Flask-RBAC Documentation

Release 0.2.0

shonenada

Jan 13, 2021

Contents

Flask-RBAC provides a Role-based Access Control module in Flask applications. It can help you to control different role of users to access your website.

CHAPTER 1

Installation

Your can use pip to install Flask-RBAC:

```
$ pip install flask-rbac
```

CHAPTER 2

Quick Start

This documents show how to create Flask-RBAC extension easily and quickly.

2.1 Configuration Your Application

As same as many Flask extensions, you need to configuration your application:

```
from flask import Flask
from flask_rbac import RBAC

app = Flask(__name__)
rbac = RBAC(app)
```

or you can configuration using factory method:

```
from flask import Flask
from flask_rbac import RBAC

rbac = RBAC()

def create_app():
    app = Flask(__name__)

    rbac.init_app(app)

    return app
```

2.2 Mode Setting

There are two modes for Flask-RBAC, *RBAC_USE_WHITE* decide whether use white list to check the permission. And it set *False* to default.

RBAC_USE_WHITE = True	Only allowing rules can access the resources. This means, all deny rules and rules you did not add cannot access the resources.
RBAC_USE_WHITE = False	Only denying rules cannot access the resources. In case you set an allow rule, denying rules will also be automatically created for existing non-added roles in this route.

Change it using:

```
app.config['RBAC_USE_WHITE'] = True
```

2.3 Set Role Model

Flask-RBAC implements some methods need by Flask-RBAC in RoleMixin class. You can use RoleMixin as your role model:

```
class Role(RoleMixin):
    pass

anonymous = Role('anonymous')
```

However, if your application is working under SQLAlchemy, and you want to save the roles in database, you need to override the Role class to adapt your application, here is an example:

```
from flask_rbac import RoleMixin
from your_package.app import db

roles_parents = db.Table(
    'roles_parents',
    db.Column('role_id', db.Integer, db.ForeignKey('role.id')),
    db.Column('parent_id', db.Integer, db.ForeignKey('role.id'))
)

class Role(db.Model, RoleMixin):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20))
    parents = db.relationship(
        'Role',
        secondary=roles_parents,
        primaryjoin=(id == roles_parents.c.role_id),
        secondaryjoin=(id == roles_parents.c.parent_id),
        backref=db.backref('children', lazy='dynamic')
    )

    def __init__(self, name):
        RoleMixin.__init__(self)
        self.name = name

    def add_parent(self, parent):
        # You don't need to add this role to parent's children set,
        # relationship between roles would do this work automatically
        self.parents.append(parent)

    def add_parents(self, *parents):
        for parent in parents:
            self.add_parent(parent)
```

(continues on next page)

(continued from previous page)

```
    @staticmethod
    def get_by_name(name):
        return Role.query.filter_by(name=name).first()
```

After create role model, you can add your model to Flask-RBAC:

```
rbac.set_role_model(Role)
```

Or use decorator to set role model for Flask-RBAC:

```
@rbac.as_role_model
class Role(RoleMixin):
    # codes go here
```

2.4 Set User Model

Same as the RoleMixin, UserMixin implements some methods for Flask-RBAC, You can extend it directly:

```
from flask_rbac import UserMixin

class User(UserMixin):
    pass

a_user = User()
```

Well, if your application works under SQLAlchemy:

```
from flask_rbac import UserMixin
from your_package.app import db

users_roles = db.Table(
    'users_roles',
    db.Column('user_id', db.Integer, db.ForeignKey('user.id')),
    db.Column('role_id', db.Integer, db.ForeignKey('role.id'))
)

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(30), unllable=True)
    # Other columns
    roles = db.relationship(
        'Role',
        secondary=users_roles,
        backref=db.backref('roles', lazy='dynamic')
    )

    def add_role(self, role):
        self.roles.append(role)

    def add_roles(self, roles):
        for role in roles:
            self.add_role(role)
```

(continues on next page)

(continued from previous page)

```
    def get_roles(self):
        for role in self.roles:
            yield role
```

Same as role model, you should add user model to Flask-RBAC:

```
rbac.set_user_model(User)
```

Or using decorator:

```
@rbac.as_user_model
class User(UserMixin):
    # codes go here
```

2.5 Set User Loader

Flask-RBAC need to know who is current user, so it requires you to provide a function which tells it who is current user.

