DJANGO ADMIN COOKBOOK

How to do things with Django admin

Updated for Django 2.0 and Python 3.6

Django Admin Cookbook

DJANGO
ADMIN
COOKBOOK

*How to do things
with Django admin*

**Updated for Django 2.0
and Python 3.6**

Django Admin Cookbook - How to do things with Django admin.

This is a book about doing things with Django admin. It takes the form of about forty questions and common tasks with Django admin we answer.

The chapters are based on a common set of models, which you can read in detail here ([Models used in this book](#)). In short, we have two apps, `events` and `entities`. The models are

- Events: `Epic`, `Event`, `EventHero`, `EventVillian`
- Entities: `Category`, `Origin`, `Hero`, `Villain`

- [Introduction](#)

Text and Design

- [1. How to change 'Django administration' text?](#)
- [2. How to set the plural text for a model?](#)
- [3. How to create two independent admin sites?](#)
- [4. How to remove default apps from Django admin?](#)
- [5. How to add a logo to Django admin?](#)
- [6. How to override Django admin templates?](#)

Calculated fields

Bulk and custom actions

- [1. How to add additional actions in Django admin?](#)
- [2. How to export CSV from Django admin?](#)
- [3. How to remove the delete selected action in Django admin?](#)
- [4. How to add Custom Action Buttons (not actions) to Django Admin list page?](#)
- [5. How to import CSV using Django admin?](#)

Permissions

- [1. How to restrict Django admin to specific users?](#)
- [2. How to restrict access to parts of Django admin?](#)
- [3. How to allow creating only one object from the admin?](#)
- [4. How to remove the 'Add'/'Delete' button for a model?](#)

Multiple models and inlines

Listview Page

- [1. How to show larger number of rows on listview page?](#)
- [2. How to disable django admin pagination?](#)
- [3. How to add date based filtering in Django admin?](#)
- [4. How to show many to many or reverse FK fields on listview page?](#)

Changeview Page

- [1. How to show image from Imagefield in Django admin.](#)
- [2. How to associate model with current user while saving?](#)
- [3. How to mark a field as readonly in admin?](#)
- [4. How to show an uneditable field in admin?](#)
- [5. How to make a field editable while creating, but read only in existing objects?](#)
- [6. How to filter FK dropdown values in django admin?](#)
- [7. How to manage a model with a FK with a large number of objects?](#)
- [8. How to change ForeignKey display text in dropdowns?](#)
- [9. How to add a custom button to Django change view page?](#)

Misc

- [1. How to get Django admin urls for specific objects?](#)
- [2. How to add a model twice to Django admin?](#)
- [3. How to override save behaviour for Django admin?](#)
- [4. How to add a database view to Django admin?](#)
- [5. How to set ordering of Apps and models in Django admin dashboard.](#)

Indices and tables

- [Models used in this book](#)

- [Index](#)

Introduction

Django Admin Cookbook is a book about doing things with Django admin. It is targeted towards intermediate Django developers, who have some experience with Django admin, but are looking to expand their knowledge of Django admin and achieve mastery of Django admin.

It takes the form of question and answers about common tasks you might do with Django admin. All the chapters are based on a common set of models, which you can read in detail here ([Models used in this book](#)). In short, we have two apps, `events` and `entities`. The models are

- Events: `Epic`, `Event`, `EventHero`, `EventVillain`
- Entities: `Category`, `Origin`, `Hero`, `Villain`

How to use this book

You can read this book either from start to end, or search for the things you need to do and only read those chapters. Each chapter focusses on a single, specific task.

In either case, you should read the `entities/models.py` and `events/models.py` first.

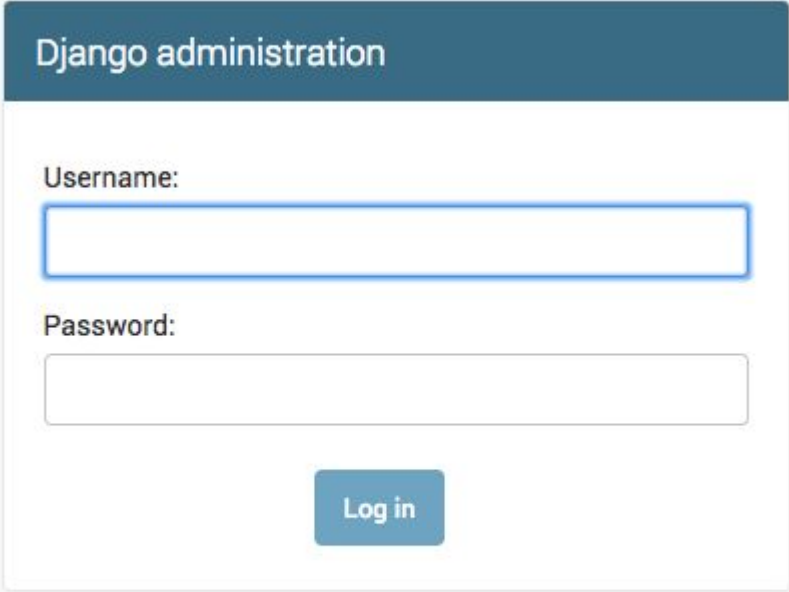# 1. How to change 'Django administration' text?

By default Django admin shows 'Django administration'. You have been asked to replace this with 'UMSRA Administration'

The text is at these pages:

- Login Page
- The listview page
- The HTML title tag

1.1. Login, Listview and Changeview Page

By default it looks like this and is set to `â€œDjango administrationâ€`

`site_header` can be set to change this.

1.2. Listview Page

BY default it looks like this and is set to `â€œSite administrationâ€`

`index_title` can be set to change this.

1.3. HTML title tag

By default it looks like this and is set to `â€œDjango site adminâ€`

`site_title` can be set to change this.

We can make the three changes in urls.py:

```
admin.site.site_header = "UMSRA Admin"
admin.site.site_title = "UMSRA Admin Portal"
admin.site.index_title = "Welcome to UMSRA Researcher Portal"
```

2. How to set the plural text for a model?

By default admin will show the name of your model appended with an â€œsâ€, aka the plural form of your model. It looks like this

You have been asked to set the correct plural spellings: *Categories* and *Heroes*

You can do this by setting the `verbose_name_plural` in your models. Change that in your models.py.:

```python
class Category(models.Model):
    ...

    class Meta:
        verbose_name_plural = "Categories"


class Hero(Entity):
```

```
...

class Meta:
    verbose_name_plural = "Heroes"
```

With the changes your Admin will look like this.

