GeekBrains

Flask

WTForms. Регистрация и авторизация пользователя

На этом уроке

1. Познакомимся с WTForms.
2. Познакомимся с библиотекой Flask-Bcrypt.
3. Научимся создавать и обрабатывать формы.
4. Узнаем, как сохранять данные формы в БД.
5. Поговорим про хэширование.
6. Создадим макрос для переиспользования в шаблонах.

Оглавление

Теория

WTForms — программная библиотека на языке Python для работы с HTML-формами, чтобы было проще.

CSRF — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Если жертва заходит на сайт, созданный злоумышленником, от её лица тайно отправляется запрос на другой сервер, осуществляющий некую вредоносную операцию.

Flask-Bcrypt — библиотека, позволяющая упростить безопасное хранение паролей. Основана на библиотеке bcrypt (занимается хэшированием паролей).

MD5 — 128-битный алгоритм хэширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института в 1991 году. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности. Сейчас считается небезопасным, так как с развитием видеокарт и майнинга большое количество хэшей были скомпрометированы, а остальные легко взламываются.

Secure Hash Algorithm 1 — алгоритм криптографического хэширования. Описан в RFC 3174. Для входного сообщения произвольной длины алгоритм генерирует 160-битное хэш-значение, называемое также дайджестом сообщения, которое обычно отображается как шестнадцатиричное число длиной в 40 цифр.

Практическое задание

Установка и настройка Flask-Bcrypt

Устанавливаем Flask-Bcrypt. Обратите внимание, что название проекта в PyPI отличается:

```
pip install Bcrypt-Flask
```

Создаём `blog/security.py`:

```
from flask_bcrypt import Bcrypt

flask_bcrypt = Bcrypt()

__all__ = [
    "flask_bcrypt",
]
```

Инициализируем в app.py:

```
from blog.security import flask_bcrypt

flask_bcrypt.init_app(app)
```

Добавляем пароль пользователю

Редактируем blog/models/user.py. Объявляем колонку, геттер и сеттер. При установке пароля (user.password = ‘secure pass’) будет происходить хэширование пароля и установка хэша в колонку, связанную с БД.

```
from sqlalchemy import Column, Integer, String, Boolean, LargeBinary
from blog.security import flask_bcrypt

class User(db.Model, UserMixin):
    ...

    _password = Column(LargeBinary, nullable=True)

    @property
    def password(self):
        return self._password

    @password.setter
    def password(self, value):
        self._password = flask_bcrypt.generate_password_hash(value)

    def validate_password(self, password) -> bool:
        return flask_bcrypt.check_password_hash(self._password, password)
```

Создаём и выполняем миграцию.

Редактируем blog/app.py:

- сносим команду `init_db` — теперь созданием таблиц занимается Flask-Migrate;
- редактируем команду `create_users` — теперь это `create_admin`. Проставляем пароль:

```
@app.cli.command("create-admin")
def create_admin():
    """
    Run in your terminal:
    ➜ flask create-admin
    > created admin: <User #1 'admin'>
    """
    from blog.models import User

    admin = User(username="admin", is_staff=True)
    admin.password = os.environ.get("ADMIN_PASSWORD") or "adminpass"

    db.session.add(admin)
    db.session.commit()

    print("created admin:", admin)
```

Добавляем имя и фамилию пользователю

В очередной раз редактируем blog/models/user.py:

```
class User(db.Model, UserMixin):
    ...

    first_name = Column(String(120), unique=False, nullable=False, default="", server_default="")
    last_name = Column(String(120), unique=False, nullable=False, default="", server_default="")
```

Создаём и выполняем миграции:

- flask db migrate -m “update user model”;
- flask db upgrade.

Устанавливаем WTForms и валидатор email

```
pip install WTForms email-validator
```

Устанавливаем уникальный email у пользователя

blog/models/user.py:

```
class User(db.Model, UserMixin):
    ...

   email = Column(String(255), unique=True, nullable=False, default="", server_default="")
```

И снова создаём и выполняем миграцию.

