Learn Python with Django

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 up the top that you 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.

You can further customise the interface to make it even easier to use. Some of the things you can do are:

- 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 (e.g. Book loan status).
    - 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).

In this section 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 because they only have one field each, so there is no real benefit in doing so!

Register a ModelAdmin class

To change how a model is displayed in the admin interface you 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)
```

Now add a new `AuthorAdmin` and registration as shown below.

```
# Define the admin class
class AuthorAdmin(admin.ModelAdmin):
    pass

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

Now 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)
```

Now to create and register the new models; for the purpose of this demonstration, we'll instead use the `@register` decorator to register the models (this does exactly the same thing as the `admin.site.register()` syntax):

```
# 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! We can now extend these to define our model-specific admin behaviour.

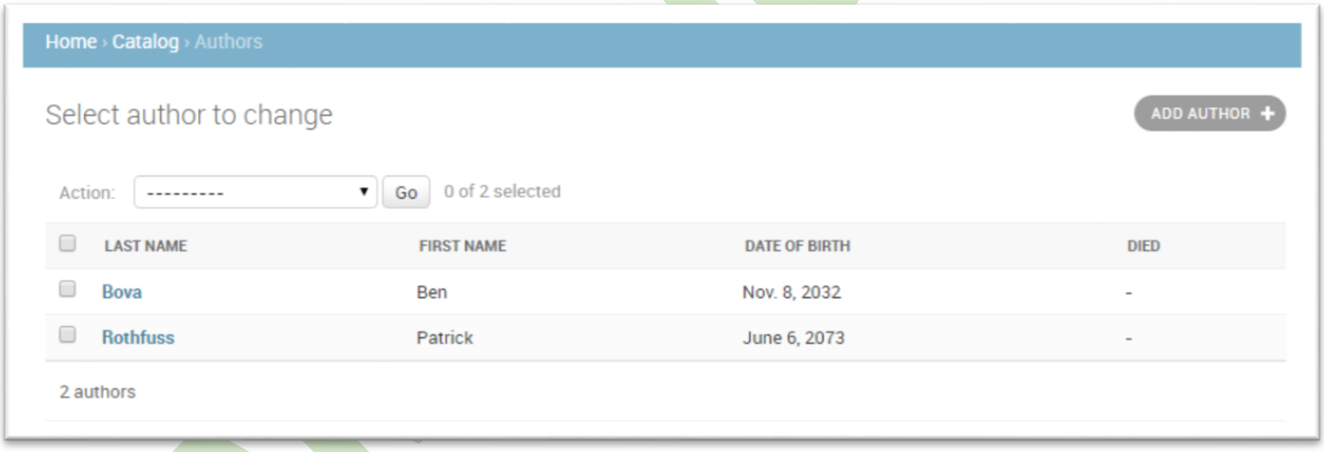
Configure list views

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

Replace your `AuthorAdmin` class with the code below. The field names to be displayed in the list are declared in a tuple in the required order, as shown (these are the same names as specified in your original model).

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

Relaunch the site and navigate to the author list. The fields above should now be displayed, like so:



LAST NAME	FIRST NAME	DATE OF BIRTH	DIED
Bova	Ben	Nov. 8, 2032	-
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-one) 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')
```

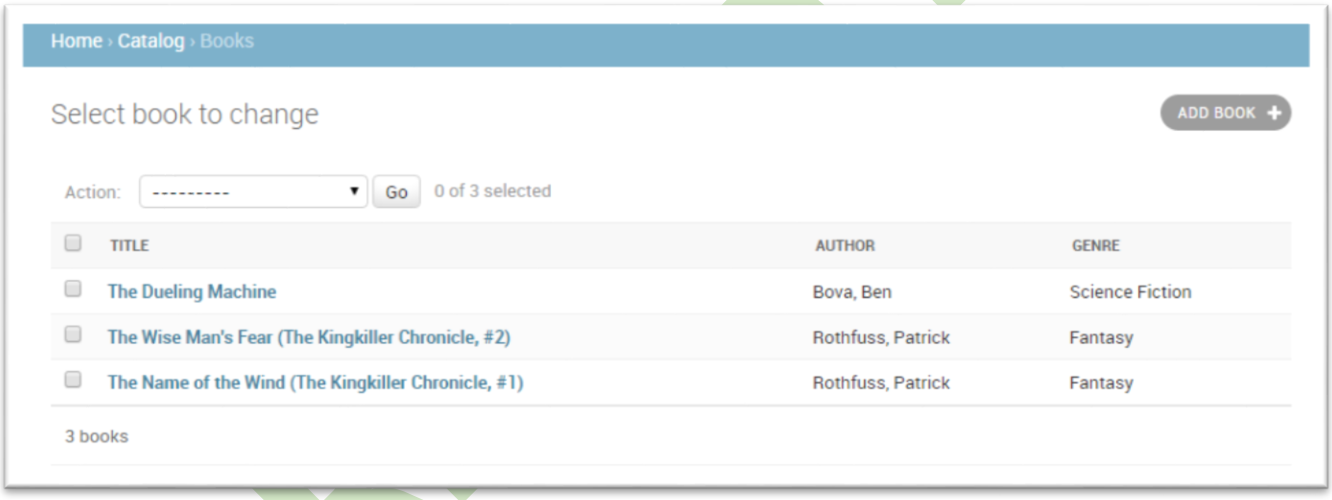
Unfortunately we can't directly specify the genre field in `list_display` because it is a `ManyToManyField` (Django prevents this because there would be a large database access "cost" in doing so). Instead we'll define a `display_genre` function to get the information as a string (this is the function we've called above; we'll define it below).

Note: Getting the `genre` may not be a good idea here, because of the "cost" of the database operation. We're showing you how because calling functions in your models can be very useful for other reasons — for example to add a *Delete* link next to every item in the list.

Add the following code into your `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):
        """
        Creates 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, relaunch the site and go to the *Books* list page; you should see a book list like the one below:



TITLE	AUTHOR	GENRE
The Dueling Machine	Bova, Ben	Science Fiction
The Wise Man's Fear (The Kingkiller Chronicle, #2)	Rothfuss, Patrick	Fantasy
The Name of the Wind (The Kingkiller Chronicle, #1)	Rothfuss, Patrick	Fantasy

3 books

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

