Book Library - Code Documentation

version 1.0

Mihalcea Marius-Alex

noiembrie 12, 2025

Contents

Book Library Documentation

Book Library is a comprehensive Django application for managing book libraries.

bookprocess package

admin

Admin customizations for the bookprocess application.

This module contains Django `ModelAdmin` subclasses, admin helper mixins and small admin forms used across the project's admin interface. Its purpose is to centralize common admin behaviour (pagination, file-based population commands, CSV/JSON export utilities) and to provide model-specific admin classes for `Author`, `Book`, `Genre`, `Nationality`, `BookAuthor` and supporting audit/logging integrations.

class bookprocess.admin.**AdminSave**
Bases: **ModelAdmin**
Helper mixin to centralize save behaviour for admin models.
This mixin sets the audit actor when saving from the admin so audit logs correctly attribute the change to the current user.

save_model(request, obj, form, change)
Save the model instance with audit actor set.

Parameters:
- **request** (*django.http.HttpRequest*) – The active HTTP request.
- **obj** (*django.db.models.Model*) – The model instance being saved.
- **form** (*django.forms.Form*) – The admin form used to validate the instance.
- **change** (*bool*) – True if updating an existing object, False for creation.

Return type: None

class bookprocess.admin.**AdminDelete**
Bases: **ModelAdmin**
Helper mixin that ensures audit actor is set when deleting models.

delete_model(request, obj)
Delete a single object with audit actor set.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **obj** (*django.db.models.Model*) – The model instance to delete.

Return type: None

delete_queryset(request, queryset)
Delete a queryset of objects with audit actor set.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **queryset** (*django.db.models.query.QuerySet*) – The queryset of objects to delete.

Return type: None

class bookprocess.admin.**AdminPagination**
Bases: **ModelAdmin**
Mixin to add pagination controls and extra changelist context.

multiple_url: **str** | **None** = *None*
URL for multi-object operations (e.g. add-multiple view).

multiple_button: **str** | **None** = *None*
Label for the multiple action button.

change_list_template = *'admin/admin_pagination_changelist.html'*
Template path used to render the custom changelist when this mixin is applied.

list_per_page = *10*
Current page size for the changelist (can be overridden via GET).

list_per_page_options = *[10, 20, 50]*
Allowed page-size options presented to the user.

__init__ (`model`, `admin_site`)
Initialize the pagination helper.

Parameters:
- **model** (*django.db.models.Model*) – The model class this admin manages.
- **admin_site** (*django.contrib.admin.AdminSite*) – The admin site instance.

Return type: None

populate_url: **str** | **None** = *None*
URL suffix for the model population view

populate_button: **str** | **None** = *None*
Label for the populate button shown in the changelist.

changelist_view (`request`, `extra_context`=None)
Customize the changelist view with pagination and extra template context.
Reads an optional `list_per_page` GET parameter to adjust the page size for the changelist and injects UI labels/URLs used by the custom changelist template.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request object.
- **extra_context** (*dict*) – Additional context to include in the template.

Returns: The response returned by the parent `changelist_view`.

Return type: django.http.HttpResponse

class Media
Bases: **object**
Media assets required by the pagination UI.

js = *('admin/js/admin_paginator_dropdown.js',)*
JavaScript files included on the admin pages that use this mixin.

class bookprocess.admin.**AdminPopulate**
Bases: **ModelAdmin**
Mixin that provides file-upload and command-based population helpers.

populate_form_class: **type**[**Form**] | **None** = *None*
Form class used to accept uploaded files (when provided).

populate_command_class: **type** | **None** = *None*
Management command class responsible for importing data.

is_zip_file: **bool** = *False*
When True uploaded files are treated as ZIP.

admin_file_upload_form = *'admin/admin_file_upload_form.html'*
Template path used to render the populate file upload form.

__init__ (`model`, `admin_site`)
Initialize the populate helper for this admin.

Parameters:
- **model** (*django.db.models.Model*) – The model class this admin is registered for.
- **admin_site** (*django.contrib.admin.AdminSite*) – The admin site instance.

Return type: None

populate_route: **str** | **None** = *None*
URL route suffix for the populate view.

populate_title: **str** | **None** = *None*
Human readable title for the upload page.

get_urls ()
Return admin URLs, prepending the populate route.

Returns: List of URL patterns for the admin, including the populate route.

Return type: list

populate_view (`request`)
Handle requests to the populate endpoint.
Depending on which configuration attributes are set this will either render/accept a file upload form (`populate_form_class` present) or invoke a management command directly (`populate_command_class`).

Parameters: **request** (*django.http.HttpRequest*) – Incoming HTTP request from the admin UI.

Returns: Rendered form, success redirect, or error redirect.

Return type: django.http.HttpResponse or django.http.HttpResponseRedirect

_execute_command (`request`, `**extra_kwargs`)
Execute the configured management command for population.
The configured command class is instantiated, given access to the request as `cmd.request` and then its `handle` method is called with the current user and any `extra_kwargs` (for example the path to an extracted Excel file).

Parameters:
- **request** (*django.http.HttpRequest*) – The current HTTP request.
- ****extra_kwargs** (*dict*) – Extra keyword arguments forwarded to the command's `handle`.

Returns: Redirects back to the model changelist after execution.

Return type: django.http.HttpResponseRedirect

_handle_file_upload (`request`)
Process the populate file upload and run the populate command.
On POST this validates the form, writes the uploaded file to a temporary directory and, if configured, extracts an Excel file from a ZIP archive before calling `_execute_command` with the file path.

