DJANGO ADMIN + AUTH

06016322 - WEB PROGRAMMING

INTRODUCING THE DJANGO ADMIN

https://docs.djangoproject.com/en/2.2/ref/contrib/admin/

- First we'll need to create a user who can login to the admin site. Run the following command:

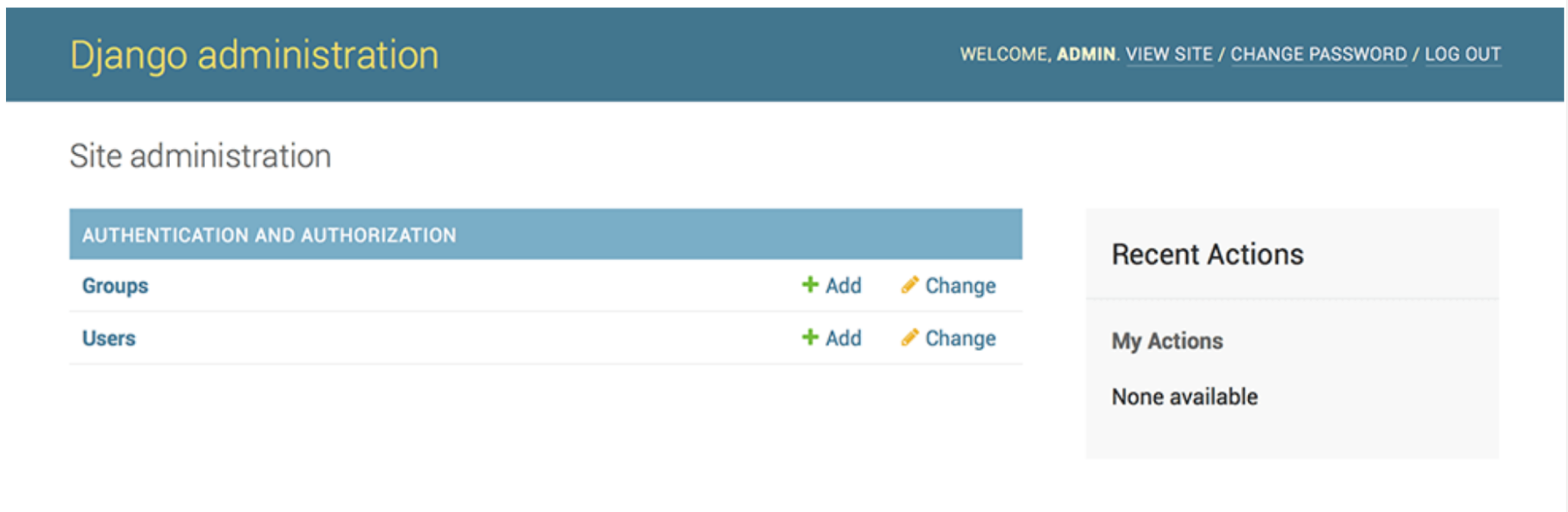
```
$ python manage.py createsuperuser
```

- Start the server

```
$ python manage.py runserver
```

- Now, open a Web browser and go to "/admin/" on your local domain – e.g., http://127.0.0.1:8000/admin/. You should see the admin's login screen:

THE ADMIN SITE



MAKE THE POLL APP MODIFIABLE IN THE ADMIN SITE

- We need to tell the admin that Poll, Question and Answer objects have an admin interface. To do this, open the polls/admin.py file, and edit it to look like this:

polls/admin.py

```
from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

MODELADMIN OBJECTS

- The ModelAdmin class is the representation of a model in the admin interface. Usually, these are stored in a file named admin.py in your application.