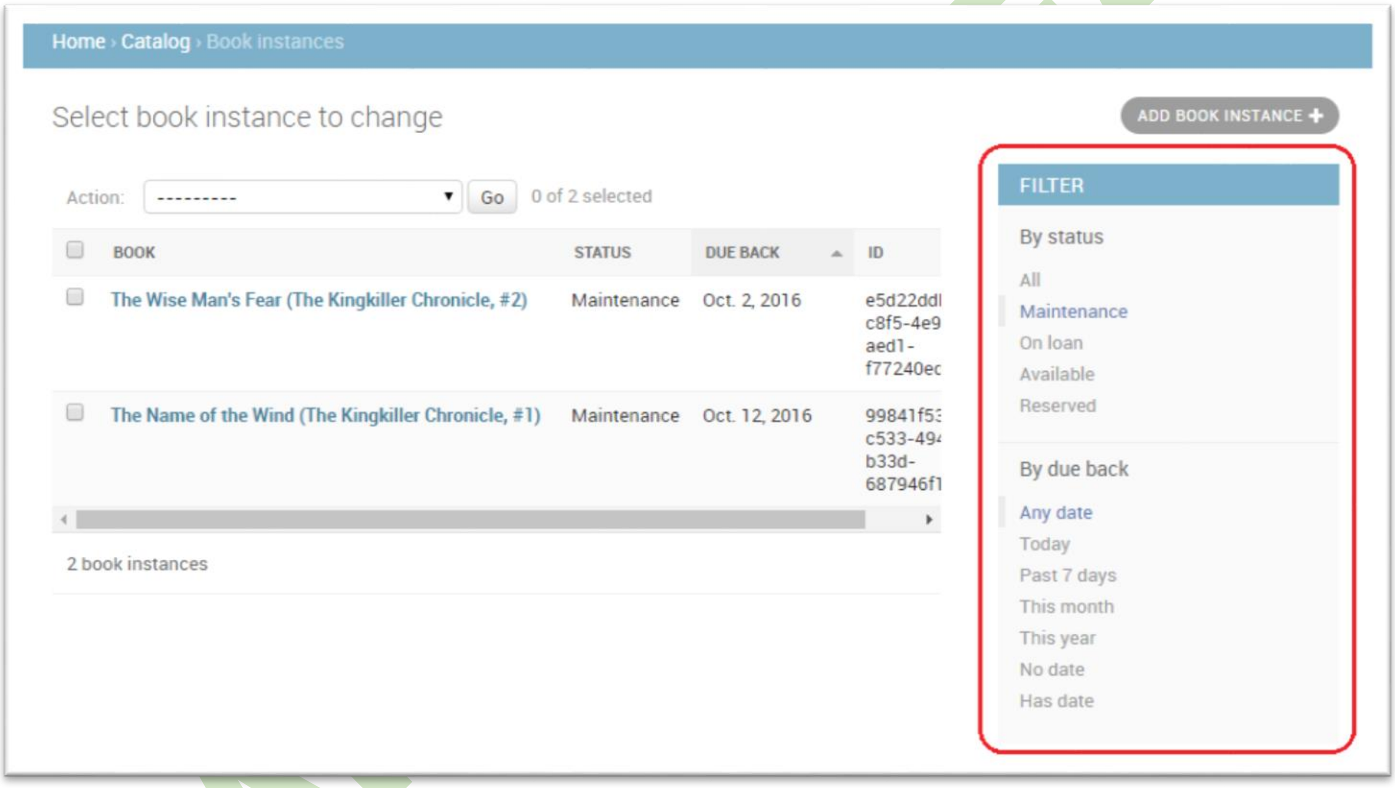
Note: It is worth updating the `BookInstance` model list to show at least the status and the expected return date. We've added that as a challenge at the end of this article!

Add list filters

Once you'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 your 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 you can choose dates and status to filter the values:


Home › Catalog › Book instances

Select book instance to change

	BOOK	STATUS	DUE BACK	ID
	The Wise Man's Fear (The Kingkiller Chronicle, #2)	Maintenance	Oct. 2, 2016	e5d22ddl c8f5-4e9 aed1- f77240ec
	The Name of the Wind (The Kingkiller Chronicle, #1)	Maintenance	Oct. 12, 2016	99841f53 c533-494 b33d- 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. You can change the order of declaration, which fields are displayed (or excluded), whether sections are used to organize the information, whether fields are displayed horizontally or vertically, and even what edit widgets are used in the admin forms.

Note: The *LocalLibrary* models are relatively simple so there isn't a huge need for us to change the layout; we'll make some changes anyway however, just to show you how.

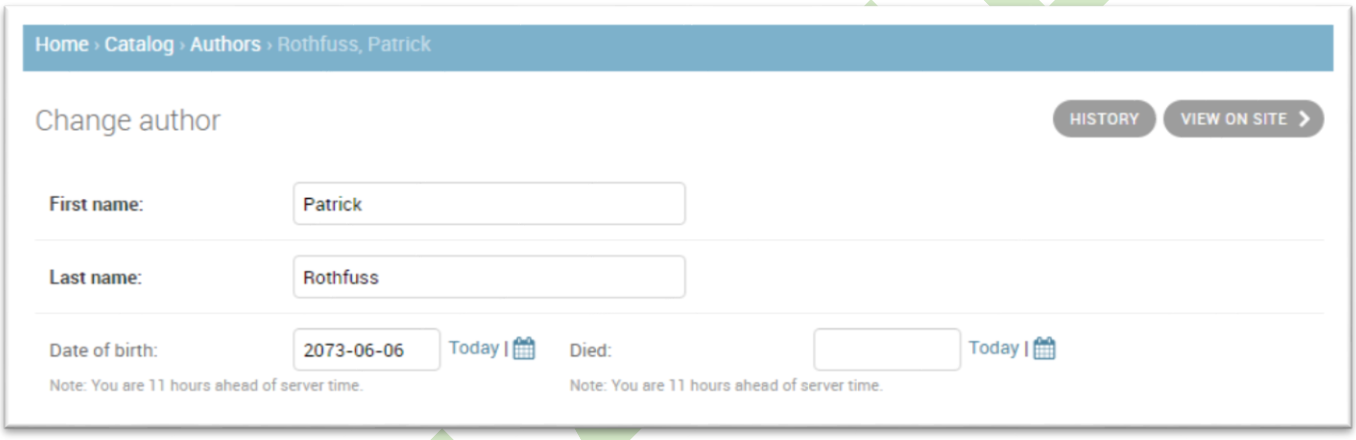
Controlling which fields are displayed and laid out

Update your `AuthorAdmin` class to add the `fields` line, as shown below (in bold):