Parameters: **request** (*django.http.HttpRequest*) – The incoming request containing POST data and FILES.

Returns: Redirect after processing or the rendered upload form for GET or invalid data.

Return type: django.http.HttpResponse or django.shortcuts.render

_handle_command_execution (`request`)
Execute the configured populate command without a file upload.
This is used when the admin configuration provides only `populate_command_class` and expects the command to perform any required actions itself.

Parameters: **request** (*django.http.HttpRequest*) – The incoming request.

Returns: Redirect back to the model changelist.

Return type: django.http.HttpResponseRedirect

_extract_excel_from_zip(zip_path)
Extract the first Excel file (*.xlsx) from a ZIP archive.

Parameters: **zip_path** (*pathlib.Path*) – Path to the ZIP file on disk.
Returns: Path to the extracted Excel file if found, otherwise None.
Return type: pathlib.Path

class bookprocess.admin.**AdminCount**
Bases: **ModelAdmin**
Small mixin providing a grouped count helper used by statistics admins.

aggregate_count(model, value_field)
Return aggregated counts grouped by a related field.

Parameters:
- **model** (*django.db.models.Model*) – The model to query (for example Book).
- **value_field** (*str*) – The related field to group by (for example 'genre__name').

Returns: QuerySet of dicts containing the grouped value and a count key.
Return type: django.db.models.query.QuerySet

class bookprocess.admin.**AdminWriteCSV**
Bases: **ModelAdmin**
Mixin to export admin data as CSV files.

sections: **list**[**tuple**[**str**, **list**[**str**], **str**, **Callable**]] = []
Sections configuration used by **export_csv()** describing title headers, rows iterable and a row formatting callable.

__init__(model, admin_site)
Initialize the CSV export mixin.

Parameters:
- **model** (*django.db.models.Model*) – The model class that the admin wraps.
- **admin_site** (*django.contrib.admin.AdminSite*) – The admin site instance.

Return type: None

filename: **str** | **None** = *None*
Default filename for the generated CSV when not overridden.

export_csv(sections)
Generate an HttpResponse with the supplied sections written as CSV.

Parameters: **sections** (*Iterable*) – Iterable of tuples (title, headers, rows, row_func) describing each CSV section. row_func is called with each row object to return an iterable of values to write.
Returns: Response object containing the CSV payload and appropriate Content-Disposition header for download.
Return type: django.http.HttpResponse

class bookprocess.admin.**AdminWriteJSON**
Bases: **ModelAdmin**
Mixin to add a JSON export action to admin classes.

actions = ['export_as_json']
Admin actions exposed by this mixin.

__init__(model, admin_site)
Initialize the JSON export mixin.

Parameters:
- **model** (*django.db.models.Model*) – The model class this admin represents.
- **admin_site** (*django.contrib.admin.AdminSite*) – The admin site instance.

Return type: None

filename: **str** | **None** = *None*
Default filename for the exported JSON file.

export_as_json(request, queryset)
Admin action: export the selected objects as a JSON file.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **queryset** (*django.db.models.query.QuerySet*) – The selected objects to export.

Returns: A response containing the JSON representation and an appropriate Content-Disposition header for download.

Return type: django.http.HttpResponse

class bookprocess.admin.**AdminPopulateForm**
Bases: **Form**
Base form used by admin populate views to accept a single file.

allowed_extensions = *[]*
File extensions accepted by the form (e.g. [‘.xlsx’, ‘.zip’]).

__init__(*args, **kwargs)
Initialize the populate form.

Parameters:
- ***args** – Forwarded to `forms.Form.__init__`.
- ****kwargs** – Forwarded to `forms.Form.__init__`.

Return type: None

file_label = *None*
Label for the input; auto-generated when `None`.

file_help_text = *None*
Help text for the input; auto-generated when `None`.

clean()
Validate the uploaded file and ensure allowed extension.

Returns: The cleaned form data.

Return type: dict

class bookprocess.admin.**NationalityAdmin**
Bases: **AdminPagination**, **AdminPopulate**, **AdminSave**, **AdminDelete**
Admin for the **bookprocess.models.Nationality** lookup model.

list_display = *('name', 'code')*
Fields shown in the changelist (`name`, `code`).

search_fields = *('name', 'code')*
Fields used for search.

populate_command_class
Management command used to populate nationality data.
alias of **Command**

class bookprocess.admin.**GenreAdmin**

Bases: **AdminPagination**, **AdminPopulate**, **AdminSave**, **AdminDelete**
Admin for the **bookprocess.models.Genre** lookup model.

list_display = *('name',)*
Fields shown in the changelist (name).

search_fields = *('name',)*
Fields used for search.

populate_command_class
Management command used to populate genre data.
alias of **Command**

class bookprocess.admin.**LogEntryAdmin**
Bases: **AdminWriteJSON**
Admin for audit log entries with helper formatters.

list_display = *('formatted_timestamp', 'actor', 'action', 'object_repr', 'changes_formatted')*
Standard Django admin option for list display and filtering.

list_filter = *('action', 'actor')*
Standard Django admin option for list display and filtering.

search_fields = *('object_repr', 'actor__username', 'changes')*
Standard Django admin option for list display and filtering.

exclude = *['object_pk', 'serialized_data', 'changes_text', 'cid', 'remote_addr', 'remote_port', 'additional_data', 'actor_email']*
Standard Django admin option for list display and filtering.

changes_formatted(obj)
Format the stored change payload resolving related FK names.
This helper transforms the JSON-like changes stored on audit log entries into a readable string. When a changed field is a ForeignKey the helper will attempt to look up the referred object and display its string representation instead of the raw id.

