Flask Simple Login

Maely Brandão

Aug 25, 2021

INDEX:

The simplest way to add login to Flask!

CHAPTER

ONE

SO WHY FLASK SIMPLE LOGIN?

Sometimes you need something simple for that small project or for prototyping.

CHAPTER

TWO

FLASK SIMPLE LOGIN

What it provides:

- Login and Logout forms and pages
- Function to check if user is logged-in
- Decorator for views
- Easy and customizable `login_checker`
- Basic auth for API endpoints

What it does not provide:

- Database Integration
- Password management
- API authentication with Token or JWT
- Role or user based access control

Of course you can easily implement all above by your own. Take a look at example.

CHAPTER

THREE

INSTALL

First install it from PyPI:

```
pip install flask_simplelogin
```

Flask Simple Login depends on Flask-WTF and WTForms, as well as on a *SECRET_KEY* set in your *app.config*.

CHAPTER

FOUR

QUICK START

```
from flask import Flask
from flask_simplelogin import SimpleLogin

app = Flask(__name__)
SimpleLogin(app)
```

That's it!

Now you have `/login` and `/logout` routes in your application.

The user name defaults to `admin` and the password defaults to `secret` — yeah that's not clever, let's see how to change it!

4.1 Configuring

Simplest way:

```
from flask import Flask
from flask_simplelogin import SimpleLogin

app = Flask(__name__)
app.config['SECRET_KEY'] = 'something-secret'
app.config['SIMPLELOGIN_USERNAME'] = 'chuck'
app.config['SIMPLELOGIN_PASSWORD'] = 'norris'

SimpleLogin(app)
```

That works, but is not so clever, let's use environment variables:

```
$ export SIMPLELOGIN_USERNAME=chuck
$ export SIMPLELOGIN_PASSWORD=norris
```

Now Simple Login will read and use them automatically:

```
from flask import Flask
from flask_simplelogin import SimpleLogin

app = Flask(__name__)
app.config['SECRET_KEY'] = 'something-secret'
SimpleLogin(app)
```

But what if you have more users and more complex authentication logic?

4.1.1 Using a custom login checker

```
from flask import Flask
from flask_simplelogin import SimpleLogin

app = Flask(__name__)
app.config['SECRET_KEY'] = 'something-secret'


def only_chuck_norris_can_login(user):
    """:param user: dict {'username': 'foo', 'password': 'bar'}"""
    if user.get('username') == 'chuck' and user.get('password') == 'norris':
       return True  # <--- Allowed
    return False  # <--- Denied


SimpleLogin(app, login_checker=only_chuck_norris_can_login)
```

4.1.2 Using a custom login, logout or home URL

Simple Login automatically loads Flask configurations prefixed with `SIMPLELOGIN_`, thus to set a custom login, logout or home URL:

```
from flask import Flask
from flask_simplelogin import SimpleLogin

app = Flask(__name__)
app.config['SECRET_KEY'] = 'something-secret'
app.config['SIMPLELOGIN_LOGIN_URL'] = '/signin/'
app.config['SIMPLELOGIN_LOGOUT_URL'] = '/exit/'
app.config['SIMPLELOGIN_HOME_URL'] = '/en/'

SimpleLogin(app)
```

4.1.3 Protection against open redirects

Flask Simple Login doesn't allow redirects to external URLs, but it can be configured to do so:

```
app.config["ALLOWED_HOSTS"] = ["myothersite.com"]
```

Then it is possible to redirect to an external URL in the `next=` parameter:

```
url_for('simplelogin.login', next='http://myothersite.com/')
```

4.1.4 Encrypting passwords

You can use the `from werkzeug.security import check_password_hash, generate_password_hash` utilities to encrypt passwords.

A working example is available in `manage.py` of example app

4.2 Usage

4.2.1 Checking if user is logged in

```
from flask_simplelogin import is_logged_in

if is_logged_in():
    # do things if anyone is logged in

if is_logged_in('admin'):
    # do things only if admin is logged in
```

4.2.2 Protecting your views

```
from flask_simplelogin import login_required

@app.route('/it_is_protected')
@login_required  # < --- simple decorator
def foo():
    return 'secret'

@app.route('/only_mary_can_access')
@login_required(username='mary')  # < --- accepts a list of names
def bar():
    return "Mary's secret"

@app.route('/api', methods=['POST'])
@login_required(basic=True)  # < --- Basic HTTP Auth for API
def api():
    # curl -XPOST localhost:5000/api -H "Authorization: Basic Y2h1Y2s6bm9ycmlz" -H
↪"Content-Type: application/json"
    # Basic-Auth takes base64 encripted username:password
    return jsonify(data='You are logged in with basic auth')

class ProtectedView(MethodView):  # < --- Class Based Views
    decorators = [login_required]
    def get(self):
        return "only loged in users can see this"
```