Создаём помощника для формы — макрос

Добавляем макрос blog/templates/macro/formhelpers.html — это переиспользуемый кусок шаблона, который мы будем применять для отрисовки полей формы.

```
{% macro render_field(field) %}
  <div class="row my-3">
    {{ field.label(class="col-sm-4 col-md-3 col-lg-2 col-form-label") }}
    <div class="col-sm-8 col-md-6">
      {{ field(class="form-control" + (" is-invalid" if field.errors else ""))|safe }}
      {% if field.errors %}
        {% for error in field.errors %}
          <div class="invalid-feedback">
            {{ error }}
          </div>
        {% endfor %}
      {% endif %}
    </div>
  </div>
{% endmacro %}
```

Включаем защиту CSRF

Добавляем в BaseConfig в blog/configs.py:

```
class BaseConfig(object):
    ...

    WTF_CSRF_ENABLED = True
```

Создаём форму для регистрации

Создаём модуль forms (папку с файлом blog/forms/__init__.py).

В модуле blog/forms/user.py создаём форму регистрации. Добавляем валидаторы полей.

```
from flask_wtf import FlaskForm
from wtforms import StringField, validators, PasswordField, SubmitField


class UserBaseForm(FlaskForm):
    first_name = StringField("First Name")
    last_name = StringField("Last Name")
    username = StringField(
        "username",
        [validators.DataRequired()],
    )
    email = StringField(
        "Email Address",
        [
            validators.DataRequired(),
            validators.Email(),
            validators.Length(min=6, max=200),
        ],
        filters=[lambda data: data and data.lower()],
    )


class RegistrationForm(UserBaseForm):
    password = PasswordField(
        "New Password",
        [
            validators.DataRequired(),
            validators.EqualTo("confirm", message="Passwords must match"),
        ],
    )
    confirm = PasswordField("Repeat Password")
    submit = SubmitField("Register")
```

Создаём шаблон регистрации blog/templates/auth/register.html. В нём отрисовываем csrf_token и при помощи render_field добавляем поля формы. Добавляем алерт для возможности вывода сообщения на страницу.

```
{% extends 'base.html' %}
{% from "macro/formhelpers.html" import render_field %}

{% block title %}
  Register
{% endblock %}

{% block body %}
  <h1>Register</h1>
  {% if error %}
    <div class="alert alert-danger">
      {{ error }}
    </div>
  {% endif %}
  <form method="POST">
    {# csrf protection #}
    {{ form.csrf_token() }}

    {# show inputs #}
    {% for field in ['first_name', 'last_name', 'username', 'email', 'password', 'confirm'] %}
      {{ render_field(form[field]) }}
    {% endfor %}

    {# submit #}
    <div>
      {{ form.submit(class="btn btn-primary btn-lg") }}
    </div>

  </form>
{% endblock %}
```

Редактируем blog/views/auth.py, используя новые модули. Добавляем регистрацию. Используем форму для проверки введенных данных. Валидируем почту и юзернейм.

```
from flask import Blueprint, render_template, request, redirect, url_for, current_app
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from sqlalchemy.exc import IntegrityError

from blog.models.database import db
from blog.models import User
from blog.forms.user import RegistrationForm


@auth_app.route("/register/", methods=["GET", "POST"], endpoint="register")
def register():
    if current_user.is_authenticated:
        return redirect("index")

    error = None
    form = RegistrationForm(request.form)
    if request.method == "POST" and form.validate_on_submit():
        if User.query.filter_by(username=form.username.data).count():
            form.username.errors.append("username already exists!")
            return render_template("auth/register.html", form=form)

        if User.query.filter_by(email=form.email.data).count():
            form.email.errors.append("email already exists!")
            return render_template("auth/register.html", form=form)

        user = User(
            first_name=form.first_name.data,
            last_name=form.last_name.data,
            username=form.username.data,
            email=form.email.data,
            is_staff=False,
        )
        user.password = form.password.data
        db.session.add(user)
        try:
            db.session.commit()
        except IntegrityError:
            current_app.logger.exception("Could not create user!")
            error = "Could not create user!"
        else:
            current_app.logger.info("Created user %s", user)
            login_user(user)
            return redirect(url_for("index"))
    return render_template("auth/register.html", form=form, error=error)
```