```
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 you further group them in a tuple (as shown in the "date" fields above).

Restart your application and go to the author detail view — it should now appear as shown below:



First name: Patrick

Last name: Rothfuss

Date of birth: 2073-06-06 Today |

Note: You are 11 hours ahead of server time.

Died: Today |

Note: You are 11 hours ahead of server time.

Note: You can also use the `exclude` attribute to declare a list of attributes to be excluded from the form (all other attributes in the model will be displayed).

Sectioning the detail view

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

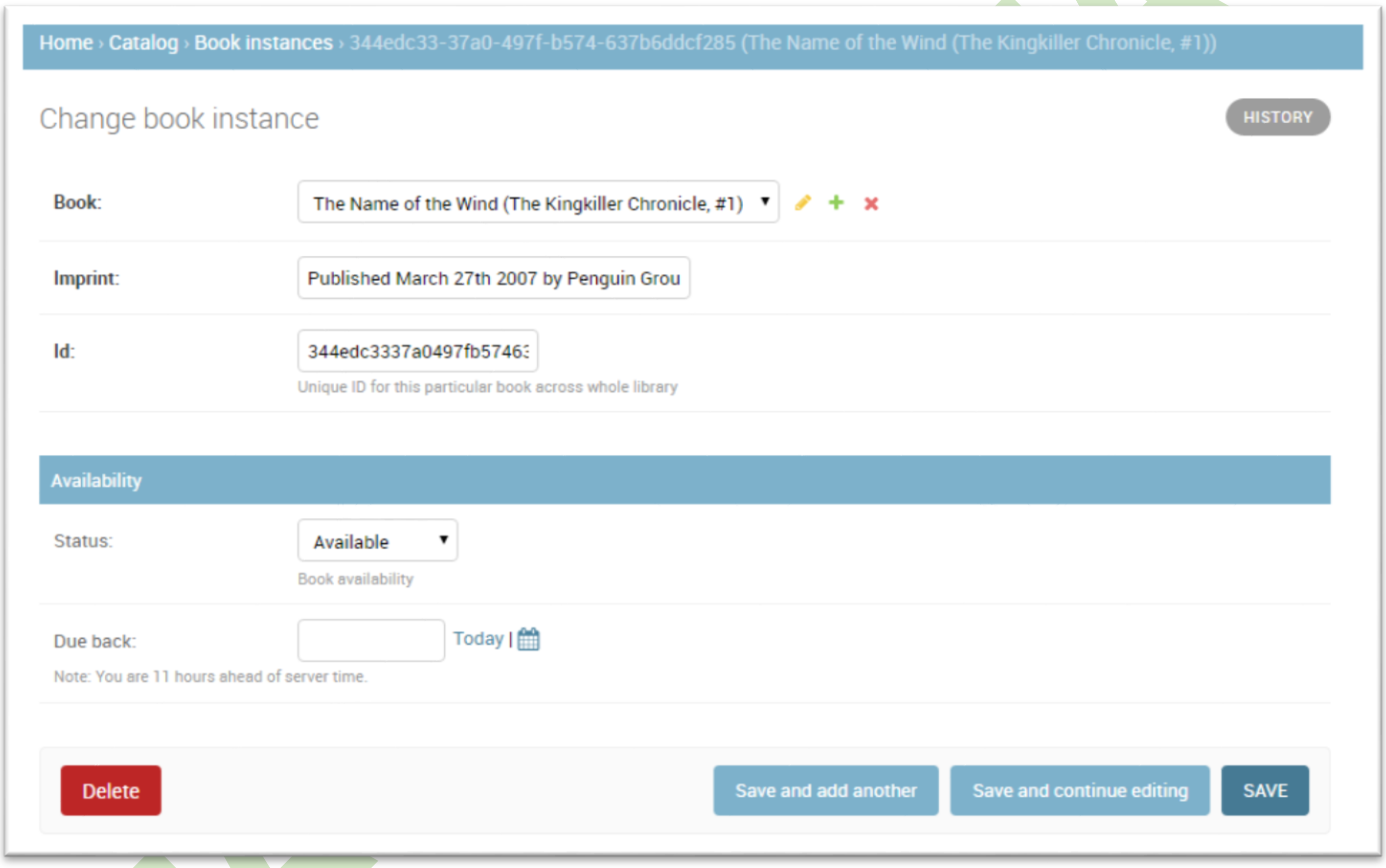
In the `BookInstance` model we have information related to what the book is (i.e. `name`, `imprint`, and `id`) and when it will be available (`status`, `due_back`). We can add these in different sections by adding the text in bold to our `BookInstanceAdmin` class.

```
@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 — the format is complicated to describe, but fairly easy to understand if you look at the code fragment immediately above.

