a string to a hash.

Args
`string` (*str*): input string

Returns
str: hash of the string

Parameters
`string` (*`str`*) –

Return type
str

1.2.3 changelog

1.2.4 Changes

5.x

- support discord oauth2
- use coockie to store user info
- add http error pages(404, 405)
- support github oauth2
- use **wtform** for validation forms
    - add **recaptcha(v2)** to login and register form

4.x and lower

unfortunately, I forgot to write the changes before this version, but maybe in the future it will be completed here by browsing the git commitments, but not for now.

1.2.5 contributing to the project

this project is open source, and we welcome contributions from the community in any way. your contributions will be reviewed by a team member and will be integrated into the project as soon as possible.

support questions

please don't hesitate to ask questions, and we will try to answer them as soon as possible. your can ask questions in the github-discussion Please don't use the issue tracker for this. The issue tracker is a tool to address bugs and feature requests in Breeze itself.

Reporting issues

Include the following information when reporting issues:

- Describe what you expected to happen.
- Describe what you actually did.
- List your Python, Flask and SQLalchemy version, OS, and other relevant details.

Submitting pull requests

your can work on any issue without assigning it to a member. if your like work on a feature, you can see *TODOs* for find TODOs. but if you like to work on a bug, you can report bug in the issue tracker and work on it.

Include the following in your patch:

- use black to format your code. This and other tools will run automatically if you install pre-commit using the instructions below.
- Include tests if your patch adds or changes code. Make sure the test fails without your patch.
- Update any relevant docs pages and docstrings. Docs pages and docstrings should be wrapped at 72 characters.

work on a feature or fix a bug

steps:

- 1: fork the project and clone the repo.
- 2: Clone the main repository locally.

```
git clone https://github.com/mmdbalkhi/breeze
cd breeze
```

- 3: Add your fork as a remote to push your work to. Replace {username} with your username. This names the remote “fork”, the default Pallets remote is “origin”.

```
git remote add fork https://github.com/{username}/breeze
```

- 4: install and set-up virtualenv.
- *nix

```
    python3 -m venv env
    . env/bin/activate
```

- windows

```
    > py -3 -m venv env
    > env\Scripts\activate
```

- 4.2: Upgrade pip and setuptools.

```
    pip install --upgrade pip setuptools
```

- 4.3: Install the dependencies, then install breeze in editable mode.

```
pip install -r requirements.txt && pip install -e .
```

- 4.4: install pre-commit hooks

```
pip install pre-commit
pre-commit install
```

- 5: create a branch

```
git fetch origin
git checkout -b your-branch-name origin/main
```

- 6: Using your favorite editor, make your changes, committing as you go.
  - 6.2: we use this structure for our commits:
    - docs: `docs: <commit message> <filename>`
    - fix bug: `bug: <commit message> <filename>`
    - feature: `feature: <commit message> <filename>`
    - refactor: `refactor: <commit message> <filename>`
    - test: `test: <commit message> <filename>`
    - chore: `chore: <commit message> <filename>`
    - style: `style: <commit message> <filename>`
    - revert: `revert: <commit message> <filename>`
    - work in progress: `wip: <commit message> <filename>`

    we use this structure for easier read of the commit messages and git log.
  - 6.3: if your change many files on a commit you should split this commit into smaller commits.
  - 6.4: if your commit message is too long, you should use a short commit message on title and a longer commit message on body:

```
$ git commit -m "<tag>: <short commit message> <filename>" -m "<long commit message>"
```

- 6.2 Include tests that cover any code changes you make. Make sure the test fails without your patch. Run the tests as described below.
- 6.3 Push your commits to your fork on GitHub and create a pull request. Link to the issue being addressed with fixes #123 in the pull request.

```
$ git push --set-upstream fork your-branch-name
```

- 7: Run the tests.

```
pytest
```

- 7.2: Running test coverage

Generating a report of lines that do not have test coverage can indicate where to start contributing. Run `pytest` using `coverage` and generate a report.

```
pip install coverage
coverage run -m pytest
coverage html
```

Open `htmlcov/index.html` in your browser to explore the report.

Read more about coverage and pytest.

Building the docs

Build the docs in the docs directory using `Sphinx` .

```
cd docs
make html
```

Open `_build/html/index.html` in your browser to view the docs.

Read more about Sphinx.

1.2.6 License

MIT License

PYTHON MODULE INDEX

INDEX