GeekBrains

Flask

Flask-Admin

© geekbrains.ru

На этом уроке

1. Установим Flask-Admin.
2. Добавим модели в админку.
3. Научимся работать в админке.
4. Узнаем про ограничения ModelView.

Оглавление

Теория

Flask-Admin

В двух словах, это расширение для фреймворка Flask, которое позволяет быстро создавать административный интерфейс, напоминающий Django.

Что умеет Flask-Admin из коробки:

1. Генерация меню (до двух уровней) из подключенных «кирпичиков» с учётом правил доступа.

2. Возможность управления доступом без каких-либо предположений об используемой системе авторизации.

3. Набор базовых классов для создания своих «кирпичиков».

3. CRUD для моделей SQLAlchemy, включая пейджинг, сортировку, фильтры, поиск и тому подобное.

4. Файловый менеджер.

5. Локализация.

Практическое задание

Установка и настройка Flask-Admin

```
pip install Flask-Admin
```

И создаём файл blog/admin.py:

```
from flask_admin import Admin
from flask_admin.contrib.sqla import ModelView

from blog import models
from blog.models.database import db


# Customized admin interface
class CustomView(ModelView):
    pass


# Create admin with custom base template
admin = Admin(name="Blog Admin", template_mode="bootstrap4")

# Add views
admin.add_view(CustomView(models.Tag, db.session, category="Models"))
```

В конфиге blog/configs.py прописываем желаемую тему для админки (bootswatch — бесплатные темы для Bootstrap. Список можно посмотреть тут: https://bootswatch.com/).

```
FLASK_ADMIN_SWATCH = 'cosmo'
```

В модуле блога в уже знакомом файле app.py добавляем код для работы с админкой:

```
from blog.admin import admin

admin.init_app(app)
```

Заменяем посадочную страницу админки своей. В файле blog/templates/admin/index.html необходимо расширить master шаблон и добавить туда желаемый текст. Обратите внимание, что мы добавляем шаблон в папку templates/admin/ — таким образом он заменит стандартный шаблон Flask-Admin.

```
{% extends 'admin/master.html' %}
 {% block body %}
   <h1>Hello Blog Admin</h1>
   <h5>Use navbar to select model</h5>
 {% endblock %}
```

Добавляем модели статьи и тега в админку

blog/admin.py:

```
# Add views
admin.add_view(CustomView(models.Tag, db.session, category="Models"))
admin.add_view(CustomView(models.Article, db.session, category="Models"))
```

Создаём строковые данные для некоторых моделей

Это необходимо, чтобы в админке модели отображались в понятном для нас виде.

В статьях blog/models/article.py

```
class Article(db.Model):
    ...

    def __str__(self):
        return self.title
```

В авторах blog/models/author.py

```
class Author(db.Model):
    ...

    def __str__(self):
         return self.user.username
```

В тегах blog/models/tag.py

```
class Tag(db.Model):
    ...

    def __str__(self):
         return self.name
```

Определяем кастомные свойства для админки тегов

Это просто. В модуле blog/admin.py создаём новый view (наследуемся от `CustomView`) и определяем желаемые свойства. Например, параметры для фильтрации, поиска, экспорта и т.д.

Регистрируем новый класс view для нужной модели. Мы можем переиспользовать этот класс и для других моделей, но создавая его для нужной модели, мы можем строго связать правила администрирования с выбранной моделью.

```
class TagAdminView(CustomView):
    column_searchable_list = ("name",)
    column_filters = ("name",)
    can_export = True
    export_types = ["csv", "xlsx"]
    create_modal = True
    edit_modal = True

admin.add_view(TagAdminView(models.Tag, db.session, category="Models"))
```