Parameters: **obj** (*auditlog.models.LogEntry*) – The audit log entry instance.
Returns: A human readable description of the changes or "-" when no meaningful data is present.
Return type: str

formatted_timestamp(obj)
Return a human-friendly timestamp for display in list views.

Parameters: **obj** (*auditlog.models.LogEntry*) – The audit log entry instance.
Returns: Formatted timestamp as YYYY-MM-DD HH:MM:SS.
Return type: str

class bookprocess.admin.**StatisticAdmin**
Bases: **AdminCount**, **AdminWriteCSV**
Admin used to display aggregated site statistics.

change_list_template = *'admin/admin_statistics_changelist.html'*
Template used to render the statistics changelist.

get_books_per_genre()
Return counts of books grouped by genre name.

Returns: QuerySet of dicts with keys genre__name and count.
Return type: django.db.models.query.QuerySet

get_authors_per_nationality ()
Return counts of authors grouped by nationality name.

Returns: QuerySet of dicts with keys `nationality__name` and `count`.
Return type: django.db.models.query.QuerySet

get_author_stats ()
Compute per-author statistics for solo and co-authored books.
The function annotates authors with counts of books where they are the sole author and counts where the book has more than one author.

Returns: List of dictionaries in the form `{"author": str, "solo_books": int, "coauthored_books": int}`.
Return type: list

changelist_view (`request`, `extra_context`=None)
Render the custom statistics changelist view.
This view prepares multiple statistic sections (books per genre, authors per nationality and per-author stats) and injects them into the changelist template. Supports CSV export via `?export=csv`.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **extra_context** (*dict*) – Additional context to pass to the changelist template.

Returns: The response for the changelist (or CSV file when requested).
Return type: django.http.HttpResponse

class bookprocess.admin.**AuthorBookInLine**
Bases: **TabularInline**
Inline showing basic book info for an Author's books.

model
The through model (`BookAuthor`) used by this inline.
alias of **BookAuthor**

can_delete = *False*
Whether items can be deleted from the inline.

extra = *0*
Number of extra blank rows shown on the add view.

max_num = *0*
Maximum number of inline forms to display.

show_change_link = *False*
Whether a change link is shown for each inline row.

fields = *('title', 'authors', 'genre', 'isbn', 'cover', 'adapted', 'film_title')*
The fields shown in the inline.

readonly_fields = *('title', 'authors', 'genre', 'isbn', 'cover', 'adapted', 'film_title')*
Fields that are displayed read-only in the inline.

title (`obj`)
Return the title of the related book for display in the inline.

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: The related Book title.
Return type: str

authors (`obj`)

Return a comma-separated list of authors for the related book.

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: Comma-separated author names.
Return type: str

genre (obj)
Return the genre name of the related book.

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: The related Book's genre name.
Return type: str

isbn (obj)
Return the ISBN of the related book.

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: The ISBN string.
Return type: str

cover (obj)
Return an HTML thumbnail linking to the book cover or a placeholder.

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: HTML-safe snippet with an <img> tag or "-" when no cover exists.
Return type: str

adapted (obj)
Return whether the related book is an adaptation (film/etc.).

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: True when the book is marked as adapted, False otherwise.
Return type: bool

film_title (obj)
Return the film title for adapted works or '-' when not present.

Parameters: **obj** (*bookprocess.models.BookAuthor*) – The inline relation instance.
Returns: The film title or `"-"` when not available.
Return type: str

class bookprocess.admin.**AuthorPopulateForm**
Bases: **AdminPopulateForm**
Populate form specialized for author imports (Excel files).

allowed_extensions = *['.xlsx']*
Allowed file extensions for the form.

class bookprocess.admin.**AuthorAdmin**
Bases: **AdminPagination**, **AdminPopulate**, **AdminSave**, **AdminDelete**
Admin for the **bookprocess.models.Author** model.

inlines = *[<class 'bookprocess.admin.AuthorBookInLine'>]*
Inline admin classes shown on the author change view (e.g. book list).

list_display = *('name', 'nationality')*
Columns shown in the changelist.

search_fields = *('first_name', 'last_name')*

Fields used for search in the changelist.

list_filter = *(('nationality', <class 'django_admin_listfilter_dropdown.filters.RelatedDropdownFilter'>),)*
Tuple of list filters applied in the changelist.

populate_form_class
Form class used to upload author import files.
alias of **AuthorPopulateForm**

populate_command_class

Management command used to import author data.
alias of **Command**

get_readonly_fields(request, obj=None)
Return read-only fields for the Author admin.
If the author is the primary author on any book, prevent changing the `nationality` via the admin to preserve data consistency.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **obj** (*Author*) – The instance being viewed/edited (None on add form).

Returns: A list of field names that should be read-only.
Return type: list

name(obj)
Return a human-friendly name for the author instance.
This method is used in the admin list display. It proxies to the model's `name()` helper.