Переносим логин

Продолжаем работать с blog/views/auth.py. Авторизацию по юзернейму оставляем админу в виде login-as. Делаем доступ только для администратора (добавляем проверку `current_user.is_authenticated and current_user.is_staff`). Дальше добавим авторизацию по юзернейму и паролю для всех пользователей.

```
from werkzeug.exceptions import NotFound

def login():
    return "WIP"
    # if request.method == "GET":
    #     return render_template("auth/login.html")
    #
    #
    # login_user(user)
    # return redirect(url_for("index"))


@auth_app.route("/login-as/", methods=["GET", "POST"], endpoint="login-as")
def login_as():
    if not (current_user.is_authenticated and current_user.is_staff):
        # non-admin users should not know about this feature
        raise NotFound
    ...
```

Создаём логин для пользователей

Редактируем файл с формами пользователя blog/forms/user.py. Добавляем `LoginForm`:

```
class LoginForm(FlaskForm):
    username = StringField(
        "username",
        [validators.DataRequired()],
    )
    password = PasswordField(
        "Password",
        [validators.DataRequired()],
    )
    submit = SubmitField("Login")
```

Создаём шаблон для входа пользователя blog/templates/auth/login.html. Используем форму. Не забываем проставить CSRF защиту.

```
{% extends 'base.html' %}
{% from "macro/formhelpers.html" import render_field %}

{% block title %}
  Register
{% endblock %}

{% block body %}
  <h1>Register</h1>
  {% if error %}
    <div class="alert alert-danger">
      {{ error }}
    </div>
  {% endif %}
  <form method="POST">
    {# csrf protection #}
    {{ form.csrf_token() }}

    {# show inputs #}
    {% for field in ['username', 'password'] %}
      {{ render_field(form[field]) }}
    {% endfor %}

    {# submit #}
    <div>
      {{ form.submit(class="btn btn-primary btn-lg") }}
    </div>

  </form>
{% endblock %}
```

И теперь добавляем view для регистрации в blog/views/auth.py:

```
from blog.forms.user import RegistrationForm, LoginForm

def login():
    if current_user.is_authenticated:
        return redirect("index")

    form = LoginForm(request.form)

    if request.method == "POST" and form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).one_or_none()
        if user is None:
            return render_template("auth/login.html", form=form, error="username doesn't exist")
        if not user.validate_password(form.password.data):
            return render_template("auth/login.html", form=form, error="invalid username or password")

        login_user(user)
        return redirect(url_for("index"))

    return render_template("auth/login.html", form=form)
```

Теперь для удобства пользователя редактируем базовый темплейт blog/templates/base.html: добавляем ссылки в навигационную панель.

```
{% for (endpoint, label) in [('auth_app.login', 'Login'), ('auth_app.register',
'Register')] %}
   <a href="{{ url_for(endpoint) }}"
        class="nav-link {% if request.endpoint == endpoint -%}active{%- endif %}">
     {{ label }}
   </a>
  {% endfor %}
```

Теперь пользователь может выполнять вход по юзернейму и паролю, а затем разлогиниваться. Админ всё ещё может выполнять авторизацию от имени любого пользователя.

Итоги

Мы рассмотрели библиотеку WTForms и применим ее для создания и обработки анкет пользователей. Узнали, что такое CSRF и как его применять. Добавили view для регистрации и входа пользователей.

Практическое задание

1. Создать форму регистрации с использованием WTForms.
2. Добавить view для регистрации пользователя.
3. Создать форму входа с использованием WTForms.
4. Добавить view для авторизации пользователя.

Дополнительные материалы

1. Flask-Migrate.
2. WTForms FAQ.
3. https://flask-bcrypt.readthedocs.io/en/latest/.
4. https://owasp.org/www-community/attacks/csrf.