Welcome to UMSRA Researcher Portal

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	✏ Change
Users	+ Add	✏ Change

ENTITIES

Categories	+ Add	✏ Change
Heroes	+ Add	✏ Change
Origins	+ Add	✏ Change
Villains	+ Add	✏ Change

3. How to create two independent admin sites?

The usual way to create admin pages is to put all models in a single admin. However it is possible to have multiple admin sites in a single Django app.

Right now our `entity` and `event` models are in same place. UMSRA has two distinct group researching *Events* and *Entities*, and so wants to split the admins.

We will keep the default admin for *entities* and create a new subclass of `AdminSite` for *events*.

In our `events/admin.py` we do:

```python
from django.contrib.admin import AdminSite
class EventAdminSite(AdminSite):
    site_header = "UMSRA Events Admin"
    site_title = "UMSRA Events Admin Portal"
    index_title = "Welcome to UMSRA Researcher Events Portal"

event_admin_site = EventAdminSite(name='event_admin')


event_admin_site.register(Epic)
event_admin_site.register(Event)
event_admin_site.register(EventHero)
event_admin_site.register(EventVillain)
```

And change the `urls.py` to

```python
from events.admin import event_admin_site


urlpatterns = [
    path('entity-admin/', admin.site.urls),
    path('event-admin/', event_admin_site.urls),
]
```

This separates the admin. Both admins are available at their respective urls, `/entity-admin/` and `event-admin/`.

4. How to remove default apps from Django admin?

Django will include `django.contrib.auth` in `INSTALLED_APPS`, which means *User* and *Groups* models are included in admin automatically.

Welcome to UMSRA Researcher Portal

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	✏ Change
Users	+ Add	✏ Change

ENTITIES		
Categories	+ Add	✏ Change
Heroes	+ Add	✏ Change
Origins	+ Add	✏ Change
Villains	+ Add	✏ Change

If you want to remove it, you will have to unregister them.

```python
from django.contrib.auth.models import User, Group


admin.site.unregister(User)
admin.site.unregister(Group)
```

After making these changes, your admin should look like this.

Welcome to UMSRA Researcher Portal

ENTITIES		
Categories	✚ Add	✏ Change
Heroes	✚ Add	✏ Change
Origins	✚ Add	✏ Change
Villains	✚ Add	✏ Change

5. How to add a logo to Django admin?

Your higher ups at UMSRA love the admin you have created till now, but marketing wants to put the UMSRA logo on all admin pages.

You need to override the default templates provided by Django. In your django settings, you code::*TEMPLATES* setting looks like this.

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

This means that Django will look for templates in a directory called `templates` inside each app, but you can override that by setting a value for `TEMPLATES.DIRS`.

We change the `'DIRS': [],` to `'DIRS': [os.path.join(BASE_DIR, 'templates/')],`, and create the `templates` folder. If your `STATICFILES_DIRS` is empty set it to:

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
```

Now copy the `base_site.html` from the admin app to `templates\admin` folder you just created. Replace thre default text in *branding* block with:

```html
<h1 id="site-name">
    <a href="{% url 'admin:index' %}">
        <img src="{% static 'umsra_logo.png' %}"
height="40px" />
    </a>
</h1>
```

With the changes your base_site.html will look like this:

```html
{% extends "admin/base.html" %}

{% load staticfiles %}

{% block title %}{{ title }} | {{
site_title|default:_('Django site admin') }}{% endblock %}

{% block branding %}
<h1 id="site-name">
    <a href="{% url 'admin:index' %}">
        <img src="{% static 'umsra_logo.png' %}"
height="40px" />
    </a>
</h1>
{% endblock %}

{% block nav-global %}{% endblock %}
```

And your admin will look like this

UMSRA

Welcome to UMSRA Researcher Events Portal

EVENTS		
Epics	**+** Add	✎ Change
Event heros	**+** Add	✎ Change
Event villains	**+** Add	✎ Change
Events	**+** Add	✎ Change

6. How to override Django admin templates?

https://docs.djangoproject.com/en/dev/ref/contrib/admin/#overriding-admin-templates

1. How to show calculated fields on listview page?

You have an admin for the `Origin` model like this:

```python
@admin.register(Origin)
class OriginAdmin(admin.ModelAdmin):
    list_display = ("name",)
```

Apart from the name, we also want to show the number of heroes and number of villains for each origin, which is not a DB field on `Origin`. You can do this in two ways.

1.1. Adding a method to the model

You can add two methods to your `Origin` model like this:

```python
def hero_count(self,):
    return self.hero_set.count()

def villain_count(self):
    return self.villain_set.count()
```

And change `list_display` to `list_display = ("name", "hero_count", "villain_count")`.

1.2. Adding a method to the ModelAdmin

If you donâ€™t want to add method to the model, you can do instead add the method to the ModelAdmin.

```python
def hero_count(self, obj):
    return obj.hero_set.count()

def villain_count(self, obj):
    return obj.villain_set.count()
```

The `list_display`, as earlier, changes to `list_display = ("name", "hero_count", "villain_count")`.

1.3. Performance considerations for calculated_fields

With either of the above approaches, you would be running two exta queries per object (One per calculated field). You can find how to optimize this in How to optimize queries in Django admin?.

With any of these changes your admin looks like this:

2. How to optimize queries in Django admin?

If you have a lot of calculated fields in your admin, you can be running multiple queries per object leading to your admin can becoming quite slow. To fix this you can override the `get_queryset` method on model admin to annotate the calculated fields.

Lets take the example of this `ModelAdmin` we have for `Origin`:

```python
@admin.register(Origin)
class OriginAdmin(admin.ModelAdmin):
    list_display = ("name", "hero_count", "villain_count")

    def hero_count(self, obj):
        return obj.hero_set.count()

    def villain_count(self, obj):
        return obj.villain_set.count()
```

This adds two extra queries per row in your listview page. To fix this you can override the `get_queryset` to annotate the counted fields, and then use the annotated fields in your ModelAdmin methods.

With the changes, your ModelAdmin field looks like this:

```python
@admin.register(Origin)
class OriginAdmin(admin.ModelAdmin):
    list_display = ("name", "hero_count", "villain_count")

    def get_queryset(self, request):
        queryset = super().get_queryset(request)
        queryset = queryset.annotate(
            _hero_count=Count("hero", distinct=True),
            _villain_count=Count("villain", distinct=True),
        )
        return queryset

    def hero_count(self, obj):
```

```
        return obj._hero_count

    def villain_count(self, obj):
        return obj._villain_count
```

There are no per object extra queries. Your admin continues to look like it did before the `annotate` call.

3. How to enable sorting on calculated fields?

Django adds sorting capabilities on fields which are attributes on the models. When you add a calculated field Django doesnâ€™t know how to do a `order_by`, so it doesnâ€™t add sorting capability on that field.

If you want to add sorting on a calculated field, you have to tell Django what to pass to `order_by`. You can do this by setting the `admin_order_field` attribute on the calculated field method.

You start from the admin you wrote in the previous chapter ([How to optimize queries in Django admin?](#)).:

```
hero_count.admin_order_field = '_hero_count'
villain_count.admin_order_field = '_villain_count'
```

With these changes your admin becomes:

```python
@admin.register(Origin)
class OriginAdmin(admin.ModelAdmin):
    list_display = ("name", "hero_count", "villain_count")

    def get_queryset(self, request):
        queryset = super().get_queryset(request)
        queryset = queryset.annotate(
            _hero_count=Count("hero", distinct=True),
            _villain_count=Count("villain", distinct=True),
        )
        return queryset

    def hero_count(self, obj):
        return obj._hero_count

    def villain_count(self, obj):
        return obj._villain_count

    hero_count.admin_order_field = '_hero_count'
    villain_count.admin_order_field = '_villain_count'
```

Here is the admin sorted on `hero_count`

Select origin to change

ADD ORIGIN +

Action: --------- [Go] 0 of 3 selected

	NAME	HERO COUNT		VILLAIN COUNT
☐	India	2	▼	1
☐	Norse	1		1
☐	Greece	1		0

3 origins

4. How to enable filtering on calculated fields?

You have a `Hero` admin which looks like this:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin):
    list_display = ("name", "is_immortal", "category",
"origin", "is_very_benevolent")
    list_filter = ("is_immortal", "category", "origin",)

    def is_very_benevolent(self, obj):
        return obj.benevolence_factor > 75
```

It has one calculated field `is_very_benevolent`, and your admin looks like this

You have added filtering on the fields which come from the models, but you also want to add filtering on the calculated field. To do this, you will need to subclass `SimpleListFilter` like this:

```python
class IsVeryBenevolentFilter(admin.SimpleListFilter):
    title = 'is_very_benevolent'
    parameter_name = 'is_very_benevolent'

    def lookups(self, request, model_admin):
        return (
            ('Yes', 'Yes'),
            ('No', 'No'),
        )

    def queryset(self, request, queryset):
        value = self.value()
        if value == 'Yes':
            return queryset.filter(benevolence_factor__gt=75)
        elif value == 'No':
            return
queryset.exclude(benevolence_factor__gt=75)
        return queryset
```

And then change your `list_filter` to `list_filter = ("is_immortal", "category", "origin", IsVeryBenevolentFilter)`.

With this you can filter on the calculated field, and your admin looks like this:

Select hero to change

ADD HERO +

Action: ‒‒‒‒‒‒‒‒‒ ⇕ | Go | 0 of 2 selected

	NAME	IS IMMORTAL	CATEGORY	ORIGIN	IS VERY BENEVOLENT
☐	**Thor**	✖	God	Norse	True
☐	**Vishnu**	✔	God	India	True

2 Heroes

FILTER

By is immortal

All
Yes
No

By category

All
God
Avatar
Demi God

By origin

All
India
Greece
Norse

By is_very_benevolent

All
Yes
No

5. How to show â€œonâ€ or â€œoffâ€ icons for calculated boolean fields?

In the previous chapter, [How to enable filtering on calculated fields?](#) you added a boolean field.:

```python
def is_very_benevolent(self, obj):
    return obj.benevolence_factor > 75
```

Which looks like this

The `is_very_benevolent` field shows the string *True* and *False*, unlike the builtin BooleanFields which show an on and off indicator. To fix this, you add a `boolean` attribute on your method. You final modeladmin looks like this:

```
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin):
    list_display = ("name", "is_immortal", "category",
"origin", "is_very_benevolent")
    list_filter = ("is_immortal", "category", "origin",
IsVeryBenevolentFilter)

    def is_very_benevolent(self, obj):
        return obj.benevolence_factor > 75

    is_very_benevolent.boolean = True
```

And your admin looks like this

	NAME	IS IMMORTAL	CATEGORY	ORIGIN	IS VERY BENEVOLENT
☐	Thor	✗	God	Norse	✓
☐	Achilles	✗	Demi God	Greece	✗
☐	Vishnu	✓	God	India	✓
☐	Krishna	✗	Avatar	India	✗

4 Heroes

1. How to add additional actions in Django admin?

Django admin allows you to add additional actions which allow you to do bulk actions. You have been asked to add an action which will mark multiple `Hero` s as immortal.

You can do this by adding the action as method to ModelAdmin and adding the method as a string to `actions`

```python
actions = ["mark_immortal"]

def mark_immortal(self, request, queryset):
    queryset.update(is_immortal=True)
```

2. How to export CSV from Django admin?

You have been asked to add ability to export `Hero` and `Villain` from the admin. There are a number of third party apps which allow doing this, but its quite easy without adding another dependency. You will add an admin action to `HeroAdmin` and `VillanAdmin`.

An admin action always has this signature `def admin_action(modeladmin, request, queryset):`, alternatively you can add it directly as a method on the `ModelAdmin` like this:

```python
class SomeModelAdmin(admin.ModelAdmin):

    def admin_action(self, request, queryset):
```

To add csv export to `HeroAdmin` you can do something like this:

```python
actions = ["export_as_csv"]

def export_as_csv(self, request, queryset):
    pass

export_as_csv.short_description = "Export Selected"
```

This adds an action called export selected, which looks like this:

Select hero to change

Action: ☑ ---------   Go   0 of 4 selected
        Delete selected Heroes
        Export Selected

☐	NA		IS IMMORTAL
☐	**Thor**		✖
☐	**Achilles**		✖
☐	**Vishnu**		✔
☐	**Krishna**		✖

4 Heroes

You will then change the `export_as_csv` to this:

```python
import csv
from django.http import HttpResponse
...

def export_as_csv(self, request, queryset):

    meta = self.model._meta
    field_names = [field.name for field in meta.fields]

    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename=
{}.csv'.format(meta)
    writer = csv.writer(response)

    writer.writerow(field_names)
    for obj in queryset:
        row = writer.writerow([getattr(obj, field) for field
in field_names])

    return response
```

This exports all of the selected rows. If you notice, `export_as_csv` doensâ€™t have anything specific to `Hero`, so you can extract the method to a mixin.

