breeze

Komeil Parseh

May 26, 2022

DEVELOPMENT

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

CHAPTER

ONE

RUN THE BREEZE

- breeze need to python 3.7 or upper

first your need to install breeze requirements:

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

then your add breeze to your environment and run breeze:

- bash:

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

- fish

```
set -x FLASK_APP myapp
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

1.1 API

class breeze.**Config**

A base configuration class from which other configuration classes inherit. for use in `flask.config.Config`

breeze.**create_app**()

create a flask application with application Factory pattern application Factory

Returns `flask.Flask`: flask application

1.1.1 Auth

class breeze.auth.**Auth**

Authentication class for Breeze

Attributes `User` (class): User class

property current_user

Get the current user

Returns `flask.g`: flask global object

get_user(*user_id*)

Get the user by id

Args *user_id* (`breeze.User.id()`)

Returns `breeze.User`: user row in db

init_app(*app*)

Initialize the authentication class with the app

Args `app` (`flask.Flask`): flask application

property is_authenticated

Check if the user is authenticated

Returns `bool`: if current user is authenticated

login(*user*)

Login the user to the session

Args *User* (:class`breeze.User`): user row in db

logout()

Logout the user from the session

register(*user*)

Register the user to the db

Args *user* (`breeze.User`): user row in db

1.1.2 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

1.1.3 Authentication

breeze.blueprints.auth.**load_logged_in_user**()

If a user id is stored in the session, load the user object from the database into `g.user`.

breeze.blueprints.auth.**login**()

Login a user to the system.

Returns

flask.Response:

- if the request method is GET, render the login template
- if the request method is POST, login the user and redirect to the index page

breeze.blueprints.auth.**logout**()

Logout a user from the system.

Returns `flask.Response`: redirect to the login page

breeze.blueprints.auth.**profile**()

Show the current user's profile.

Returns `flask.Response`: redirect to the user's profile to /u/username

breeze.blueprints.auth.**register**()

Register a new user to the system.

Returns

flask.Response:

- if the request method is GET, render the register template
- if the request method is POST, register the user and redirect to the login page

breeze.blueprints.auth.**user**(*username*)

Show a user's profile.

Args `username` (*str*): The username of the user to show

Returns `flask.Response`: The rendered template

Parameters **username** (*str*) –

1.1.4 Index

breeze.blueprints.index.**index**()

Render the index page

Returns `flask.Response`: The rendered template

1.1.5 Posts

breeze.blueprints.posts.**delete**(*id*)

Delete a post

Args `id` (*int*): The id of the post to delete

Returns `flask.Response`: The redirect to the home page

Parameters **id** (*int*) –

breeze.blueprints.posts.**new**()

Create a new post

Returns `flask.Response`: The rendered template if the request is GET, `flask.Response`: The redirect to the post if the request is POST

breeze.blueprints.posts.**show**(*id*)

Show a post

Args `id` (*int*): The id of the post to show

Returns `flask.Response`: The rendered template

Parameters **id** (*int*) –

1.1.6 Config

class breeze.config.**Config**

A base configuration class from which other configuration classes inherit. for use in `flask.config.Config`

1.1.7 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*

1.1.8 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

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*

1.1.9 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.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 License

MIT License

MIT License

Copyright (c) 2022 komeil Parseh

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PYTHON MODULE INDEX

b

INDEX

R

S

T

U