

# Django Tutorial Part 4: Django admin site

Created by 이윤준 ([yoonyoon.lee@gmail.com](mailto:yoonyoon.lee@gmail.com))

May, 2019

We've created models for the LocalLibrary website, we'll use the Django Admin site to add some "real" book data.

We'll show you

- how to register the models with the admin site,
- how to login and create some data.
- some of the ways you can further improve the presentation of the Admin site.

# Overview

The Django admin *application* can use our models to automatically build a site area that we can use to create, view, update, and delete records.

This can save us a lot of time during development, making it very easy to test your models and get a feel for whether we have the *right* data.

The admin application can also be useful for managing data in production, depending on the type of website.

The Django project recommends it only for internal data management, as the model-centric approach is not necessarily the best possible interface for all users, and exposes a lot of unnecessary detail about the models.

All the configuration required was done automatically when we created the skeleton project.

As a result, all you **must** do to add our models to the admin application is to register them.

At the end, we'll provide a brief demonstration of how we might further configure the admin area to better display our model data.

After registering the models we'll show how to create a new "superuser", login to the site, and create some books, authors, book instances, and genres

# Registering models

Open **admin.py** in the catalog application (**/locallibrary/catalog/admin.py**).

```
from django.contrib import admin  
  
# Register your models here.
```

Register the models. This code simply imports the models and then calls `admin.site.register` to register each of them.

```
from catalog.models import Author, Genre, Book, BookInstance  
  
admin.site.register(Book)  
admin.site.register(Author)  
admin.site.register(Genre)  
admin.site.register(BookInstance)
```

This is the simplest way of registering a model, or models, with the site.

# Creating a superuser

In order to log into the admin site, we need a user account with Staff status enabled.

In order to view and create records we also need this user to have permissions to manage all our objects.

You can create a "superuser" account that has full access to the site and all needed permissions using **manage.py**.

```
python3 manage.py createsuperuser
```



Now restart the development server so we can test the login:

```
python3 manage.py runserver
```

# Logging in and using the site

To login to the site, open the /admin URL (e.g. <http://127.0.0.1:8000/admin>) and enter your new superuser userid and password credentials.

This part of the site displays all our models, grouped by installed application.

You can click on a model name to go to a screen that lists all its associated records, and You can further click on those records to edit them.

You can also directly click the **Add** link next to each model to start creating a record of that type.

The screenshot shows the Django administration interface in a web browser. The browser's address bar displays `127.0.0.1:8000/admin/`. The page title is "Django administration". The main content area is titled "Site administration".

The interface is divided into two main sections: "AUTHENTICATION AND AUTHORIZATION" and "CATALOG".

**AUTHENTICATION AND AUTHORIZATION**

Model	Actions
Groups	<a href="#">+ Add</a> <a href="#">Change</a>
Users	<a href="#">+ Add</a> <a href="#">Change</a>

**CATALOG**

Model	Actions
Authors	<a href="#">+ Add</a> <a href="#">Change</a>
Book instances	<a href="#">+ Add</a> <a href="#">Change</a>
Books	<a href="#">+ Add</a> <a href="#">Change</a>
Genres	<a href="#">+ Add</a> <a href="#">Change</a>

On the right side, there is a sidebar with the following sections:

- Recent actions**
- My actions**  
None available




Click on the **Add** link to the right of *Books* to create a new book.

Django administration WELCOME, UBUNTU. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

[Home](#) » [Catalog](#) » [Books](#) » [Add book](#)

### Add book

**Title:**

**Author:**    


**Summary:**

Enter a brief description of the book

**ISBN:**   
13 Character ISBN number

**Genre:**

Science Fiction  
Fantasy  
Western  
French Poetry



Select a genre for this book Hold down "Control", or "Command" on a Mac, to select more than one.

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

Enter values for the fields. You can create new authors or genres by pressing the + button next to the respective fields. When you're done you can press **SAVE**, **Save and add another**, or **Save and continue editing** to save the record.

When you've finished adding books, click on the Home link in the top bookmark to be taken back to the main admin page.

Then click on the Books link to display the current list of books.

The title of each book is displayed; this is the value returned in the Book model's `__str__()` method that we specified in the last article.

