Permission 시스템을 통해 사용자 접근 제한하기

(중급편) 장고 기본 인증 뽀개기

Ask Company

Permissions [1]

Binary (yes/no) flags designating whether a user may perform a certain task.

---

[1] 장고 2.0 관련문서 - https://docs.djangoproject.com/en/2.0/topics/auth/default/#topic-authorization

허가 시스템 (Permissions)

- 특정 User/Group에 Permission 할당
- User/Group admin 페이지에서 각 User/Group 별로 Permission 할당 UI 제공
- **모델 별**로 Permissions 지정
  - 디폴트 Permissions 제공 : 추가(add), 수정(change), 삭제(delete)
  - 각 모델이 **migrate**될 때, 지정된 Permissions이 등록됩니다. (Permission 테이블에 대한 새로운 Record)
  - admin에서는 change권한이 있어야 목록조회가 가능합니다.
  - view 권한은 정의되어있지 않습니다. 필요하다면 커스텀 지정 가능.

새로운 Permission 추가하기

```
class 모델클래스(models.Model):
    필드정의

    class Meta:
        # 기존 ['add', 'change', 'delete'] 덮어쓰기
        default_permissions = ['vote']

        # 추가 퍼미션 지정
        permissions = [
            ['can_view_post', 'Can view post'],
        ]
```

이렇게 정의된 퍼미션은 **migration**시에 해당 Permission Record가 없을 경우, 새로이 생성됩니다.
이미 생성된 Permission Record는 관련 Permission이 제거되더라도, **migration** 시에 같이 제거되지 않습니다.

상황: post를 view할 권한이 있는 유저만 뷰를 호출토록 제한할려면 !

```
@login_required
def post_detail(request, pk):
    # TODO: request.user가 권한이 있는 지를 매번 체크
    #  => 권한설정은 어디에 저장을 할 것이며, 어떤 로직을 거쳐서 권한을 확인할 것인지 직접 구현
```

이를 Permission 시스템을 활용하면 다음 코드가 가능

1) 원하는 User에게 **"blog.can_view_post"** 권한을 부여. 이는 *"AppLabel.codename"* 형식의 단순한 문자열 일 뿐입니다.

2) 지정 Permission이 없을 경우, 원하는 페이지(ex: login_url)로 redirect시키거나 403 Forbidden 페이지를 보여줄 수 있습니다.

```
@permission_required('blog.can_view_post')
def post_detail(request, pk):
    pass
```

주요 모델 코드 [2]

2.0.7/django/contrib/auth/models.py

```
class Permission(models.Model):
    name = models.CharField(max_length=255)                         # Label (ex: "Can change profile")
    content_type = models.ForeignKey(ContentType, models.CASCADE)   # 대상 모델 지정
    codename = models.CharField(max_length=100)                     # Permission 종류 지정

    class Meta:
        unique_together = [
            ('content_type', 'codename'),
        ]

class Group(models.Model):
    name = models.CharField(max_length=80, unique=True)
    permissions = models.ManyToManyField(Permission, blank=True)
```

[2] contenttypes Framework: https://docs.djangoproject.com/en/2.0/ref/contrib/contenttypes/

```python
class PermissionsMixin(models.Model):
    groups = models.ManyToManyField(Group, blank=True)
    user_permissions = models.ManyToManyField(Permission, blank=True)

    def has_perm(self, perm, obj=None):            # 1개 퍼미션에 대한 권한 체크
        if self.is_active and self.is_superuser:   # superuser는 모든 퍼미션 허용
            return True
        return _user_has_perm(self, perm, obj)

    def has_perms(self, perm_list, obj=None):      # 다수 퍼미션에 대해 모두 권한이 있는 지 체크
        return all(self.has_perm(perm, obj) for perm in perm_list)

class AbstractUser(AbstractBaseUser, PermissionsMixin):
    # 생략
```

실제 데이터베이스 확인

```
>>> from django.contrib.auth.models import Permission

>>> Permission.objects.all().count()
27

>>> for perm in Permission.objects.all():
...     print(perm)

accounts | profile | Can add profile
accounts | profile | Can change profile
accounts | profile | Can delete profile
accounts | user | Can add user
accounts | user | Can change user
accounts | user | Can delete user
... 중략 ...
contenttypes | content type | Can add content type
contenttypes | content type | Can change content type
contenttypes | content type | Can delete content type
sessions | session | Can add session
sessions | session | Can change session
sessions | session | Can delete session
```

```
>>> for perm in Permission.objects.all():
...     print('{}.{}'.format(perm.content_type.app_label, perm.codename))

accounts.add_profile
accounts.change_profile
accounts.delete_profile
accounts.add_user
accounts.change_user
accounts.delete_user
... 중략 ...
contenttypes.add_contenttype
contenttypes.change_contenttype
contenttypes.delete_contenttype
session.add_session
session.change_session
session.delete_session
```

특정 User에 특정 Permission을 가지고 있는 지 체크하고자 할 때

myapp(app label)에 있는 MyModel에 대한 체크

```
user.has_perm('myapp.add_mymodel')      # True or False
user.has_perm('myapp.change_mymodel')
user.has_perm('myapp.delete_mymodel')
```

내부 동작

1. 지정 user 객체에 대한 모든 Permission Set을 만들어서 메모리에 캐싱합니다.
2. 캐싱된 Permission Set에 대해서 membership check를 수행합니다.
3. 문자열로 지정된 퍼미션에 대한 Permission이 없더라도 **Permission.DoesNotExist 예외**가 발생하지 않습니다. 단순히 **가진 Permission Set에 대한 membership check** 방식이기 때문입니다.