```
from django.contrib import admin
from myproject.myapp.models import Author

class AuthorAdmin(admin.ModelAdmin):
    pass
admin.site.register(Author, AuthorAdmin)
```

MODELADMIN OPTIONS

- ModelAdmin.date_hierarchy
- ModelAdmin.fields
  - To display multiple fields on the same line, wrap those fields in their own tuple.
- ModelAdmin.exclude
- ModelAdmin.fieldsets
- ModelAdmin.list_display

```
from django.contrib import admin

class AuthorAdmin(admin.ModelAdmin):
    fields = ('name', 'title')

class AuthorAdmin(admin.ModelAdmin):
    exclude = ('birth_date',)
```

```
class FlatPageAdmin(admin.ModelAdmin):
    fields = (('url', 'title'), 'content')
```

CUSTOMIZE THE ADMIN FORM

- This change makes the "Publication date" come before the "Question" field:

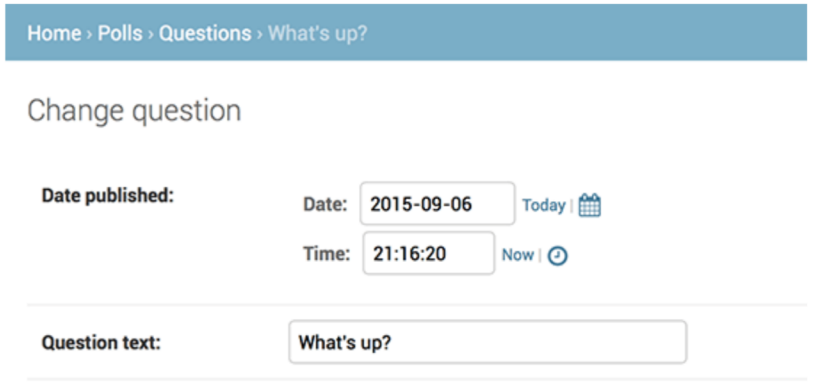
polls/admin.py

```
from django.contrib import admin

from .models import Question


class QuestionAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question_text']

admin.site.register(Question, QuestionAdmin)
```



CUSTOMIZE THE ADMIN FORM

- You might want to split the form up into fieldsets:

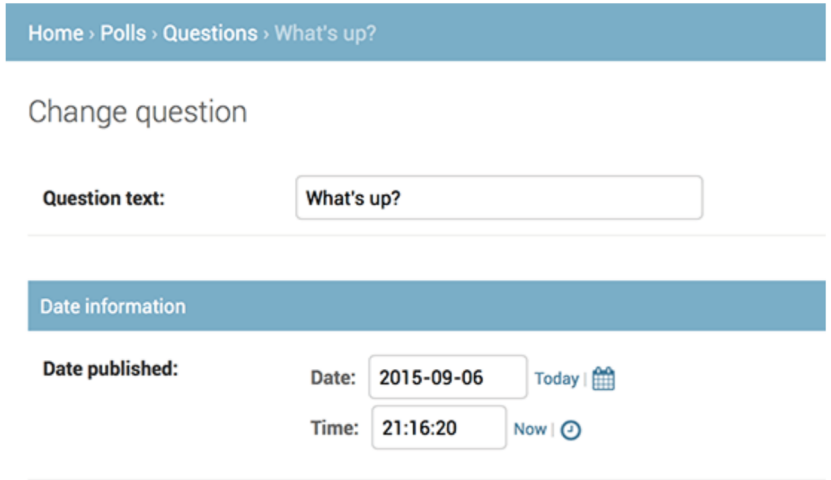
- fieldsets is a list of two-tuples, in which each two-tuple represents a <fieldset> on the admin form page. (A <fieldset> is a "section" of the form.)

```python
from django.contrib import admin

from .models import Question


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date']}),
    ]

admin.site.register(Question, QuestionAdmin)
```



ADDING RELATED OBJECTS

- This tells Django: "Choice objects are edited on the Question admin page. By default, provide enough fields for 3 choices."

- Try change "admin.StackedInline" to "admin.TabularInline"