4.2.3 Protecting Flask Admin views

```
from flask_admin.contrib.foo import ModelView
from flask_simplelogin import is_logged_in


class AdminView(ModelView)
    def is_accessible(self):
        return is_logged_in('admin')
```

4.3 Customizing

4.3.1 Customizing templates

There are only one template to customize and it is called `login.html`. Example is:

```
{% extends 'base.html' %}
{% block title %}Login{% endblock %}
{% block messages %}
   {{super()}}
   {%if form.errors %}
     <ul class="alert alert-danger">
       {% for field, errors in form.errors.items() %}
         <li>{{field}} {% for error in errors %}{{ error }}{% endfor %}</li>
       {% endfor %}
     </ul>
   {% endif %}
{% endblock %}

{% block page_body %}
       <form action="{{ url_for('simplelogin.login', next=request.args.get('next', '/
↪')) }}" method="post">
            <div class="form-group">
            {{ form.csrf_token }}
            {{form.username.label}}<div class="form-control">{{ form.username }}</div>
↪<br>
            {{form.password.label}}<div class="form-control"> {{ form.password }}</
↪div><br>
            </form>
           <input type="submit" value="Send">
       </form>
{% endblock %}
```

Take a look at the example app.

And you can customize it in anyway you want and need, it receives a `form` in the context and it is a WTForms form. The submit should be done to `request.path` which is the same as the login view.

You can also use `{% if is_logged_in() %}` in your template if needed.

4.3.2 Customizing or translating message alerts

The default message alerts are:

In the `auth_error` message, the `{0}` in the authentication error is a required placeholder that is replaced by the validator error message.

```
from flask_simplelogin import Message,
# ...

app = Flask(__name__)

messages = {
    'login_success': Message('Você está dentro!'),  # the default CSS class is␣
↪`primary`
    'login_failure': 'ungültige Anmeldeinformationen',  # this also uses the default␣
↪CSS class
    'is_logged_in': Message('Iam initium', 'success'), # this uses `success` as the␣
↪CSS class
    'logout': None, # this disables the message for logout
    'login_required': 'Devi prima accedere',
    'access_denied': 'Acceso denegado',
    'auth_error': ' {0}'
}
SimpleLogin(app, messages=messages)
```

4.3.3 Custom validators

When you pass the `must` argument to `login_required` decorator, it can be a function or a list of functions. If function returns `None`, it means **no** error and validation passed. If function returns an error message (stringt), it means validation failed.

```
def be_admin(username):
    """Validator to check if user has admin role"""
    user_data = my_users.get(username)
    if not user_data or 'admin' not in user_data.get('roles', []):
        return "User does not have admin role"


def have_approval(username):
    """Validator: all users approved so return None"""
    return


@app.route('/protected')
@login_required(must=[be_admin, have_approval])
def protected():
    return render_template('secret.html')
```

Take a look at the example app.

4.4 Extras

4.4.1 Do you need Access Control?

You can easily mix Flask Simple Login withFlask-Allows:

```
$ pip install flask_allows
```

And then:

```
from flask import Flask, g
from flask_simplelogin import SimpleLogin
from flask_allows import Allows

app = Flask(__name__)
app.config['SECRET_KEY'] = 'something-secret'


def is_staff(ident, request):
    return ident.permlevel == 'staff'

def only_chuck_norris_can_login(user):
    if user.get('username') == 'chuck' and user.get('password') == 'norris':
        # Bind the logged in user data to the `g` global object
        g.user.username = user['username']
        g.user.permlevel = 'staff'  # set user permission level
        return True  # Allowed
    return False  # Denied

# init allows
allows = Allows(identity_loader=lambda: g.user)

# init SimpleLogin
SimpleLogin(app, login_checker=only_chuck_norris_can_login)

# a view which requires a logged in user to be member of the staff group
@app.route('/staff_only')
@allows.requires(is_staff)
@login_required
def a_view():
    return "staff only can see this"
```

4.4.2 Need JSON Web Token (JWT) support?

Take a look at Flask-JWT-Simple and of course you can mix it with Flask Simple Login.

Alternatives:

- Flask-Login
- Flask-User
- Flask-Security
- Flask-Principal

Those extensions are really complete and production ready!

CHAPTER

FIVE

REFERENCES:

- genindex
- modindex
- search