Parameters: **obj** (*Author*) – The author instance.
Returns: The formatted author name.
Return type: str

class bookprocess.admin.**BookForm**
Bases: **ModelForm**
Form used in the Book admin providing validation and media assets.

class **Meta**
Bases: **object**
Meta configuration linking the form to the Book model and including all fields.

model
Model used for this form.
alias of **Book**

fields = *'__all__'*
Fields from model.

clean()
Validate interdependent fields for Book.

Returns: The cleaned form data dictionary.
Return type: dict

class **Media**
Bases: **object**
Include custom JavaScript for dynamic field behavior within the admin form.

js = *('admin/js/admin_book_form_adapted_film_isbn_cover.js', 'admin/js/admin_book_form_authors.js')*
JavaScript files included for dynamic behavior in the admin form.

_meta = *<django.forms.models.ModelFormOptions object>*

class bookprocess.admin.**BookAddMultipleForm**
Bases: **Form**
Form for adding multiple books for the same author dynamically.

author
The ModelChoiceField selecting the author to attach to created books.

class bookprocess.admin.**BookAuthorInline**
Bases: **SortableInlineAdminMixin**, **TabularInline**
Inline admin for the BookAuthor through-model.
Presents a sortable inline to associate authors with a Book and controls the number of empty extra forms shown on add vs edit.

model
The through model (`BookAuthor`) used by this inline.
alias of **BookAuthor**

extra = *1*
Number of extra blank forms to display on the add view.

fields = *('author',)*
Fields shown in the inline form.

autocomplete_fields = *('author',)*
Fields that use autocomplete widgets.

sortable_field_name = *'order'*
Name of the field used by adminsortable2 for ordering.

get_extra(`request`, `obj`=None, `**kwargs`)
Return the number of extra inline forms to display.
When editing an existing Book instance we don't show extra empty inline rows; when adding a new book we provide one extra row to easily add a first author.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **obj** (*None*) – The parent object being edited. None on the add form.
- ****kwargs** (*dict*) – A dictionary of optional keyword arguments passed by Django.

Returns: Number of extra inline forms to display.

Return type: int

class bookprocess.admin.**BookPopulateForm**
Bases: **AdminPopulateForm**
Populate form for importing books; expects a ZIP containing an Excel file.

allowed_extensions = *['.zip']*
Allowed file extensions

class bookprocess.admin.**BookAdmin**
Bases: **SortableAdminBase**, **AdminPagination**, **AdminPopulate**, **AdminDelete**
Admin for the **bookprocess.models.Book** model providing helpers for authors, covers and bulk add.

form
Custom form class used for add/change views (`BookForm`).
alias of **BookForm**

inlines = *[<class 'bookprocess.admin.BookAuthorInline'>]*
Inline classes (`BookAuthorInline`) to manage authors.

list_display = *('title', 'display_authors', 'isbn', 'genre', 'adapted', 'film_title')*
Columns shown in the changelist.

search_fields = *('title', 'isbn', 'bookauthor__author__first_name', 'bookauthor__author__last_name')*
Fields used in changelist search.

list_filter = *(('genre', <class 'django_admin_listfilter_dropdown.filters.RelatedDropdownFilter'>), ('adapted', <class 'django_admin_listfilter_dropdown.filters.DropdownFilter'>))*
Filters displayed in the changelist.

readonly_fields = *('isbn', 'cover_preview')*
Read-only fields in the change view.

fieldsets = *((None, {'fields': ('title', 'genre', 'cover', 'cover_preview', 'adapted', 'film_title', 'isbn')}),)*
Fieldset configuration used in the change form.

multiple_url: **str** | **None** = *'multiple/'*
Route suffix for the add-multiple view.

multiple_button: **str** | **None** = *'Add multiple books for the same author'*
Label for the multiple-add action button.

populate_form_class
Populate form class used for importing books.
alias of **BookPopulateForm**

populate_command_class
Management command used for population imports.
alias of **Command**

is_zip_file: **bool** = *True*
When True the populate view expects a ZIP containing an Excel file.

display_authors(obj)
Return a comma-separated list of the book's authors for display.

Parameters: **obj** (*Book*) – The Book instance.
Returns: Comma-separated author names or an em-dash when none are present.
Return type: str

cover_preview(obj)
Return HTML for a clickable cover preview image.

Parameters: **obj** (*Book*) – The Book instance. May be `None` in some admin contexts.
Returns: An HTML snippet with an <img> wrapped in a link or an empty string when no cover is available.
Return type: str

save_model(request, obj, form, change)
Create a new Book instance with proper audit handling.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **obj** (*Book*) – The Book instance to save.
- **form** (*django.forms.Form*) – The admin form used to validate the instance.
- **change** (*bool*) – True if updating an existing object, False for creation.

Return type: None

save_formset(request, form, formset, change)

Save related formset data and perform post-save bookkeeping.

Parameters:
- **request** (*django.http.HttpRequest*) – The current request.
- **form** (*django.forms.Form*) – The parent Book admin form.
- **formset** (*django.forms.BaseInlineFormSet*) – The inline formset containing BookAuthor relations.
- **change** (*bool*) – True when updating an existing book, False for create.

Returns: This method performs in-place saves and does not return a value.

Return type: None

`_save_authors_from_formset` (`formset`)

Marks objects as `_from_admin` prior to saving so downstream hooks can differentiate admin-created relations.

Parameters: **formset** (*django.forms.BaseInlineFormSet*) – The inline formset of BookAuthor instances.

Return type: None

`_finalize_book_creation` (`user`, `book`)

Generate an ISBN for a newly created Book and emit a CREATE log.