Flask-RBAC will load current user from Flask-Login if you have install it by default.

If you save current user in *flask.g*, here is an example for you:

```
from flask import g, current_app

@app.route('/signin', methods=['POST'])
@rbac.allow(['anonmous'], methods=['POST'])
def signin():
    # Sign in logic...
    g.current_user = user

def get_current_user():
    with current_app.request_context():
        return g.current_user

rbac.set_user_loader(get_current_user)
```

2.6 Set Access Rules

You can use *allow* and *deny* to add rules to Flask-RBAC:

```
@app.route('/')
@rbac.allow(['anonymous'], methods=['GET'])
def index():
    # your codes.
    pass

@app.route('/account/signin', methods=['GET', 'POST'])
@rbac.deny(['logged_user'], methods=['GET', 'POST'])
def signin():
    # show sign in page or handle sign in request.
    pass
```

The code above adding two rules:

- Allows user of *anonymous* role to *GET* /.
- Deny user of *logged_user* role to *GET* and *POST /account/signin*.

Flask itself assumes the name of the view function as the endpoint for the registered URL rule, that's why in rules validation by default we use the decorated function name to check against the endpoint of the input request. But, in case you specified a different endpoint or you use the decorators inside a blueprint or abstracted blueprints extensions like Flask-Admin you can directly specify to the decorator the endpoint used in your route.

```
@app.route('/signin', methods=['GET', 'POST'], endpoint='account.signin')
@rbac.deny(['logged_user'], methods=['GET', 'POST'],
    endpoint='account.signin')
def signin():
    # show sign in page or handle sign in request.
    pass
```

CHAPTER 3

API Reference

3.1 Flask.ext.rbac.RBAC

class flask_rbac.__init__.**RBAC**(*app=None*, ***kwargs*)

This class implements role-based access control module in Flask. There are two way to initialize Flask-RBAC:

```
app = Flask(__name__)
rbac = RBAC(app)
```

or:

```
rbac = RBAC
def create_app():
    app = Flask(__name__)
    rbac.init_app(app)
    return app
```

Parameters

- **app** – the Flask object
- **role_model** – custom role model
- **user_model** – custom user model
- **user_loader** – custom user loader, used to load current user
- **permission_failed_hook** – called when permission denied.

allow(*roles*, *methods*, *with_children=True*, *endpoint=None*)

This is a decorator function.

You can allow roles to access the view func with it.

An example:

```
@app.route('/website/setting', methods=['GET', 'POST'])
@rbac.allow(['administrator', 'super_user'], ['GET', 'POST'])
def website_setting():
    return Response('Setting page.')
```

Parameters

- **roles** – List, each name of roles. Please note that, *anonymous* is refered to anonymous. If you add *anonymous* to the rule, everyone can access the resource, unless you deny other roles.
- **methods** – List, each name of methods. methods is valid in ['GET', 'POST', 'PUT', 'DELETE']
- **with_children** – Whether allow children of roles as well. True by default.

as_role_model (*model_cls*)
A decorator to set custom model or role.

Parameters **model_cls** – Model of role.

as_user_model (*model_cls*)
A decorator to set custom model or user.

Parameters **model_cls** – Model of user.

deny (*roles*, *methods*, *with_children=False*, *endpoint=None*)
This is a decorator function.

You can deny roles to access the view func with it.

An example:

```
@app.route('/article/post', methods=['GET', 'POST'])
@rbac.deny(['anonymous', 'unactivated_role'], ['GET', 'POST'])
def article_post():
    return Response('post page.')
```

Parameters

- **roles** – List, each name of roles.
- **methods** – List, each name of methods. methods is valid in ['GET', 'POST', 'PUT', 'DELETE']
- **with_children** – Whether allow children of roles as well. True by default.

exempt (*view_func*)
Exempt a view function from being checked permission. It is useful when you are using white list checking.

