GeekBrains

Flask

Продолжаем работу с Flask-COMBO-JSONAPI, знакомство с ComboJSONAPI, системой плагинов

На этом уроке

1. Создадим RPC views.
2. Ограничим доступы.

Оглавление

Теория

Практическое задание

- Подключение EventPlugin
- Создание ArticleListEvents с подсчётом статей
- Создаём AuthorDetailEvents: event_get_articles_count
- Настраиваем PermissionPlugin и создаем UserPermission
- Создаём модуль blog/permissions
- Добавляем UserPermission для User
- Ограничиваем редактирование пользователя

Итоги

Практическое задание

Дополнительные материалы

Теория

Удалённый вызов процедур, реже Вызов удалённых процедур (от англ. Remote Procedure Call, RPC) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (на удалённых компьютерах либо в независимой сторонней системе на том же устройстве). В текущем контексте это любой кастомный view, отличающийся от CRUD.

Практическое задание

Подключение EventPlugin

В blog/api/__init__.py подключаем EventPlugin:

```
from combojsonapi.event import EventPlugin

def init_api(app):
    event_plugin = EventPlugin()
    api_spec_plugin = create_api_spec_plugin(app)
    api = Api(
        app,
        plugins=[
            event_plugin,
            api_spec_plugin,
        ],
    )
```

Создание ArticleListEvents с подсчётом статей

Например, мы хотим создать RPC (event), связанный не с конкретной моделью (статьи, пользователя, тега), а с самим ресурсом. Это может быть запрос количества элементов (мы можем выяснить количество и через стандартный CRUD, но в данном случае это максимально простой пример. Мы можем в кастомных view делать любые вызовы и действия).

blog/api/article.py:

```
from combojsonapi.event.resource import EventsResource

class ArticleListEvents(EventsResource):
    def event_get_count(self):
        return {"count": Article.query.count()}


class ArticleList(ResourceList):
    events = ArticleListEvents
    ...
```

Теперь нам доступно api /api/articles/event_get_count/. Мы можем найти его и в Swagger.

Стоит обратить внимание, что так как метод называется event_get_..., то метод, используемый для этого ресурса, тоже GET. В ином случае будет использован метод POST.

Создаём AuthorDetailEvents: event_get_articles_count

Например, мы хотим создать RPC (event), связанный с конкретной моделью — в данном случае мы хотим узнать количество статей, связанных с выбранным автором. В таком случае мы добавляем новый event, связанный с details resource (с классом AuthorDetail через атрибут events).

Редактируем blog/api/author.py:

```
from combojsonapi.event.resource import EventsResource
from blog.models import Author, Article

class AuthorDetailEvents(EventsResource):
    def event_get_articles_count(self, **kwargs):
        return {"count": Article.query.filter(Article.author_id == kwargs["id"]).count()}

class AuthorDetail(ResourceDetail):
    events = AuthorDetailEvents
```

Новый ресурс доступен по пути /api/authors/<int:id>/event_get_articles_count/.

Настраиваем PermissionPlugin и создаем UserPermission

Импортируем и инициализируем PermissionPlugin. Если передать strict=True, то объект Permission будет требоваться для всех методов CRUD, иначе будут применены только доступные ограничения.

blog/api/__init__.py

```
from combojsonapi.permission import PermissionPlugin

def init_api(app):
    event_plugin = EventPlugin()
    api_spec_plugin = create_api_spec_plugin(app)
    permission_plugin = PermissionPlugin(strict=False)
    api = Api(
        app,
        plugins=[
            event_plugin,
            api_spec_plugin,
            permission_plugin,
        ],
    )
```

Создаём модуль blog/permissions

Добавляем Permission для пользователя. Нам необходимо указать доступные поля. Добавляем все, кроме пароля. А также ограничиваем доступ только для админов.

blog/permissions/user.py

```
from combojsonapi.permission.permission_system import (
    PermissionMixin,
    PermissionUser,
    PermissionForGet,
)
from flask_combo_jsonapi.exceptions import AccessDenied
from flask_login import current_user

from blog.models.user import User


class UserPermission(PermissionMixin):
    ALL_AVAILABLE_FIELDS = [
        "id",
        "first_name",
        "last_name",
        "username",
        "is_staff",
        "email",
    ]

    def get(self, *args, many=True, user_permission: PermissionUser = None, **kwargs)
-> PermissionForGet:
        """
        Set available columns

        :param args:
        :param many:
        :param user_permission:
        :param kwargs:
        :return:
        """
        if not current_user.is_authenticated:
            raise AccessDenied("no access")

        self.permission_for_get.allow_columns = (self.ALL_AVAILABLE_FIELDS, 10)
        return self.permission_for_get
```

Добавляем UserPermission для User

Редактируем blog/api/user.py. Необходимо в data_layer передать список доступов для выбранного метода.

```
from blog.permissions.user import UserPermission

class UserDetail(ResourceDetail):
    ...

    data_layer = {
        ...
        "permission_get": [UserPermission],
     }
```

Теперь можно попробовать обратиться к API пользователя от имени анонимного пользователя, обычного пользователя и администратора. Доступ будет только у последнего. Также по API нам недоступен пароль пользователя (несмотря на то что он хеширован, отдавать его на фронтенд небезопасно).

Ограничиваем редактирование пользователя

Однако отредактировать пользователя всё ещё могут любые пользователи. И они могут менять любые поля. Поэтому добавим ограничения и для такого случая.

Так как редактирование производится методом PATCH, нужно создать для него ограничения. Добавляем доступные для редактирования поля, указываем их в методе patch_permission и применяем в методе patch_data — оставляем в данных для редактирования только доступные для редактирования поля.

blog/permissions/user.py

```
from combojsonapi.permission.permission_system import (
    ...,
    PermissionForPatch,
)


class UserPermission(PermissionMixin):
    ...
    PATCH_AVAILABLE_FIELDS = [
        "first_name",
        "last_name",
    ]

    ...

    def patch_permission(self, *args, user_permission: PermissionUser = None, **kwargs) ->
PermissionForPatch:
        self.permission_for_patch.allow_columns = (self.PATCH_AVAILABLE_FIELDS, 10)
        return self.permission_for_patch

    def patch_data(self, *args, data: dict = None, obj: User = None, user_permission:
PermissionUser = None, **kwargs) -> dict:
        permission_for_patch = user_permission.permission_for_patch_permission(model=User)
        return {
            i_key: i_val
            for i_key, i_val in data.items()
            if i_key in permission_for_patch.columns
        }
```

Итоги

На занятии мы научились создавать RPC (event) views и ограничивать доступ к различным views и отдельным полям. Мы настроили систему доступов таким образом, чтобы можно было ограничить права пользователей: пользователя может редактировать только админ.

Практическое задание

1. Добавить в свой проект на Flask страницу со списком публикаций, которая будет подгружать данные через новый JSON API ресурс.
2. Настроить систему доступов так, чтобы ограничить права пользователей:
    - ограничить чтение / редактирование пользователя;
    - статью может редактировать админ или автор статьи.

Дополнительные материалы

1. https://github.com/AdCombo/combojsonapi/blob/master/docs/en/event_plugin.rst.
2. https://github.com/AdCombo/combojsonapi/blob/master/docs/en/permission_plugin.rst.
3. https://flask-combo-jsonapi.readthedocs.io/en/latest/permission.html.