With the changes, your code looks like this:

```python
class ExportCsvMixin:
    def export_as_csv(self, request, queryset):

        meta = self.model._meta
        field_names = [field.name for field in meta.fields]

        response = HttpResponse(content_type='text/csv')
        response['Content-Disposition'] = 'attachment; filename={}.csv'.format(meta)
        writer = csv.writer(response)

        writer.writerow(field_names)
        for obj in queryset:
            row = writer.writerow([getattr(obj, field) for field in field_names])

        return response

    export_as_csv.short_description = "Export Selected"


@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    list_display = ("name", "is_immortal", "category", "origin", "is_very_benevolent")
    list_filter = ("is_immortal", "category", "origin", IsVeryBenevolentFilter)
    actions = ["export_as_csv"]

...


@admin.register(Villain)
class VillainAdmin(admin.ModelAdmin, ExportCsvMixin):
    list_display = ("name", "category", "origin")
    actions = ["export_as_csv"]
```

You can add such an export to other models by subclassing from `ExportCsvMixin`

3. How to remove the delete selected action in Django admin?

By default Django adds a *Delete Selected* action to the listview page. You have been asked to remove the action from the `Hero` admin.

The method `ModelAdmin.get_actions` returns the actions shown. By overriding this method, to remove `delete_selected` We can remove it form the dropdown. Your code looks like this with the changes.:

```python
def get_actions(self, request):
    actions = super().get_actions(request)
    if 'delete_selected' in actions:
        del actions['delete_selected']
    return actions
```

And your admin looks like this

You should also read [How to remove the 'Add'/'Delete' button for a model?](#).

4. How to add Custom Action Buttons (not actions) to Django Admin list page?

UMSRA has decided that given sufficient kryptonite, all Heroes are mortal. However, they want to be able to change their mind and say all heroes are immortal.

You have been asked to add two buttons - One which makes all heroes mortal, and one which makes all immortal. Since it affects all heroes irrespective of the selection, this needs to be a separate button, not an action dropdown.

First, we will change the template on the `HeroAdmin` so we can add two buttons.:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    change_list_template = "entities/heroes_changelist.html"
```

Then we will override the `get_urls`, and add the `set_immortal` and `set_mortal` methods on the model admin. They will serve as the two view methods.:

```python
def get_urls(self):
    urls = super().get_urls()
    my_urls = [
        path('immortal/', self.set_immortal),
        path('mortal/', self.set_mortal),
    ]
    return my_urls + urls

def set_immortal(self, request):
    self.model.objects.all().update(is_immortal=True)
    self.message_user(request, "All heroes are now immortal")
    return HttpResponseRedirect("../")

def set_mortal(self, request):
```

```
    self.model.objects.all().update(is_immortal=False)
    self.message_user(request, "All heroes are now mortal")
    return HttpResponseRedirect("../")
```

Finally, we create the `entities/heroes_changelist.html` template by extending the `admin/change_list.html`.:

```
{% extends 'admin/change_list.html' %}

{% block object-tools %}
    <div>
        <form action="immortal/" method="POST">
            {% csrf_token %}
                <button type="submit">Make Immortal</button>
        </form>
        <form action="mortal/" method="POST">
            {% csrf_token %}
                <button type="submit">Make Mortal</button>
        </form>
    </div>
    <br />
    {{ block.super }}
{% endblock %}
```

Select hero to change

Make Immortal
Make Mortal

Action: --------- Go   0 of 5 selected

And after using the *make_mortal* action, the Heroes are all mortal and you see this message.

✅ All heroes are now mortal

Select hero to change

Make Immortal
Make Mortal

Action: ‑‑‑‑‑‑‑‑‑ ⬍ | Go | 0 of 5 selected

	NAME	IS IMMORTAL	CATEGORY	ORIGIN	IS VERY BENEVOLENT
☐	**Zeus**	❌	God	Greece	❌
☐	**Thor**	❌	God	Norse	✅
☐	**Achilles**	❌	Demi God	Greece	❌
☐	**Vishnu**	❌	God	India	✅
☐	**Krishna**	❌	Avatar	India	❌

5 Heroes

5. How to import CSV using Django admin?

You have been asked to allow csv imports on the `Hero` admin. You will do this by adding a link to the `Hero` changelist page, which will take to a page with an upload form. You will write a handler for the *POST* action to create the objects from the csv.:

```python
class CsvImportForm(forms.Form):
    csv_file = forms.FileField()


@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    change_list_template = "entities/heroes_changelist.html"

    def get_urls(self):
        urls = super().get_urls()
        my_urls = [
            ...
            path('import-csv/', self.import_csv),
        ]
        return my_urls + urls

    def import_csv(self, request):
        if request.method == "POST":
            csv_file = request.FILES["csv_file"]
            reader = csv.reader(csv_file)
            # Create Hero objects from passed in data
            # ...
            self.message_user(request, "Your csv file has
been imported")
            return redirect("..")
        form = CsvImportForm()
        payload = {"form": form}
        return render(
            request, "admin/csv_form.html", payload
        )
```

Then you create the `entities/heroes_changelist.html` template, by overriding the `admin/change_list.html` template like this.:

```
{% extends 'admin/change_list.html' %}

{% block object-tools %}
    <a href="import-csv/">Import CSV</a>
    <br />
    {{ block.super }}
{% endblock %}
```

Finally you create the `csv_form.html` like this.:

```
{% extends 'admin/base.html' %}

{% block content %}
    <div>
        <form action="." method="POST"
enctype="multipart/form-data">
            {{ form.as_p }}
            {% csrf_token %}