Example:

```
@app.route('/everyone/can/access')
@rbac.exempt
def everyone_can_access():
    return 'Hello~'
```

Parameters **view_func** – The view function going to be exempted.

get_app(*reference_app=None*)
Helper method that implements the logic to look up an application.

has_permission(*method*, *endpoint*, *user=None*)
Return does the current user can access the resource. Example:

```
@app.route('/some_url', methods=['GET', 'POST'])
@rbac.allow(['anonymous'], ['GET'])
def a_view_func():
    return Response('Blah Blah...')
```

If you are not logged.

rbac.has_permission('GET', 'a_view_func') return True. *rbac.has_permission('POST', 'a_view_func')* return False.

Parameters

- **method** – The method wait to check.
- **endpoint** – The application endpoint.
- **user** – user who you need to check. Current user by default.

init_app(*app*)
Initialize application in Flask-RBAC. Adds (RBAC, app) to flask extensions. Adds hook to authenticate permission before request.

Parameters **app** – Flask object

set_hook(*hook*)
Set hook which called when permission is denied If you haven't set any hook, Flask-RBAC will call:

```
abort(403)
```

Parameters **hook** – Hook function

set_role_model(*model*)
Set custom model of role.

Parameters **model** – Model of role.

set_user_loader(*loader*)
Set user loader, which is used to load current user. An example:

```
from flask_login import current_user
rbac.set_user_loader(lambda: current_user)
```

Parameters **loader** – Current user function.

set_user_model(*model*)
Set custom model of User

Parameters **model** – Model of user

3.2 Flask.ext.rbac.model.RoleMixin

class flask_rbac.model.**RoleMixin**(*name=None*)
This provides implementations for the methods that Flask-RBAC wants the role model to have.

Parameters **name** – The name of role.

add_parent(*parent*)
Add a parent to this role, and add role itself to the parent's children set. you should override this function if neccessary.

Example:

```
logged_user = RoleMixin('logged_user')
student = RoleMixin('student')
student.add_parent(logged_user)
```

Parameters **parent** – Parent role to add in.

add_parents(**parents*)
Add parents to this role. Also should override if neccessary. Example:

```
editor_of_articles = RoleMixin('editor_of_articles')
editor_of_photonews = RoleMixin('editor_of_photonews')
editor_of_all = RoleMixin('editor_of_all')
editor_of_all.add_parents(editor_of_articles, editor_of_photonews)
```

Parameters **parents** – Parents to add.

classmethod get_all()
Return all existing roles

static get_by_name(*name*)
A static method to return the role which has the input name.

Parameters **name** – The name of role.

get_name()
Return the name of this role

3.3 Flask.ext.rbac.model.UserMixin

class flask_rbac.model.**UserMixin**(*roles=[]*)
This provides implementations for the methods that Flask-RBAC wants the user model to have.

Parameters **roles** – The roles of this user should have.

add_role(*role*)
Add a role to this user.

Parameters **role** – Role to add.

add_roles(**roles*)
Add roles to this user.

Parameters **roles** – Roles to add.

CHAPTER 4

Release Changes

4.1 Change Logs

- Introduce *endpoint* in *allow* and *deny* methods.
- Deprecate Python2.x, upgrade to Python3.x

4.1.1 Release 0.4.0 (Dec 8, 2020)

- Improve performance of *_check_permission*.

4.1.2 Release 0.3.0 (Mar 13, 2017)

- Fix *AnonymouseUserMixin*

4.1.3 Release 0.2.1 (Apr 4, 2014)

(Thanks suggestion from tonyseek)

- Make flask.ext.login.current_user to be default user.
- Add *as_role_model* and *as_user_model* decorator to set role and user model.

4.1.4 Release 0.2.0 (Jan 31, 2014)

- Add *exempt* method.

4.1.5 Release 0.1.0 (Jan 27, 2014)

- First public release.

CHAPTER 5

Author & Contributor

- Yaoda Liu <shonenada@gmail.com>
- tonyseek
- lixxu
- aurigadl
- trendsetter37
- hieuvo
- carlosgalvez-tiendeo

Index