특정 User에 코드로 Permission 적용하기

```
# User에게 직접 Permissions 적용
myuser.user_permissions.set([permission_list])                # 지정 Permission 으로만 설정
myuser.user_permissions.add(permission, permission, ...)      # 지정 Permission 추가
myuser.user_permissions.remove(permission, permission, ...)   # 지정 Permission 제거
myuser.user_permissions.clear()                               # 모든 Permission 제거

# Group에 직접 Permission 적용
mygroup.permissions.set([permission_list])
mygroup.permissions.add(permission, permission, ...)
mygroup.permissions.remove(permission, permission, ...)
mygroup.permissions.clear()

# User에게 다수 Group 지정/제거함으로서, 해당 Group에 지정된 Permissions 적용
myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
```

User에게 필요한 권한지정을 위해 Permission 객체 획득하기

```
class Permission(models.Model):
    name = models.CharField(max_length=255)
    content_type = models.ForeignKey(ContentType, models.CASCADE)  # 모델 지정
    codename = models.CharField(max_length=100)

    class Meta:
        unique_together = [
            ('content_type', 'codename'),
        ]
```

Permission 체크 시에 실제로 참조하는 값인, **content_type**과 **codename**을 통해서 획득하세요.

```
from django.contrib.auth.models import Permission

# 다음 코드에서는 DoesNotExist 예외와 더불어 MultipleObjectsReturned 예외가 발생할 수 있습니다.
# 모델명은 다른 앱과의 관계에서 중복될 수 있으며, name은 label로서 key로서의 의미가 부족합니다.
perm = Permission.objects.get(name='Can change post')
```

```
from django.contrib.auth.models import Permission
from django.contrib.contenttypes.models import ContentType
from blog.models import Post

# 다음과 같이 content_type과 codename을 같이 지정해주세요.
content_type = ContentType.objects.get_for_model(Post)
perm = Permission.objects.get(
    content_type=content_type,
    codename='change_post')

user.user_permissions.add(perm)
```

```python
from django.contrib.contenttypes.models import ContentType

# Permission 객체 획득
content_type = ContentType.objects.get_for_model(BlogPost)
perm = Permission.objects.get(
    codename='change_blogpost',
    content_type=content_type,
)

user.user_permissions.add(perm)
```

[ModelBackend] 한 User에 대한 모든 Permission Set을 캐싱

request/response Cycle에서 유용한 전략

2.0.7/django/contrib/auth/backends.py#L77

```
def myview(request):
    # 내부적으로 현재 User에 대한 모든 Permission Set을 메모리에 캐싱합니다.
    request.user.has_perm('blog.change_post')

    # 현재 user에 대해 Permission이 변경되더라도 ...

    # 기존에 캐싱된 Permission Set을 통해 루틴이 수행됩니다.
    request.user.has_perm('blog.change_post')
```

*Permission*을 다시 체크하고자한다면, **user** 객체를 *DB*에서 다시 읽어와야합니다.

장식자 (1) [3]

user.has_perms(퍼미션) => False 시에 login_url로 이동 (디폴트: settings.LOGIN_URL)

```
from django.contrib.auth.decorators import permission_required

@permission_required('blog.can_view_post')
def post_detail(request, pk):
    pass

@permission_required('blog.can_view_post', login_url='/loginpage/')
def post_detail(request, pk):
    pass
```

[3] Code - https://docs.djangoproject.com/en/2.0/_modules/django/contrib/auth/decorators/#permission_required

장식자 (2)

user.has_perms(퍼미션) => False 시에 403 (HTTP Forbidden) 보여주기

로그인 여부를 먼저 검증하고, 로그인 유저가 특정 퍼미션이 없을 경우
HTTP Forbidden 페이지 보여주기

```
from django.contrib.auth.decorators import login_required, permission_required

@login_required
@permission_required('blog.can_view_post', raise_exception=True)
def post_detail(request, pk):
    pass
```

미션

로그인 여부를 먼저 검증하고, 로그인 유저가 GoldUser가 아닐 경우 등급 업그레이드 안내 페이지 보여주기

1. goldmembership_guide 페이지 만들기
2. blog.can_view_goldpage 권한 추가 => migrate
3. 뷰에 blog.can_view_goldpage 퍼미션 제한 => 접근 확인
4. 유저 혹은 그룹에 blog.can_view_goldpage 권한 부여 => 접근 확인

힌트

```
@login_required
@permission_required('blog.can_view_goldpage', login_url=reverse_lazy('goldmembership_guide'))
def post_detail(request, pk):
    pass
```

추천 써드파티: django-rules

https://github.com/dfunckt/django-rules

A tiny but powerful app providing object-level permissions to Django, without requiring a database. At its core, it is a generic framework for building rule-based systems, similar to decision trees. It can also be used as a standalone library in other contexts and frameworks.

```
@rules.predicate
def is_book_author(user, book):
    return book.author == user

rules.add_rule('can_edit_book', is_book_author)

guidetodjango = Book.objects.get(isbn='978-1-4302-1936-1')
adrian = User.objects.get(username='adrian')
rules.test_rule('can_edit_book', adrian, guidetodjango)  # True or False
```

Life is short,

Use Python3/Django.

여러분의
파이썬/장고 페이스메이커가
되겠습니다. ;)

ASK
COMPANY