```
from django.contrib import admin

from .models import Choice, Question


class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes':
['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)
```

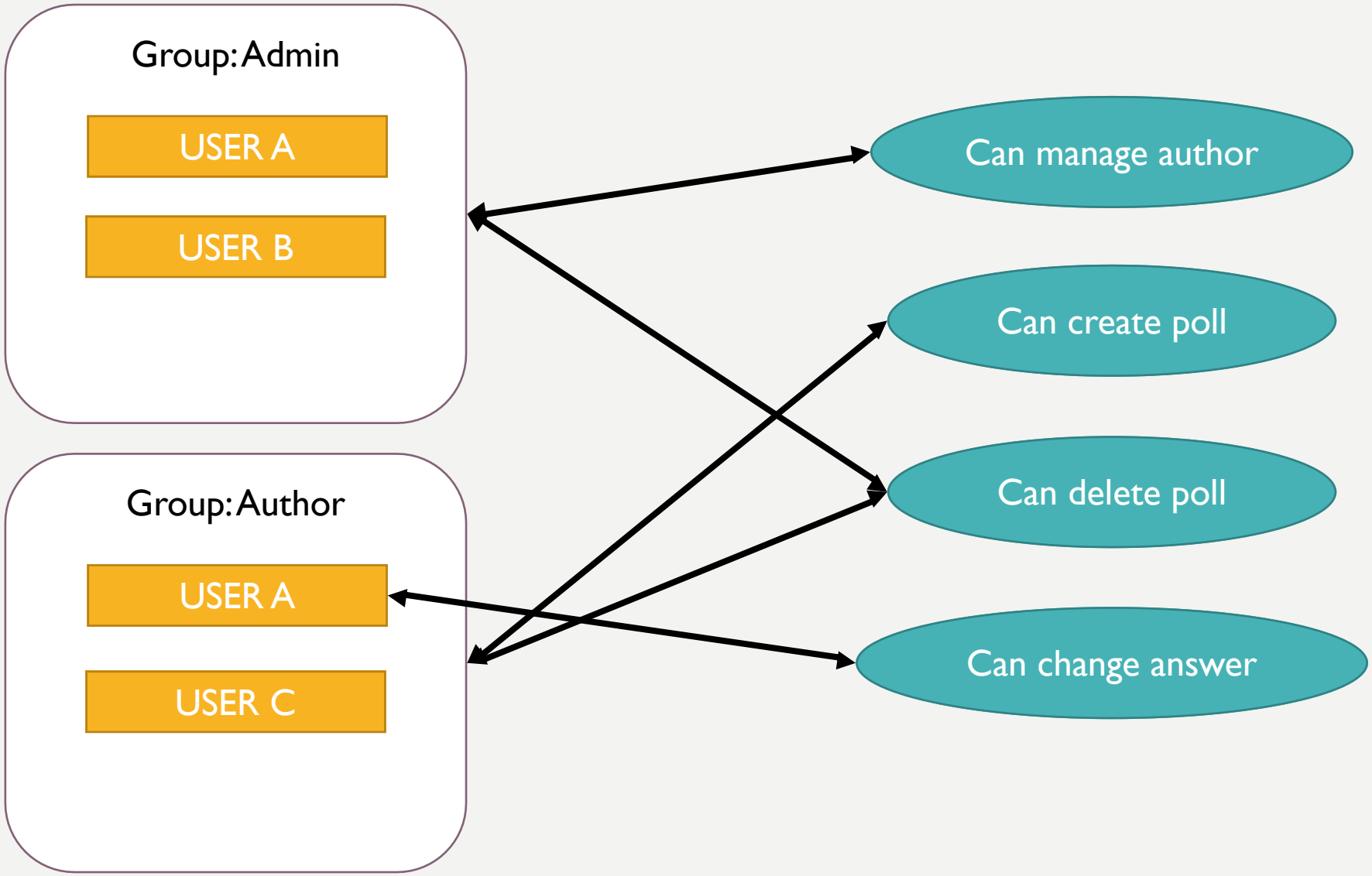
DJANGO AUTHENTICATION

USER AUTHENTICATION IN DJANGO

- The Django authentication system handles both authentication and authorization.

- **Authentication** verifies a user is who they claim to be
- **Authorization** determines what an authenticated user is allowed to do.

- The auth system consists of:
  - Users
  - Permissions
  - Groups
  - Forms and view tools for logging in users, or restricting content


Group: Admin
USER A
USER B
Group: Author
USER A
USER C
Can manage author
Can create poll
Can delete poll
Can change answer

USER OBJECTS

- The primary attributes of the default user are:
  - username
  - password
  - email
  - first_name
  - last_name
  - groups - Many-to-many relationship to Group
  - user_permissions - Many-to-many relationship to Permission
  - is_staff
  - is_active
  - is_superuser
  - last_login
  - date_joined

CREATING USERS

- The most direct way to create users is to use the included create_user() helper function (or you can create in the admin site):

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user('john', 'lennon@thebeatles.com', 'johnpassword')

# At this point, user is a User object that has already been saved
# to the database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = 'Lennon'
>>> user.save()
```