Select book to change

ADD BOOK +

Action:   0 of 3 selected

- ☐ **BOOK**
- ☐ **The Wise Man's Fear**
- ☐ **The Dueling Machine**
- ☐ **The Name of the Wind**

3 books

From this list you can delete books by selecting the checkbox next to the book, selecting the *delete...* action from the *Action* drop-down list, and then pressing the **Go** button. You can also add new books by pressing the **ADD BOOK** button.

You can edit a book by selecting its name in the link. The edit page for a book, is almost identical to the "Add" page. The main differences are the page title (*Change book*) and the addition of **Delete**, **HISTORY** and **VIEW ON SITE** buttons (this last button appears because we defined the `get_absolute_url()` method in our model).



## Change book

[HISTORY](#)[VIEW ON SITE](#) >

Title:

Author:



Summary:

Told in Kvothe's own voice, this is the tale of the magically gifted young man who grows to be the most notorious wizard his world has ever seen.

Enter a brief description of the book

ISBN:

13 Character ISBN number

Genre:

Science Fiction  
Fantasy  
Western  
French Poetry



Select a genre for this book Hold down "Control", or "Command" on a Mac, to select more than one.

[Delete](#)[Save and add another](#)[Save and continue editing](#)[SAVE](#)

Now navigate back to the **Home** page (using the *Home* link the breadcrumb trail) and then view the **Author** and **Genre** lists.

What you won't have is any *Book Instances*, because these are not created from Books (although you can create a **Book** from a **BookInstance** — this is the nature of the ForeignKey field).

Navigate back to the *Home* page and press the associated **Add** button to display the *Add book instance*. Note the large, globally unique Id, which can be used to separately identify a single copy of a book in the library.

## Add book instance

**Id:**

945bb5fe804949808c276aa

Unique ID for this particular book across whole library

**Book:**

-----    

**Imprint:**

**Due back:**

Today | 

Note: You are 10 hours ahead of server time.

**Status:**

Maintenance 

Book availability

Save and add another

Save and continue editing

SAVE

# Advanced configuration

Django does a pretty good job of creating a basic admin site using the information from the registered models:

- Each model has a list of individual records, identified by the string created with the model's `str()` method, and linked to detail views/forms for editing. By default, this view has an action menu at the top that we can use to perform bulk delete operations on records.
- The model detail record forms for editing and adding records contain all the fields in the model, laid out vertically in their declaration order.

We can further customise the interface to make it even easier to use.

- List views:

- Add additional fields/information displayed for each record.
- Add filters to select which records are listed, based on date or some other selection value.
- Add additional options to the actions menu in list views and choose where this menu is displayed on the form.

- Detail views

- Choose which fields to display (or exclude), along with their order, grouping, whether they are editable, the widget used, orientation etc.
- Add related fields to a record to allow inline editing (e.g. add the ability to add and edit book records while you're creating their author record).

We're going to look at a few changes that will improve the interface for our LocalLibrary, including adding more information to `Book` and `Author` model lists, and improving the layout of their edit views. We won't change the `Language` and `Genre` model presentation.

## Register a ModelAdmin class

To change how a model is displayed in the admin interface we define a ModelAdmin class (which describes the layout) and register it with the model.

Let's start with the `Author` model. Open **admin.py** in the catalog application (**/locallibrary/catalog/admin.py**). Comment out your original registration (prefix it with a `#`) for the `Author` model:

```
# admin.site.register(Author)
```

Add a new `AuthorAdmin` and registration.

```
# Define the admin class  
class AuthorAdmin(admin.ModelAdmin):  
    pass  
  
# Register the admin class with the associated model  
admin.site.register(Author, AuthorAdmin)
```

We'll add `ModelAdmin` classes for `Book`, and `BookInstance`. We again need to comment out the original registrations:

```
# admin.site.register(Book)  
# admin.site.register(BookInstance)
```



To create and register the new models, we'll instead use the `@register` decorator to register the models:

```
# Register the Admin classes for Book using the decorator
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    pass

# Register the Admin classes for BookInstance using the decorator
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    pass
```

Currently all of our admin classes are empty (see `pass`) so the admin behaviour will be unchanged!

## Configure list views

The *LocalLibrary* currently lists all authors using the object name generated from the model `__str__()` method. This is fine when there are a few authors, but once many we may end up having duplicates. To differentiate them, or just because we want to show more interesting information about each author, we can use `list_display` to add additional fields to the view.