Parameters:
- **user** (*django.contrib.auth.models.User*) – The user performing the creation.
- **book** (*Book*) – The Book instance that was created.

Return type: None

`_update_book_isbn_if_needed` (`user`, `book`)

Regenerate the ISBN only when the primary author's nationality changed.

Parameters:
- **user** (*django.contrib.auth.models.User*) – Acting user (audit actor).
- **book** (*Book*) – Instance to possibly update.

Return type: None

`_snapshot_book_state` (`book`)

Return a snapshot dict of fields used in manual update audit logs.

Returns: Keys: id, title, genre, cover, adapted, film_title, isbn, authors.

Return type: dict

`_diff_snapshots` (`before`, `after`)

Compute a changes dict for audit logging.

Parameters:
- **before** (*dict*) – Pre-change snapshot.
- **after** (*dict*) – Post-change snapshot.

Returns: Mapping field_name -> [old_value, new_value]. Empty when there is no actual change.

Return type: dict

`get_urls` ()

Return admin URL patterns including the multiple-create view.

Returns: List of URL patterns for this admin.

Return type: list

`add_multiple_books_view` (`request`)

Handle the add-multiple-books admin view.

Supports GET to render the form and POST to process submitted multiple-book data.

Parameters: **request** (*django.http.HttpRequest*) – The incoming request.

Returns: The rendered form or a redirect after processing.

Return type: django.http.HttpResponse

_handle_multiple_books_post (`request`)

Validates the form, creates Book instances and associated BookAuthor relations and shows a summary message.

Parameters: **request** (*django.http.HttpRequest*) – The POST request containing form data.

Returns: Redirect back to the changelist on success or the form rendered with errors on invalid input.

Return type: django.http.HttpResponseRedirect or django.http.HttpResponse

_create_books_from_post_data (`request`, `author`)

Loop through POSTed book entries and create Book objects.

Parameters:

- **request** (*django.http.HttpRequest*) – The request containing POST data and FILES.
- **author** (*Author*) – The author to attach as the primary author to each created book.

Returns: Number of successfully created books.

Return type: int

_create_single_book_from_post (`request`, `index`, `author`)

Create a single Book from indexed POST fields.

Parameters:

- **request** (*django.http.HttpRequest*) – The request with POST data and FILES.
- **index** (*int*) – The index used to locate the set of fields for this book.
- **author** (*Author*) – The primary author to associate with the created Book.

Returns: True when the book was created, False when required fields were missing.

Return type: bool

_extract_book_data_from_post (`request`, `index`)

Extract a book's fields from POST using the given index.

Parameters:

- **request** (*django.http.HttpRequest*) – The request containing POST and FILES.
- **index** (*int*) – The index used to construct field names.

Returns: Keyword args suitable for `Book.objects.create(**kwargs)`.

Return type: dict

_show_creation_message (`request`, `books_created`, `author`)

Display a success or warning message after attempting creation.

Parameters:

- **request** (*django.http.HttpRequest*) – The current request for context (used by `messages`).
- **books_created** (*int*) – Number of books successfully created.
- **author** (*Author*) – The author for whom books were created.

Return type: None

_render_multiple_books_form (`request`, `form`=None)

Render the add-multiple-books form template.

Parameters:

- **request** (*django.http.HttpRequest*) – The current request.
- **form** (*BookAddMultipleForm*) – Optional form instance (used to re-render validation errors).

Returns: Rendered HTML response of the form.

Return type: django.http.HttpResponse

apps

Django application configuration for the `bookprocess` app.

This module exposes the **`BookprocessConfig`** AppConfig used to register application metadata with Django.

class bookprocess.apps.**BookprocessConfig**
Bases: **AppConfig**
Application configuration for the `bookprocess` Django app.

default_auto_field = *'django.db.models.BigAutoField'*
The default type for automatically generated primary key fields.

name = *'bookprocess'*
The Python path to the application package. Django uses this to look up the module.

models

Database models for the bookprocess application.

This module defines the core data models used by the application: - `Nationality` – country/nationality lookup - `Genre` – book genre - `Author` – book author - `Book` – main book record - `BookAuthor` – through model to order book authors - `Statistic` – simple JSON-backed statistics container

class bookprocess.models.**Nationality**
Bases: **Model**
A country or nationality used to classify authors.

name
Human-readable name of the nationality (unique).

code
ISO-like short code for the nationality (unique, max length 3).

exception **DoesNotExist**
Bases: **ObjectDoesNotExist**

exception **MultipleObjectsReturned**
Bases: **MultipleObjectsReturned**

author_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.
In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.
Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = *<django.db.models.manager.Manager object>*

class bookprocess.models.**Genre**
Bases: **Model**
A book genre/category.

name
Name of the genre (unique).

exception DoesNotExist
Bases: **ObjectDoesNotExist**

exception MultipleObjectsReturned
Bases: **MultipleObjectsReturned**

book_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.
In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.
Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = *<django.db.models.manager.Manager object>*

class bookprocess.models.**Author**
Bases: **Model**
An author of books in the library.

first_name
Given name of the author.

last_name
Family name of the author.

nationality
Foreign key to the author's nationality.

name ()
Return the author's full name as "First Last".