                <button type="submit">Upload CSV</button>
        </form>
    </div>
    <br />

{% endblock %}
```

With these changes, you get a link on the `Hero` changelist page.

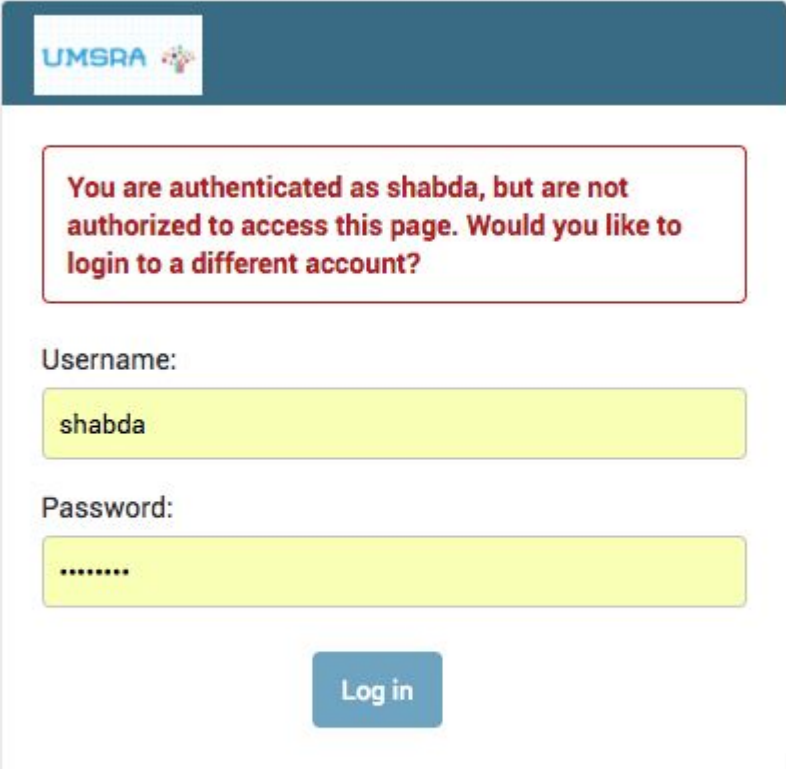And the import form apge looks like this.

Csv file: Choose File | No file chosen

Upload CSV

1. How to restrict Django admin to specific users?

Django admin allows access to users marked as `is_staff=True`. To disable a user from being able to access the admin, you should set `is_staff=False`.

This holds true even if the user is a superuser. `is_superuser=True`. If a non-staff tries to access the admin, they see a message like this.

2. How to restrict access to parts of Django admin?

You can enable and restrict access to specific parts of Django admin using the permission system. When a model is added, by default, Django creates three permissions. `add, change and delete`

Admin uses these permissions to decide access for users. For a user with `is_superuser=False`, and no permissions, the admin looks like this

Welcome to UMSRA Researcher Portal

You don't have permission to edit anything.

If you add a permission
`user.user_permissions.add(Permission.objects.get(codename="add_hero"))`, the admin starts looking like this

Entities administration

ENTITIES	
Heroes	+ Add

You can add more complex logic to restrict access by changing these methods:

```python
def has_add_permission(self, request):
    ...

def has_change_permission(self, request, obj=None):
    ...

def has_delete_permission(self, request, obj=None):
    ...

def has_module_permission(self, request):
    ...
```

3. How to allow creating only one object from the admin?

The UMSRA admin has asked you to limit the number of categories to only one. They want every entity to be of the same category.

You can do this by:

```python
MAX_OBJECTS = 1

def has_add_permission(self, request):
    if self.model.objects.count() >= MAX_OBJECTS:
        return False
    return super().has_add_permission(request)
```

This would hide the add buton as soon as one object is created. You can set `MAX_OBJECTS` to any value to ensure that larger number of objects canâ€™t be ceated than `MAX_OBJECTS`.

4. How to remove the â€˜Addâ€™/ â€™Deleteâ€™ button for a model?

The UMSRA management has added all the Category and Origin objects and wants to disable any further addition and deletion. They have asked you to disable â€˜Addâ€™ and â€˜Deleteâ€™ buttons. You can do this by overriding the `has_add_permission` and `has_delete_permission` in the Django admin.:

```
def has_add_permission(self, request):
    return False

def has_delete_permission(self, request, obj=None):
    return False
```

With these changes, the admin looks like this

Note the removed *Add* buttons. The add and delete buttons also get removed from the detail pages. You can also read [How to remove the delete selected action in Django admin?](#).

1. How to edit mutiple models from one Django admin?

To be able to edit multiple objects from one Django admin, you need to use inlines.

You have the `Category` model, and you need to add and edit `Villain` models inside the admin for Category. You can do:

```python
class VillainInline(admin.StackedInline):
    model = Villain

@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    ...

    inlines = [VillainInline]
```

You can see the form to add and edit `Villain` inside the `Category` admin. If the Inline model has alot of fields, use `StackedInline` else use `TabularInline`.

2. How to add One to One relation as admin inline?

OneToOneFields can be set as inlines in the same way as a FK. However, only one side of the OneToOneField can be set as the inline model.

You have a `HeroAcquaintance` model which has a One to one relation to hero like this.:

```
class HeroAcquaintance(models.Model):
    "Non family contacts of a Hero"
    hero = models.OneToOneField(Hero,
on_delete=models.CASCADE)
    ....
```

You can add this as inline to Hero like this:

```
class HeroAcquaintanceInline(admin.TabularInline):
    model = HeroAcquaintance


@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    inlines = [HeroAcquaintanceInline]
```

Headshot:	Choose File No file chosen
Father:	Zeus ✎ + ✕
Mother:	--------- ✎ + ✕
Spouse:	--------- ✎ + ✕
Headshot image:	-

HERO ACQUAINTANCES

FRIENDS	DETRACTORS	MAIN ANATAGONISTS	DELETE?
Krishna Vishnu Achilles Thor	Krishna Vishnu Achilles Thor	Ravana Fenrir	
+	+	+	

Delete   Save and add another   Save and continue editing   SAVE

3. How to add nested inlines in Django admin?

You have yor models defined like this:

```
class Category(models.Model):
    ...

class Hero(models.Model):
    category = models.ForeignKey(Catgeory)
    ...

class HeroAcquaintance(models.Model):
    hero = models.OneToOneField(Hero,
on_delete=models.CASCADE)
    ...
```

You want to have one admin page to create `Category`, `Hero` and `HeroAcquaintance` objects. However, Django doesn't support nested inline with Foreign Keys or One To One relations which span more than one levels. You have a few options,

You can change the `HeroAcquaintance` model, so that it has a direct FK to `Category`, something like this:

```
class HeroAcquaintance(models.Model):
    hero = models.OneToOneField(Hero,
on_delete=models.CASCADE)
    category = models.ForeignKey(Category)

    def save(self, *args, **kwargs):
        self.category = self.hero.category
        super().save(*args, **kwargs)
```

Then you can attach `HeroAcquaintanceInline` to `CategoryAdmin`, and get a kind of nested inline.

Alternatively, there are a few third party Django apps which allow nested inline. A quick Github or DjangoPackages search will find one which suits your needs and tastes.

4. How to create a single Django admin from two different models?

`Hero` has a FK to `Category`, so you can select a category from Hero admin. If you want to also be able to create `Category` objects from Hero admin, you can change the form for Hero admin, and customise the `save_model` behaviour.:

```python
class HeroForm(forms.ModelForm):
    category_name = forms.CharField()

    class Meta:
        model = Hero
        exclude = ["category"]


@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    form = HeroForm
    ....

    def save_model(self, request, obj, form, change):
        category_name = form.cleaned_data["category_name"]
        category, _ =
Category.objects.get_or_create(name=category_name)
        obj.category = category
        super().save_model(request, obj, form, change)
```

With this change, your admin looks like below and has allows creating or updating category from the Hero admin.

Father:	Zeus ⬍	✏️ ➕ ✖️
Mother:	--------- ⬍	✏️ ➕ ✖️
Spouse:	--------- ⬍	✏️ ➕ ✖️
Category name:		
Headshot image:	-	

1. How to show larger number of rows on listview page?

You have been asked to increase the number of heroes one can see on a single page to 250. (The default is 100). You can do this by:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    list_per_page = 250
```

You can also set it to a smaller value. If we set it to 1 as `list_per_page = 1` the admin looks like this.

2. How to disable django admin pagination?

If you want to completely disable pagination on a admin listview page, you can do this.

```python
import sys
...

@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...

    list_per_page = sys.maxsize
```

You can also read [How to show larger number of rows on listview page?](#).

3. How to add date based filtering in Django admin?

You can add a date based filtering on any date field by setting the `date_hierarchy`.:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    date_hierarchy = 'added_on'
```

It looks like this:

This can be very costly with a large number of objects. As an alternative, you can subclass `SimpleListFilter`, and allow filtering only on years or the months.

4. How to show many to many or reverse FK fields on listview page?

For heroes you can track their father using this field:

```python
father = models.ForeignKey(
    "self", related_name="children", null=True, blank=True,
on_delete=models.SET_NULL
)
```

You have been asked to show the childeren of each Hero, on the listview page. Hero objects have the `children` reverse FK attribute, but you can’t add that to the :code`:*list_display*. You need to add an attribute to ModelAdmin and use that in `list_display`. You can do it like this:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...

    def children_display(self, obj):
        return ", ".join([
            child.name for child in obj.children.all()
        ])
    children_display.short_description = "Children"
```

You will see a column for children like this:

Select hero to change

Make Immortal
Make Mortal

Action: | --------- ▲ | Go | 0 of 7 selected

☐	NAME	IS IMMORTAL	CATEGORY	ORIGIN	IS VERY BENEVOLENT	CHILDREN
☐	**Apollo**	✅	God	Greece	✅	-
☐	**Athena**	✅	God	Greece	❌	-
☐	**Zeus**	❌	God	Greece	❌	Athena, Apollo
☐	**Thor**	❌	God	Norse	✅	-
☐	**Achilles**	❌	Demi God	Greece	❌	-
☐	**Vishnu**	❌	God	India	✅	-
☐	**Krishna**	❌	Avatar	India	❌	-

7 Heroes
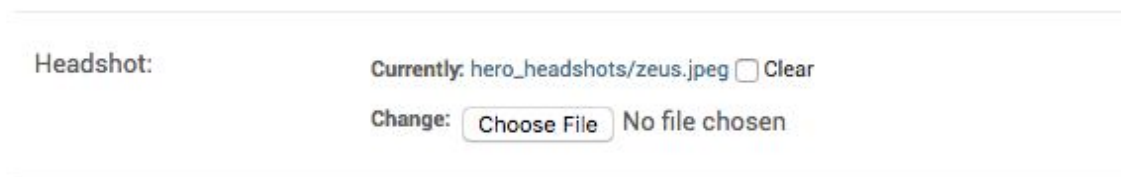
You can use the same method for M2M relations as well. You should also read How to get Django admin urls for specific objects?.

1. How to show image from Imagefield in Django admin.

In your `Hero` model, you have an image field.:

```
headshot = models.ImageField(null=True, blank=True,
upload_to="hero_headshots/")
```

By default it shows up like this:

You have been asked to change it to that the actual image also shows up on the change page. You can do it likethis:

```
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):

    readonly_fields = [..., "headshot_image"]

    def headshot_image(self, obj):
        return mark_safe('<img src="{url}" width="{width}"
height={height} />'.format(
            url = obj.headshot.url,
            width=obj.headshot.width,
            height=obj.headshot.height,
            )
    )
```

With this change, your imagefield looks like this:

Headshot:	**Currently:** hero_headshots/zeus.jpeg ☐ Clear
	Change: [Choose File] No file chosen
Father:	-
Mother:	-
Spouse:	-
Headshot image:	

Delete

2. How to associate model with current user while saving?

The `Hero` model has the following field.:

```
added_by = models.ForeignKey(settings.AUTH_USER_MODEL,
        null=True, blank=True, on_delete=models.SET_NULL)
```

You want the `added_by` field to be automatically set to current user whenever object is created from admin. You can do this.:

```python
def save_model(self, request, obj, form, change):
    if not obj.pk:
        # Only set added_by during the first save.
        obj.added_by = request.user
    super().save_model(request, obj, form, change)
```

If instead you wanted to always save the current user, you can do.:

```python
def save_model(self, request, obj, form, change):
    obj.added_by = request.user
    super().save_model(request, obj, form, change)
```

If you also want to hide the `added_by` field to not show up on the change form, you can do.:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    exclude = ['added_by',]
```

3. How to mark a field as readonly in admin?

UMSRA has temporarily decided to stop tracking the family trees of mythological entities. You have been asked to make the `father`, `mother` and `spouse` fields readonly.

You can do this by:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    readonly_fields = ["father", "mother", "spouse"]
```

Your create form looks like this:

4. How to show an uneditable field in admin?

If you have a field with `editable=False` in your model, that field, by default, is hidden in the change page. This also happens with any field marked as `auto_now` or `auto_now_add`, because that sets the `editable=False` on these fields.