Restart and navigate to a book instance view; the form should appear as shown below:


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

Change book instance

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.

Inline editing of associated records

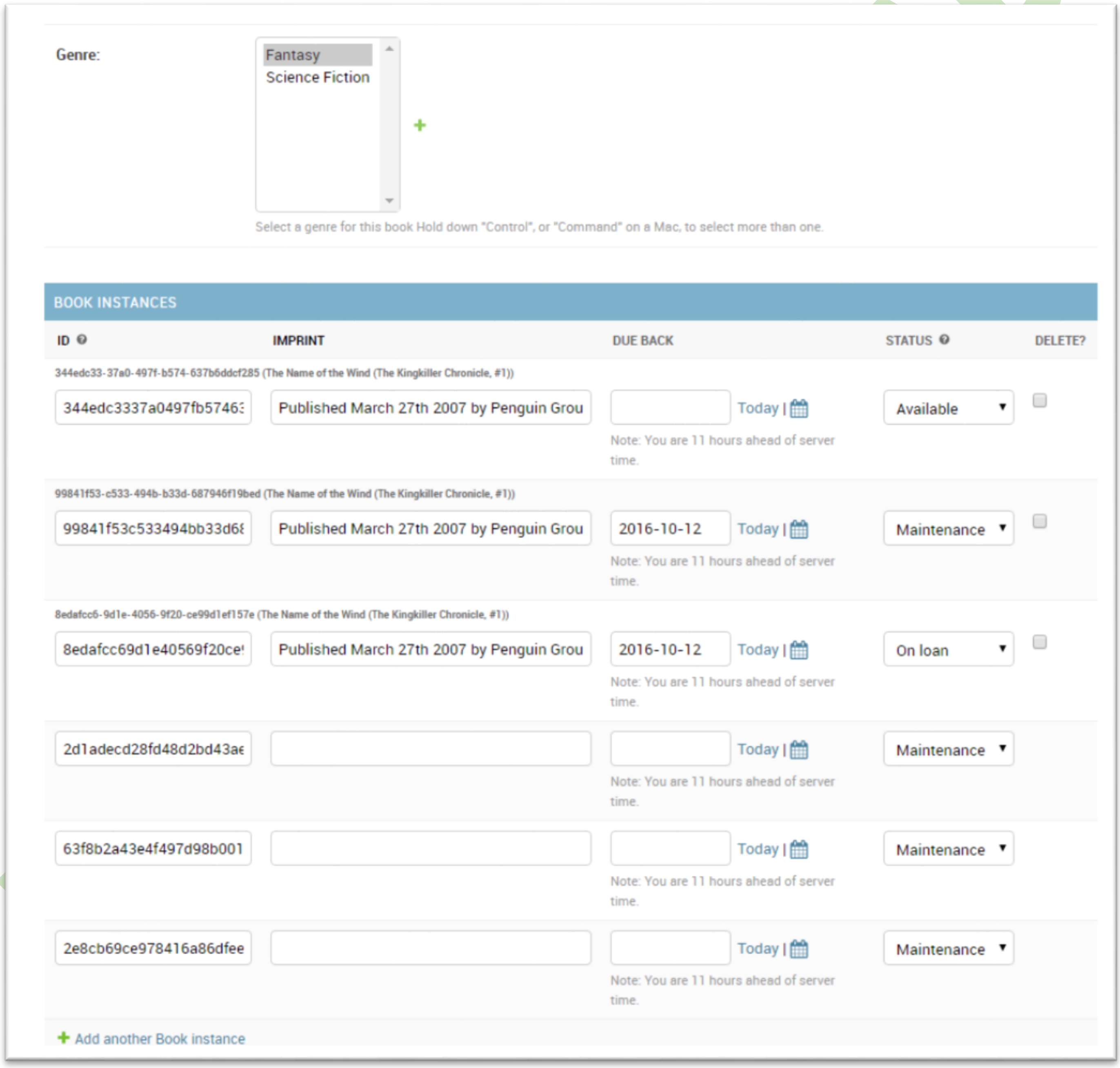
Sometimes it can make sense to be able to add associated records at the same time. For example, it may make sense to have both the book information and information about the specific copies you've got on the same detail page.

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

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

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

Try relaunching your app then looking at the view for a Book — at the bottom you should now see the book instances relating to this book:


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-b574-637b6ddcf285 (The Name of the Wind (The Kingkiller Chronicle, #1))				
344edc3337a0497fb5746:	Published March 27th 2007 by Penguin Grou	Today | Note: You are 11 hours ahead of server time.	Available	☐
99841f53-c533-494b-b33d-687946f19bed (The Name of the Wind (The Kingkiller Chronicle, #1))				
99841f53c533494bb33d6ε	Published March 27th 2007 by Penguin Grou	2016-10-12 Today | Note: You are 11 hours ahead of server time.	Maintenance	☐
8edafcc6-9d1e-4056-9f20-ce99d1ef157e (The Name of the Wind (The Kingkiller Chronicle, #1))				
8edafcc69d1e40569f20ce!	Published March 27th 2007 by Penguin Grou	2016-10-12 Today | Note: You are 11 hours ahead of server time.	On loan	☐
2d1adecd28fd48d2bd43ae		Today | Note: You are 11 hours ahead of server time.	Maintenance	
63f8b2a43e4f497d98b001		Today | Note: You are 11 hours ahead of server time.	Maintenance	
2e8cb69ce978416a86dfee		Today | Note: You are 11 hours ahead of server time.	Maintenance	

+ Add another Book instance

In this case all we've done is declare our tablular inline class, which just adds all fields from the *inlined* model. You 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. (see TabularInline for more information).