- Create superusers using the **createsuperuser** command:

```
$ python manage.py createsuperuser
```

CHANGING PASSWORDS

- Django does not store raw (clear text) passwords on the user model, but only a hash.

- To change a user's password, you have several options:
  1. manage.py changepassword *username*
  2. set_password() helper function

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(username='john')
>>> u.set_password('new password')
>>> u.save()
```

  3. Use the admin site

AUTHENTICATING USERS

- Use authenticate function:

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # A backend authenticated the credentials
else:
    # No backend authenticated the credentials
```

HOW TO LOG A USER IN

- If you have an authenticated user you want to attach to the current session - this is done with a login() function.

- To log a user in, from a view, use login(). It takes an HttpRequest object and a User object. login() saves the user's ID in the session, using Django's session framework.

```
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username, password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        ...
```

HOW TO LOG A USER OUT

- To log out a user who has been logged in via django.contrib.auth.login(), usedjango.contrib.auth.logout() within your view.

- When you call logout(), the session data for the current request is completely cleaned out. All existing data is removed.

```
from django.contrib.auth import logout

def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

LIMITING ACCESS TO LOGGED-IN USERS

- There are two ways as shown on the right.
- login_required() does the following:
  - If the user isn't logged in, redirect to settings.LOGIN_URL, passing the current absolute path in the query string. Example: /accounts/login/?next=/polls/3/.
  - If the user is logged in, execute the view normally.

```python
from django.conf import settings
from django.shortcuts import redirect

def my_view(request):
    if not request.user.is_authenticated:
        return redirect('%s?next=%s' % (settings.LOGIN_URL, request.path))
    # ...
```

```python
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    ...
```

LIMITING ACCESS TO LOGGED-IN USERS THAT PASS A TEST

- It's a relatively common task to check whether a user has a particular permission. For that reason, Django provides a shortcut for that case: the permission_required() decorator.:

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

@permission_required('polls.can_vote')
def my_view(request):
    ...
```

PERMISSIONS AND AUTHORIZATION

- Django comes with a simple permissions system. It provides a way to assign permissions to specific users and groups of users.

- User objects have two many-to-many fields: groups and user_permissions. User objects can access their related objects in the same way as any other Django model:

```
myuser.groups.set([group_list])
myuser.groups.add(group, group, ...)
myuser.groups.remove(group, group, ...)
myuser.groups.clear()
myuser.user_permissions.set([permission_list])
myuser.user_permissions.add(permission, permission, ...)
myuser.user_permissions.remove(permission, permission, ...)
myuser.user_permissions.clear()
```

DEFAULT PERMISSIONS

- Django creates four default permissions – add, change, delete, and view – for each Django model defined in one of your installed applications.

- Assuming you have an application with an app_label **foo** and a model named **Bar**, to test for basic permissions you should use:
  - add: user.has_perm('foo.add_bar')
  - change: user.has_perm('foo.change_bar')
  - delete: user.has_perm('foo.delete_bar')
  - view: user.has_perm('foo.view_bar')

EXTENDING THE EXISTING USER MODEL

- If you wish to store information related to User, you can use a **OneToOneField** to a model containing the fields for additional information.

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

class Employee(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    department = models.CharField(max_length=100)
```

```
>>> u = User.objects.get(username='fsmith')
>>> freds_department = u.employee.department
```