If you want these fields to show up on the change page, you can add them to `readonly_fields`.:

```
@admin.register(Villain)
class VillainAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    readonly_fields = ["added_on"]
```

With this change the Villain admin looks like this:

5. How to make a field editable while creating, but read only in existing objects?

You need to make the `name` and `category` read only once a `Hero` is created. However duing the first write the fields needs to be editable.

You can do this by overriding `get_readonly_fields` method, like this:

```python
def get_readonly_fields(self, request, obj=None):
    if obj:
        return ["name", "category"]
    else:
        return []
```

`obj` is `None` during the object creation, but set to the object being edited during an edit.

6. How to filter FK dropdown values in django admin?

Your `Hero` model has a FK to `Category`. So all category objects will show in the admin dropdown for category. If instead, you wanted to see only a subset, Django allows you to customize that by overriding `formfield_for_foreignkey`:

```
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    def formfield_for_foreignkey(self, db_field, request,
**kwargs):
        if db_field.name == "category":
            kwargs["queryset"] =
Category.objects.filter(name__in=['God', 'Demi God'])
        return super().formfield_for_foreignkey(db_field,
request, **kwargs)
```

7. How to manage a model with a FK with a large number of objects?

You can create a large number of categories like this:

```
categories = [Category(**{"name": "cat-{}".format(i)}) for i
in range(100000)]
Category.objects.bulk_create(categories)
```

Now as `Category` has more than 100000 objects, when you go to the `Hero` admin, it will have category dropdown with 100000 selections. This will make the page both slow and the dropdown hard to use.

You can change how admin handles it by setting the `raw_id_fields`:

```
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    raw_id_fields = ["category"]
```

This change the Hero admin to look like:

Add the popup looks like this

cat-99905

cat-99904

cat-99903

cat-99902

cat-99901

cat-99900

8. How to change ForeignKey display text in dropdowns?

`Hero` has a FK to `Catgeory`. In the dropdown, rather than just the name, you want to show the text "Category: <name>".

You can change the `__str__` method on `Category`, but you only want this change in the admin. You can do this by creating a subclassing `forms.ModelChoiceField` with a custom `label_from_instance`.:

```python
class CategoryChoiceField(forms.ModelChoiceField):
    def label_from_instance(self, obj):
        return "Category: {}".format(obj.name)
```

You can then override `formfield_for_foreignkey` to use this field type for category.:

```python
def formfield_for_foreignkey(self, db_field, request, **kwargs):
    if db_field.name == 'category':
        return CategoryChoiceField(queryset=Category.objects.all())
    return super().formfield_for_foreignkey(db_field, request, **kwargs)
```

Your admin look like this.

Name: Apollo

Alternative name: [   ]

✓ Category: God
Category: Avatar
Category: Demi God
Category: Half God

Category:

Origin:

Gender: Male ⇕

9. How to add a custom button to Django change view page?

`Villain` has a field called `is_unique`:

```python
class Villain(Entity):
    ...
    is_unique = models.BooleanField(default=True)
```

You want to add a button on Villain change form page called "Make Unique", which make this Villain unique. Any other villain with the same name should be deleted.

You start by extending the `change_form` to add a new button.:

```html
{% extends 'admin/change_form.html' %}

{% block submit_buttons_bottom %}
    {{ block.super }}
    <div class="submit-row">
            <input type="submit" value="Make Unique"
name="_make-unique">
    </div>
{% endblock %}
```

Then you can override `response_change` and connect your template to the `VillainAdmin`.:

```python
@admin.register(Villain)
class VillainAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...
    change_form_template = "entities/villain_changeform.html"


    def response_change(self, request, obj):
        if "_make-unique" in request.POST:
            matching_names_except_this =
self.get_queryset(request).filter(name=obj.name).exclude(pk=obj.id)
            matching_names_except_this.delete()
```

```python
            obj.is_unique = True
            obj.save()
            self.message_user(request, "This villain is now
unique")
            return HttpResponseRedirect(".")
        return super().response_change(request, obj)
```

This is how your admin looks now.

1. How to get Django admin urls for specific objects?

You have a children column displaying the names of each heroesâ€™ children. You have been asked to link each children to the change page. You can do it like this.:

```python
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    ...

    def children_display(self, obj):
        display_text = ", ".join([
            "<a href={}>{}</a>".format(
                reverse('admin:
{}_{}_change'.format(obj._meta.app_label,
obj._meta.model_name),
                    args=(child.pk,)),
                child.name)
            for child in obj.children.all()
        ])
        if display_text:
            return mark_safe(display_text)
        return "-"
```

The `reverse('admin:{}_{}_change'.format(obj._meta.app_label, obj._meta.model_name), args=(child.pk,))`, gives the change url for an object.

The other options are

- Delete: `reverse('admin: {}_{}_delete'.format(obj._meta.app_label, obj._meta.model_name), args=(child.pk,))`
- History: `reverse('admin: {}_{}_history'.format(obj._meta.app_label, obj._meta.model_name), args=(child.pk,))`

2. How to add a model twice to Django admin?

You need to add the `Hero` model twice to the admin, one as a regular admin area, and one as read only admin. (Some user will potentially see only the read only admin.)

If you have try to register the same model twice:

```
admin.site.register(Hero)
admin.site.register(Hero)
```

you will get an error like this:

```
raise AlreadyRegistered('The model %s is already registered'
% model.__name__)
```

THe solution is to sublass the `Hero` model as a ProxyModel.:

```python
# In models.py
class HeroProxy(Hero):

    class Meta:
        proxy = True


...
# In admin.py
@admin.register(Hero)
class HeroAdmin(admin.ModelAdmin, ExportCsvMixin):
    list_display = ("name", "is_immortal", "category",
"origin", "is_very_benevolent")
    ....


@admin.register(HeroProxy)
class HeroProxyAdmin(admin.ModelAdmin):
    readonly_fields = ("name", "is_immortal", "category",
"origin",
        ...)
```

3. How to override save behaviour for Django admin?

`ModelAdmin` has a `save_model` method, which is used for creating and updating model objects. By overriding this, you can customize the save behaviour for admin.