Replace our `AuthorAdmin` class. The field names to be displayed in the list are declared in a *tuple* in the required order.

```
class AuthorAdmin(admin.ModelAdmin):  
    list_display = ('last_name', 'first_name', 'date_of_birth', 'date_of_death')
```

Now navigate to the author list. The fields above should now be displayed,

Home › Catalog › Authors

Select author to change

ADD AUTHOR +

Action:   0 of 2 selected

<input type="checkbox"/>	LAST NAME	FIRST NAME	DATE OF BIRTH	DIED
<input type="checkbox"/>	Bova	Ben	Nov. 8, 2032	-
<input type="checkbox"/>	Rothfuss	Patrick	June 6, 2073	-

2 authors

For our Book model we'll additionally display the `author` and `genre`. The `author` is a `ForeignKey` field (one-to-many) relationship, and so will be represented by the `__str__()` value for the associated record. Replace the `BookAdmin` class with the version below.

```
class BookAdmin(admin.ModelAdmin):  
    list_display = ('title', 'author', 'display_genre')
```

Unfortunately we can't directly specify the genre field in `list_display` because it is a `ManyToManyField`. Instead we'll define a `display_genre` function to get the information as a string.

Add the code into our `Book` model (**`models.py`**). This creates a string from the first three values of the `genre` field (if they exist) and creates a `short_description` that can be used in the admin site for this method.

```
def display_genre(self):  
    """Create a string for the Genre. This is  
        required to display genre in Admin."""  
    return ', '.join(genre.name  
        for genre in self.genre.all()[:3])  
  
display_genre.short_description = 'Genre'
```

After saving the model and updated admin, open your website and go to the Books list page;

Home › Catalog › Books

Select book to change ADD BOOK +

Action:   0 of 3 selected

<input type="checkbox"/>	TITLE	AUTHOR	GENRE
<input type="checkbox"/>	The Dueling Machine	Bova, Ben	Science Fiction
<input type="checkbox"/>	The Wise Man's Fear (The Kingkiller Chronicle, #2)	Rothfuss, Patrick	Fantasy
<input type="checkbox"/>	The Name of the Wind (The Kingkiller Chronicle, #1)	Rothfuss, Patrick	Fantasy

3 books

The **Genre** model (and the **Language** model, if we defined one) both have a single field, so there is no point creating an additional model for them to display additional fields.

## Add list filters

Once we've got a lot of items in a list, it can be useful to be able to filter which items are displayed. This is done by listing fields in the `list_filter` attribute. Replace our current `BookInstanceAdmin` class with the code fragment below.

```
class BookInstanceAdmin(admin.ModelAdmin):  
    list_filter = ('status', 'due_back')
```

The list view will now include a filter box to the right. Note how we can choose dates and status to filter the values:

## Select book instance to change

ADD BOOK INSTANCE +

Action:   0 of 2 selected

<input type="checkbox"/>	BOOK	STATUS	DUE BACK	ID
<input type="checkbox"/>	The Wise Man's Fear (The Kingkiller Chronicle, #2)	Maintenance	Oct. 2, 2016	e5d22dd1-c8f5-4e9aed1-f77240ec
<input type="checkbox"/>	The Name of the Wind (The Kingkiller Chronicle, #1)	Maintenance	Oct. 12, 2016	99841f53-c533-494b33d-687946f1

2 book instances

## FILTER

## By status

All

Maintenance

On loan

Available

Reserved

## By due back

Any date

Today

Past 7 days

This month

This year

No date

Has date



## Organise detail view layout

By default, the detail views lay out all fields vertically, in their order of declaration in the model. We can change

- the order of declaration, which fields are displayed (or excluded),
- whether sections are used to organise the information,
- whether fields are displayed horizontally or vertically, and
- even what edit widgets are used in the admin forms.

## Controlling which fields are displayed and laid out

Update our `AuthorAdmin` class to add the `fields` line:

```
class AuthorAdmin(admin.ModelAdmin):  
    list_display = ('last_name', 'first_name', 'date_of_birth', 'date_of_death')  
    fields = ['first_name', 'last_name', ('date_of_birth', 'date_of_death')]
```

The `fields` attribute lists just those fields that are to be displayed on the form, in order.

Fields are displayed vertically by default, but will display horizontally if we further group them in a tuple (as shown in the "date" fields above).

In your website go to the author detail view.