Returns: Full name composed from `first_name` and `last_name`.
Return type: str

exception DoesNotExist
Bases: **ObjectDoesNotExist**

exception MultipleObjectsReturned
Bases: **MultipleObjectsReturned**

bookauthor_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.
In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.
Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

books
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.
In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.
Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

nationality_id

objects = *<django.db.models.manager.Manager object>*

class bookprocess.models.**Book**
Bases: **Model**
Represents a book record.

title
Title of the book.

authors
Authors related to the book. Uses **BookAuthor** as a through model to preserve order.

genre
Genre foreign key.

cover
Optional cover image stored under `covers/`.

adapted
True if the book was adapted to film.

film_title
Title of the film adaptation when available.

isbn
Unique, non-editable ISBN generated for the book.

generate_isbn ()
Generate a unique ISBN for this book.
This delegates to **bookprocess.utils.generate_unique_isbn_from_book()**.

Returns: A generated ISBN string (13 characters).
Return type: str

ordered_authors ()
Return authors ordered by their `order` value in **BookAuthor**.

Returns: Ordered list of **Author** instances for this book.
Return type: list[Author]

exception DoesNotExist
Bases: **ObjectDoesNotExist**

exception MultipleObjectsReturned

Bases: **MultipleObjectsReturned**

bookauthor_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.
In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.
Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

genre_id

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = *<django.db.models.manager.Manager object>*

class bookprocess.models.**BookAuthor**
Bases: **Model**
Through model that preserves the ordering of authors for a book.

book
ForeignKey to the related book.

author
ForeignKey to the related author.

order
Position of the author in the book's author list.

save (*args, **kwargs)
Auto-assign order based on existing authors for this book.

Parameters:
- ***args** – Positional arguments passed to the parent save method.
- ****kwargs** – Keyword arguments passed to the parent save method.

exception DoesNotExist
Bases: **ObjectDoesNotExist**

exception MultipleObjectsReturned
Bases: **MultipleObjectsReturned**

author_id

book_id

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = *<django.db.models.manager.Manager object>*

class bookprocess.models.**Statistic**
Bases: **Model**
Container for precomputed statistics serialized as JSON.

books_per_genre

Mapping of genre identifiers/names to book counts.

authors_per_nationality
Mapping of nationality identifiers/names to author counts.

authors_stats
Free-form list used to store computed author statistics.

exception DoesNotExist
Bases: **ObjectDoesNotExist**

exception MultipleObjectsReturned
Bases: **MultipleObjectsReturned**

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = *<django.db.models.manager.Manager object>*

views

Views for the public-facing book listing and detail pages.

These simple function-based views are intentionally thin: they prepare the queryset and context used by the templates and delegate presentation to the HTML templates under `books/`.

The views support basic filtering, search and pagination. They are designed for use in the site's frontend and admin preview pages.

`bookprocess.views.`**books_list_view**(`request`)
Render a paginated list of books with optional filters and search.
The view reads several optional GET parameters from `request.GET`:

- `search`: a free-text search applied to title, ISBN and author names (case-insensitive, partial matches).
- `genre`: a numeric genre id to filter the list by genre.
- `adapted`: when set to `'true'` filters books marked as adapted.
- `nationality`: an author nationality id to filter books by the primary author's nationality.
- `per_page`: number of results per page (allowed values: 10, 25, 50). Defaults to 10 for invalid input.
- `page`: page number for pagination.

Parameters: **request** (*django.http.HttpRequest*) – The incoming request object containing GET parameters.
Returns: A rendered template response using `books/books_list.html` and a context containing `page_obj`, `genres`, `nationalities` and the applied filters.
Return type: django.http.HttpResponse

`bookprocess.views.`**book_detail_view**(`request`, `book_id`)
Render a detailed page for a specific book.

Parameters:
- **request** (*django.http.HttpRequest*) – The incoming request object.
- **book_id** (*int*) – Primary key of the requested book.

Returns: A rendered template response using `books/book_detail.html` with `book` in the context.
Return type: django.http.HttpResponse

management

commands

admin_init_author

Management command to bulk-import authors from an Excel file.

This module provides a Django management command intended for admins to import author names and their nationalities from an Excel spreadsheet. It performs lightweight validation, reports progress via the project's `notify` helper and records the acting user in the `auditlog` context when available.

The expected Excel sheet should contain at least two columns: - `author`: a full author name (first name and optional last name(s)). - `nationality`: the exact name of an existing `Nationality` row.

class bookprocess.management.commands.admin_init_author.**Command**

Bases: **BaseCommand**

A management command to import authors from an Excel file.

This command reads an Excel file, validates the presence of required columns and creates or updates `Author` instances.

add_arguments(*parser*)

Register command-line arguments.

Parameters: **parser** (*argparse.ArgumentParser*) – The parser instance provided by Django's management framework. This method should call `add_argument` on the parser to declare accepted CLI parameters.

Return type: None

handle(**args*, ***options*)

Execute the import.

Parameters:

- ***args** – Positional arguments passed by Django.
- ****options** – A mapping containing the parsed CLI options. Expected keys: `excel_file` (str): Path to the Excel file to read. `user` (str, optional): Username to set as the audit actor.

Return type: None

admin_init_book

Management command to bulk-import books (and their authors) from Excel.

This command reads a spreadsheet containing book metadata and creates `Book`, `Author`, and `BookAuthor` records. It performs several validation checks (ISBN format, matching counts of authors and nationalities, genre existence and ISBN -> nationality consistency) and reports progress using the project's `notify` helper.