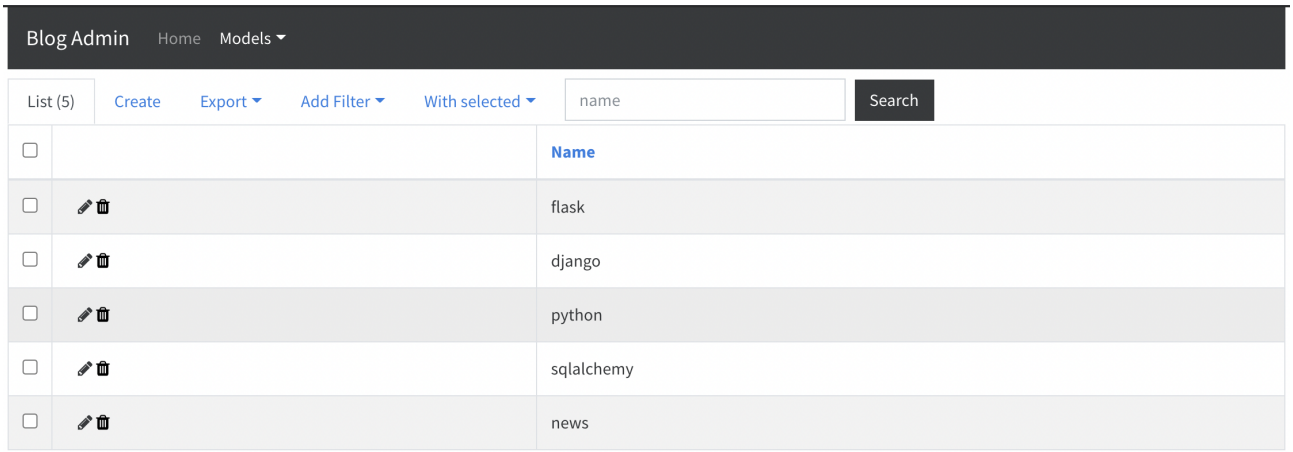
Создаём view в админке для пользователей

Все так же редактируем blog/admin.py:

```
class UserAdminView(CustomView):
    column_exclude_list = ("_password",)
    column_searchable_list = ("first_name", "last_name", "username", "is_staff", "email")
    column_filters = ("first_name", "last_name", "username", "is_staff", "email")
    column_editable_list = ("first_name", "last_name", "is_staff")
    can_create = True
    can_edit = True
    can_delete = False

admin.add_view(UserAdminView(models.User, db.session, category="Models"))
```

Теперь при посещении страницы /admin мы можем просматривать доступные модели, редактировать их и создавать новые.


Blog Admin
Home
Models
List (5)
Create
Export
Add Filter
With selected
name
Search
Name
flask
django
python
sqlalchemy
news

Ограничиваем доступ в админку

Конечно же, не всем пользователям дозволено заглядывать в админку. Поэтому необходимо ограничить доступ только администраторам. Для этого расширяем `CustomView`, который мы заблаговременно создали и от которого наследовали и создавали все утилиты управления моделями.

blog/admin.py — добавляем методы `is_accessible` и `inaccessible_callback`, в которых описываем проверки авторизации и редирект в случае отсутствия доступов у пользователя.

```
from flask import redirect, url_for
from flask_login import current_user

class CustomView(ModelView):

    def is_accessible(self):
        return current_user.is_authenticated and current_user.is_staff

    def inaccessible_callback(self, name, **kwargs):
        # redirect to login page if user doesn't have access
        return redirect(url_for("auth_app.login"))
```

При попытке зайти на редактирование любой из моделей от имени пользователя, у которого нет административных прав, мы получим редирект. Однако главная страница админки с приветствием всё ещё доступна, так как она создана через отдельный view. Чтобы переопределить доступ к нему, необходимо отредактировать стандартный view админки (blog/admin.py):

```
from flask_admin import Admin, AdminIndexView, expose

class MyAdminIndexView(AdminIndexView):

    @expose("/")
    def index(self):
        if not (current_user.is_authenticated and current_user.is_staff):
            return redirect(url_for("auth_app.login"))
        return super(MyAdminIndexView, self).index()

# Create admin with custom props
admin = Admin(
    name="Blog Admin",
    index_view=MyAdminIndexView(),
    template_mode="bootstrap4",
)
```

Теперь при попытке посетить /admin без прав администратора будет выполнен редирект на страницу авторизации.

Итоги

На занятии мы добавили Flask-Admin в свой проект, подключили модели к админке и присвоили теги, а также ограничили доступ к админке.

Практическое задание

1. Добавить Flask-Admin в свой проект на Flask.
2. Подключить модели к админке, присвоить теги.
3. Ограничить доступ в админку только администраторам.

Дополнительные материалы

1. https://flask-admin.readthedocs.io/en/latest/introduction/.