User Roles

Brian Dickinson

University of Rochester

Fall 2019
User Roles

Thus far all users have been treated equally. Only id differentiates logged in users. There are however a variety of different user classes in most online systems. e.g.

- Administrator
- Moderator
- User
- Anonymous
User Permissions

Using roles it is easy to assign different permissions to different groups of users. On a social networking website such permissions might include the following.

- Follow
- Comment
- Write
- Moderate
- Admin
class Role(db.Model):
    __tablename__ = "Roles"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)
    default = db.Column(db.Boolean, default=False, index=True)
    permissions = db.Column(db.Integer)
    users = db.relationship("User", backref="role", lazy="dynamic")

def __init__(self, **kwargs):
    super(Role, self).__init__(**kwargs)
    if self.permissions is None:
        self.permissions = 0
class Permission(object):
    FOLLOW = 1
    COMMENT = 2
    WRITE = 4
    MODERATE = 8
    ADMIN = 16
class Role(db.Model):
    # . . .
    def add_permission(self, perm):
        if not self.has_permissions(perm):
            self.permissions += perm

    def remove_permission(self, perm):
        if self.has_permission(perm):
            self.permissions -= perm

    def reset_permissions(self):
        self.permissions = 0

    def has_permission(self, perm):
        return self.permissions & perm == perm
Assigning Roles

- New users should be assigned the default role.
- The first Administrator should be set manually in the database.
- All other upgrades in user roles should be performed by an administrator through an administrator only interface.
from flask_login import UserMixin, AnonymousUserMixin

class User(UserMixin, db.Model):
    # . . .
    def can(self, perm):
        return self.role is not None and \
        self.role.has_permission(perm)

    def is_administrator(self):
        return self.can(Permission.ADMIN)

class AnonymousUser(AnonymousUserMixin):
    def can(self, perm):
        return False

    def is_administrator(self):
        return False
Custom Decorators

from functools import wraps
from flask import abort
from flask_login import current_user

def permission_required(perm):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if not current_user.can(perm):
                abort(403)
            return f(*args, **kwargs)
        return decorated_function

def admin_required(f):
    return permission_required(Permission.ADMIN)(f)
Role Verification

@app.route("/admin")
@login_required
@admin_required
def for_admins_only():
    return "Congratulations on Administrator Privileges!"

@app.route("/moderate")
@login_required
@permission_required(Permission.MODERATE)
def for_moderators_only():
    return "Congratulations on Moderator Privileges!"
Bibliography 1

**Python Docs.**
Expressions.

**Bruce Eckel.**
Decorators.

**Geir Arne Hjelle.**
Primer on python decorators.
https://realpython.com/primer-on-python-decorators/.
Ling Thio.

Data models.


Ling Thio.

Role-based authorization.