The expected columns in the Excel file include:

```
title, isbn, adapted, film_title, cover_path, authors,
nationalities, genre
```

Where `authors` and `nationalities` are comma-separated lists of equal length mapping each author to a nationality.

bookprocess.management.commands.admin_init_book.**_to_isbn_string**(*cell*)

Normalize a raw Excel cell to a 13-digit ISBN string candidate.

Parameters: **cell** (*Any*) – Raw value from the Excel cell.

Returns: A cleaned string with separators removed and trailing ".0" stripped. Can be empty when the cell is blank.

Return type: str

class bookprocess.management.commands.admin_init_book.**Command**
Bases: **BaseCommand**
A management command for importing books and related data.
The command will create `Book` instances, save provided cover files (if present), create or fetch `Author` instances, and populate the `BookAuthor` relationship with ordering preserved.

add_arguments (`parser`)
Register command-line arguments.

Parameters: **parser** (*argparse.ArgumentParser*) – The parser instance supplied by Django's management framework. This method should call `add_argument` on the parser to declare accepted CLI parameters.

Return type: None

handle (`*args`, `**options`)
Execute the import.

Parameters:
- ***args** – Positional arguments passed by Django.
- ****options** – A dict-like object with keys expected by this command: `excel_file` (str): path to the Excel file `user` (str, optional): username to set as the audit actor

Return type: None

admin_init_genre

Management command to populate the project's `Genre` table.

This module provides a small Django management command used by administrators to ensure a canonical set of genres exists in the database.

bookprocess.management.commands.admin_init_genre.**GENRES** = *['Adventure', 'Archaeology', 'Art', 'Biography', 'Cooking', 'Crime', 'Drama', 'Fantasy', 'Fiction', 'Folklore', 'Historical Fiction', 'History', 'Horror', 'Mystery', 'Nature', 'Non-Fiction', 'Philosophy', 'Poetry', 'Romance', 'Science Fiction', 'Science', 'Self-Help', 'Technology', 'Thriller', 'Travel']*
Choices available.

class bookprocess.management.commands.admin_init_genre.**Command**
Bases: **BaseCommand**
Populate the `Genre` table with a canonical list of genres.
This command is idempotent and safe to run multiple times. For each entry in the module-level **GENRES** list the command will call **django.db.models.Manager.get_or_create()** to ensure the row exists. Creation and informational messages are emitted via the **bookprocess.utils.notify()** helper which integrates with the project's admin messaging system.

add_arguments (`parser`)
Register command-line arguments.

Parameters: **parser** (*argparse.ArgumentParser*) – The parser instance provided by Django's management framework. This method should call `add_argument` on the parser to declare accepted CLI parameters.

Return type: None

handle (`*args`, `**kwargs`)
Execute the genre population.

Parameters:
- ***args** – Positional args passed by Django.
- ****kwargs** – Keyword args parsed from the CLI. Relevant keys: `user` (str, optional): username to set as the audit actor.

Return type: None

admin_init_nationality

Django management command to populate the Nationality table with all countries and their corresponding ISBN codes.

bookprocess.management.commands.admin_init_nationality.**COUNTRIES** = *[('950', 'Argentina'), ('984', 'Bangladesh'), ('985', 'Belarus'), ('619', 'Bulgaria'), ('976', 'Caribbean Community'), ('956', 'Chile'), ('958', 'Colombia'), ('953', 'Croatia'), ('959', 'Cuba'), ('977', 'Egypt'), ('951', 'Finland'), ('618', 'Greece'), ('962', 'Hong Kong'), ('615', 'Hungary'), ('602', 'Indonesia'), ('600', 'Iran'), ('965', 'Israel'), ('601', 'Kazakhstan'), ('614', 'Lebanon'), ('609', 'Lithuania'), ('967', 'Malaysia'), ('613', 'Mauritius'), ('607', 'Mexico'), ('978', 'Nigeria'), ('608', 'North Macedonia'), ('969', 'Pakistan'), ('612', 'Peru'), ('621', 'Philippines'), ('972', 'Portugal'), ('606', 'Romania'), ('603', 'Saudi Arabia'), ('981', 'Singapore'), ('961', 'Slovenia'), ('982', 'South Pacific'), ('955', 'Sri Lanka'), ('957', 'Taiwan'), ('611', 'Thailand'), ('605', 'Türkiye'), ('617', 'Ukraine'), ('111', 'United Kingdom'), ('000', 'United States'), ('980', 'Venezuela'), ('604', 'Vietnam')]*

Choices available.

class bookprocess.management.commands.admin_init_nationality.**Command**

Bases: **BaseCommand**

Populates the Nationality table with all countries and their codes.

add_arguments (parser)

Register command-line arguments.

Parameters: **parser** (*argparse.ArgumentParser*) – The parser instance provided by Django's management framework. This method should call `add_argument` on the parser to declare accepted CLI parameters.

Return type: None

handle (*args, **kwargs)

Execute the import.

Parameters:

- ***args** – Positional arguments passed by Django.
- ****options** – A mapping containing the parsed CLI options. Expected keys: `excel_file` (str): Path to the Excel file to read. `user` (str, optional): Username to set as the audit actor.

Return type: None

init_roles

Management command to initialize user groups and permissions.

This module provides a Django management command that creates predefined user groups (Administrator, Auditor) and assigns model-level permissions to each. It is designed to be run once during initial setup or whenever group configurations need to be refreshed.

class bookprocess.management.commands.init_roles.**Command**

Bases: **BaseCommand**

Initialize predefined user groups with fine-grained permissions.