Home › Catalog › Authors › Rothfuss, Patrick

Change author

HISTORYVIEW ON SITE >

First name:


Patrick

Last name:


Rothfuss

Date of birth:

2073-06-06

Today | 

Died:

Today | 

Note: You are 11 hours ahead of server time.

Note: You are 11 hours ahead of server time.

## Sectioning the detail view

We can add "sections" to group related model information within the detail form, using the fieldsets attribute.

In the `BookInstance` model we have information related to what the book is (i.e. `name`, `imprint`, and `id`) and when it will be available (`status`, `due_back`). We can add these in different sections by adding the text in bold to our `BookInstanceAdmin` class.

```
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    list_filter = ('status', 'due_back')

    fieldsets = (
        (None, {'fields': ('book', 'imprint', 'id')}),
        ('Availability', {'fields': ('status', 'due_back')}),
    )
```

Each section has its own title (or None, if you don't want a title) and an associated tuple of fields in a dictionary.




# Now navigate to a book instance view in your website.

[Home](#) › [Catalog](#) › [Book instances](#) › 344edc33-37a0-497f-b574-637b6ddcf285 (The Name of the Wind (The Kingkiller Chronicle, #1))

Change book instance

HISTORY

Book:

The Name of the Wind (The Kingkiller Chronicle, #1)   

Imprint:

Published March 27th 2007 by Penguin Grou


Id:

344edc3337a0497fb57463

Unique ID for this particular book across whole library


Availability

Status:

Available 

Book availability

Due back:

Today | 

Note: You are 11 hours ahead of server time.

Delete

Save and add another

Save and continue editing

SAVE

## Inline editing of associated records

Sometimes it can make sense to be able to add associated records at the same time.

We can do this by declaring inlines, of type TabularInline (horizontal layout) or StackedInline (vertical layout, just like the default model layout). We can add the `BookInstance` information inline to our `Book` detail by adding the lines near your `BookAdmin`:

```
class BooksInstanceInline(admin.TabularInline):
    model = BookInstance

@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'display_genre')
    inlines = [BooksInstanceInline]
```

Now navigate to a view for a Book in your website.

**Genre:**

Fantasy  
 Science Fiction

Select a genre for this book Hold down "Control", or "Command" on a Mac, to select more than one.

---

BOOK INSTANCES				
ID	IMPRINT	DUE BACK	STATUS	DELETE?
344edc33-37a0-497f-b074-637b6ddcf285 (The Name of the Wind (The Kingkiller Chronicle, #1))	Published March 27th 2007 by Penguin Group	<input type="text"/> Today	Available ▼	<input type="checkbox"/>
Note: You are 11 hours ahead of server time.				
99841f53-c533-494b-b33d-687946f15bed (The Name of the Wind (The Kingkiller Chronicle, #1))	Published March 27th 2007 by Penguin Group	2016-10-12 Today	Maintenance ▼	<input type="checkbox"/>
Note: You are 11 hours ahead of server time.				
8edafcc6-9d1e-4056-9f20-ce99d1ef157e (The Name of the Wind (The Kingkiller Chronicle, #1))	Published March 27th 2007 by Penguin Group	2016-10-12 Today	On loan ▼	<input type="checkbox"/>
Note: You are 11 hours ahead of server time.				
2d1adecd28fd48d2bd43ae	<input type="text"/>	<input type="text"/> Today	Maintenance ▼	<input type="checkbox"/>
Note: You are 11 hours ahead of server time.				
63f8b2a43e4f497d98b001	<input type="text"/>	<input type="text"/> Today	Maintenance ▼	<input type="checkbox"/>
Note: You are 11 hours ahead of server time.				
2e8cb69ce978416a86dfec	<input type="text"/>	<input type="text"/> Today	Maintenance ▼	<input type="checkbox"/>
Note: You are 11 hours ahead of server time.				

+ Add another Book Instance



All we've done is declare our tabular inline class, which just adds all fields from the inlined model. We can specify all sorts of additional information for the layout, including the fields to display, their order, whether they are read only or not, etc.

## Challenge yourself

It is time for you to try a few things.

1. For the BookInstance list view, add code to display the book, status, due back date, and id (rather than the default `__str__()` text).
2. Add an inline listing of `Book` items to the `Author` detail view using the same approach as we did for `Book/BookInstance`.