The `Hero` model has the following field.:

```
added_by = models.ForeignKey(settings.AUTH_USER_MODEL,
        null=True, blank=True, on_delete=models.SET_NULL)
```

If you want to always save the current user whenever the Hero is updated, you can do.:

```
def save_model(self, request, obj, form, change):
    obj.added_by = request.user
    super().save_model(request, obj, form, change)
```

4. How to add a database view to Django admin?

You have a database view, created as this:

```sql
create view entities_entity as
    select id, name from entities_hero
    union
    select 10000+id as id, name from entities_villain
```

It has all the names from `Hero` and `Villain`. The id's for Villain are set to `10000+id as id` because we don't intend to cross 10000 Heroes:

```
sqlite> select * from entities_entity;
1|Krishna
2|Vishnu
3|Achilles
4|Thor
5|Zeus
6|Athena
7|Apollo
10001|Ravana
10002|Fenrir
```

Then you add a `managed=False` model:

```python
class AllEntity(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        managed = False
        db_table = "entities_entity"
```

And add it to admin.:

```python
@admin.register(AllEntity)
class AllEntiryAdmin(admin.ModelAdmin):
    list_display = ("id", "name")
```

And your admin looks like this

	ID	NAME
☐	**10002**	Fenrir
☐	**10001**	Ravana
☐	**7**	Apollo
☐	**6**	Athena
☐	**5**	Zeus
☐	**4**	Thor
☐	**3**	Achilles
☐	**2**	Vishnu
☐	**1**	Krishna

Action: --------- ⬍ Go   0 of 9 selected

9 all entitys

5. How to set ordering of Apps and models in Django admin dashboard.

Django, by default, orders the models in admin alphabetically. So the order of models in `Event` admin is Epic, EventHero, EventVillain, Event

Instead you want the order to be

- EventHero, EventVillain, Epic then event.

The template used to render the admin index page is `admin/index.html` and the view function is `ModelAdmin.index`.

```python
def index(self, request, extra_context=None):
    """
    Display the main admin index page, which lists all of the installed
    apps that have been registered in this site.
    """
    app_list = self.get_app_list(request)
    context = {
        **self.each_context(request),
        'title': self.index_title,
        'app_list': app_list,
        **(extra_context or {}),
    }

    request.current_app = self.name

    return TemplateResponse(request, self.index_template or
        'admin/index.html', context)
```

The method `get_app_list`, set the order of the models.:

```python
def get_app_list(self, request):
    """
    Return a sorted list of all the installed apps that have been
    registered in this site.
    """
```

```
    app_dict = self._build_app_dict(request)

    # Sort the apps alphabetically.
    app_list = sorted(app_dict.values(), key=lambda x:
x['name'].lower())

    # Sort the models alphabetically within each app.
    for app in app_list:
        app['models'].sort(key=lambda x: x['name'])

    return app_list
```

So to set the order we override `get_app_list` as:

```
class EventAdminSite(AdminSite):
    def get_app_list(self, request):
        """
        Return a sorted list of all the installed apps that
have been
        registered in this site.
        """
        ordering = {
            "Event heros": 1,
            "Event villains": 2,
            "Epics": 3,
            "Events": 4
        }
        app_dict = self._build_app_dict(request)
        # a.sort(key=lambda x: b.index(x[0]))
        # Sort the apps alphabetically.
        app_list = sorted(app_dict.values(), key=lambda x:
x['name'].lower())

        # Sort the models alphabetically within each app.
        for app in app_list:
            app['models'].sort(key=lambda x:
ordering[x['name']])

        return app_list
```

The code `app['models'].sort(key=lambda x: ordering[x['name']])`
sets the fixed ordering. Your app now looks like this.

Welcome to UMSRA Researcher Events Portal

EVENTS		
Event heros	✚ Add	✏ Change
Event villains	✚ Add	✏ Change
Epics	✚ Add	✏ Change
Events	✚ Add	✏ Change

Models used in this book

App entities

The models are:

```python
class Category(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        verbose_name_plural = "Categories"

    def __str__(self):
        return self.name


class Origin(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name


class Entity(models.Model):
    GENDER_MALE = "Male"
    GENDER_FEMALE = "Female"
    GENDER_OTHERS = "Others/Unknown"

    name = models.CharField(max_length=100)
    alternative_name = models.CharField(
        max_length=100, null=True, blank=True
    )


    category = models.ForeignKey(Category,
on_delete=models.CASCADE)
    origin = models.ForeignKey(Origin,
on_delete=models.CASCADE)
    gender = models.CharField(
        max_length=100,
        choices=(
            (GENDER_MALE, GENDER_MALE),
            (GENDER_FEMALE, GENDER_FEMALE),
            (GENDER_OTHERS, GENDER_OTHERS),
        )
    )
```

```python
    description = models.TextField()

    def __str__(self):
        return self.name

    class Meta:
        abstract = True


class Hero(Entity):

    class Meta:
        verbose_name_plural = "Heroes"

    is_immortal = models.BooleanField(default=True)

    benevolence_factor = models.PositiveSmallIntegerField(
        help_text="How benevolent this hero is?"
    )
    arbitrariness_factor = models.PositiveSmallIntegerField(
        help_text="How arbitrary this hero is?"
    )
    # relationships
    father = models.ForeignKey(
        "self", related_name="+", null=True, blank=True,
on_delete=models.SET_NULL
    )
    mother = models.ForeignKey(
        "self", related_name="+", null=True, blank=True,
on_delete=models.SET_NULL
    )
    spouse = models.ForeignKey(
        "self", related_name="+", null=True, blank=True,
on_delete=models.SET_NULL
    )


class Villain(Entity):
    is_immortal = models.BooleanField(default=False)

    malevolence_factor = models.PositiveSmallIntegerField(
        help_text="How malevolent this villain is?"
    )
    power_factor = models.PositiveSmallIntegerField(
        help_text="How powerful this villain is?"
    )
```

```python
    is_unique = models.BooleanField(default=True)
    count = models.PositiveSmallIntegerField(default=1)
```

App events

The models are:

```python
class Epic(models.Model):
    name = models.CharField(max_length=255)
    participating_heroes = models.ManyToManyField(Hero)
    participating_villains = models.ManyToManyField(Villain)


class Event(models.Model):
    epic = models.ForeignKey(Epic, on_delete=models.CASCADE)
    details = models.TextField()
    years_ago = models.PositiveIntegerField()


class EventHero(models.Model):
    event = models.ForeignKey(Event,
on_delete=models.CASCADE)
    hero = models.ForeignKey(Hero, on_delete=models.CASCADE)
    is_primary = models.BooleanField()


class EventVillain(models.Model):
    event = models.ForeignKey(Event,
on_delete=models.CASCADE)
    hero = models.ForeignKey(Villain,
on_delete=models.CASCADE)
    is_primary = models.BooleanField()
```

Index