This command creates two groups:

- **administrator**: Has CRUD permissions on core models (Author, Book, Genre, Nationality, BookAuthor) and read-only on Statistic, Group, User.
- **auditor**: Has read-only access to audit log entries (LogEntry).

handle (*args, **options)

Execute the role initialization.

This method defines a dictionary of groups and their associated model permissions, then creates or updates each group to ensure the correct permissions are assigned.

Parameters:

- ***args** – Positional arguments passed by Django.
- ****options** – Keyword arguments passed by Django.

Return type: None

services

This module provides wrapper around the `auditlog` app to record audit entries for book-related events.

class bookprocess.services.**AuditLogService**
Bases: **object**
Helper methods to create audit log entries for domain events.

static log_book_creation(user, book)
Log a book creation event to the audit log.
The function serializes the `book` instance and stores the resulting values in the `changes` field of the audit.

Parameters:
- **user** (*str*) – The responsible for the creation.
- **book** (*Book*) – The Book model instance that was created.

Return type: None

static log_book_update(user, book, changes)
Create a single UPDATE audit log entry with the given changes.

Parameters:
- **user** (*django.contrib.auth.models.User*) – Acting user to attribute the update to.
- **book** (*Book*) – The updated instance.
- **changes** (*dict*) – Mapping field -> [old, new] for fields that changed.

Return type: None

utils

Utility helpers for ISBN generation, model serialization and messaging.

bookprocess.utils.**PREFIXES** = *['978', '979']*
Allowed ISBN-13 prefixes.

bookprocess.utils.**isbn13_check_digit**(digits12)
Calculate the ISBN-13 check digit for 12 digits.
The ISBN-13 check digit uses alternating weights 1 and 3 for the first 12 digits. This helper computes the final check digit so the full 13-digit ISBN is valid according to the ISBN-13 specification.

Parameters: **digits12** (*str*) – Exactly 12 numeric characters representing the first 12 digits of an ISBN-13 number.
Returns: A single character string representing the check digit (“0”..”9”).
Return type: str

bookprocess.utils.**_normalize_nat_code**(nat_code)
Normalize an input nationality code to a 3-digit string.
The function keeps only digits from the input and returns a 3-digit string. If input is missing or contains no digits the sentinel `'000'` is returned.

Parameters: **nat_code** (*Optional[str]*) – Input that may contain a nationality code (digits and other characters are tolerated).
Returns: A 3-character digit string (e.g. `'600'` or `'007'`) or `'000'` for missing/invalid input.
Return type: str

bookprocess.utils.**_is_allowed_nat_code**(code3)
Return whether a 3-digit nationality code is allowed for ISBN seed.
The current policy permits the sentinel codes `'000'` and `'111'`, the range `600-621` and the range `950-989`. This function centralizes that rule and can be extended if a canonical list of allowed codes is provided later.

Parameters: **code3** (*str*) – A 3-character string containing digits.

Returns: True when the code is permitted as part of the ISBN seed.

Return type: bool

bookprocess.utils.**_get_book_model** ()

Retrieve the Book model class dynamically.

This helper uses Django's app registry to avoid circular import issues when the Book model needs to be referenced in utility functions that are imported early in the module initialization.

Returns: The **bookprocess.models.Book** model class.

Return type: type

bookprocess.utils.**_get_main_author_code_from_book**(book_instance)

Extract the primary author's nationality code from a Book instance. :type book_instance: :param book_instance: A **bookprocess.models.Book** instance.

Returns: The 3-character nationality code extracted from the chosen author, or None if no author or nationality is available.

Return type: Optional[str]

bookprocess.utils.**generate_unique_isbn_for_nationality**(nat_code, max_tries=5000)

Generate a unique ISBN-13 string using a nationality code seed.

The function normalizes the provided nationality code and, if it is not allowed, falls back to the sentinel '000' code. It then repeatedly generates candidate ISBNs (prefix + nat + pub_id + check) until a non-existing ISBN is found or max_tries is reached.

Parameters:

- **nat_code** (*Optional[str]*) – Input nationality code used as part of the ISBN seed.
- **max_tries** (*int*) – Maximum number of attempts to find a unique ISBN before raising a RuntimeError.

Returns: A unique, 13-character ISBN string.

Return type: str

bookprocess.utils.**generate_unique_isbn_from_book**(book_instance, nat_code_override=None)

Generate a unique ISBN-13 for a Book instance.

The function first respects an explicit nat_code_override if provided; otherwise it attempts to extract the main author's nationality code and delegate to **generate_unique_isbn_for_nationality()**. If neither is available it falls back to using '000' as the seed.

Parameters:

- **book_instance** – The Book instance for which an ISBN should be generated.
- **nat_code_override** (*Optional[str]*) – An explicit nationality code to use instead of extracting from the book's authors.

Returns: A unique ISBN-13 string.

Return type: str

bookprocess.utils.**represent_related**(obj)

Return a human-friendly representation for a related object.

The helper inspects common attribute names (name, title, code, full_name) and returns the first matching attribute's value. If the attribute is callable it will be called. If no known attribute is present the object's str() value is returned.

Parameters: **obj** – Any model instance (or None).

Returns: A human-readable string or None if obj is None.

Return type: Optional[str]

bookprocess.utils.**serialize_model_instance**(instance)

Serialize a Django model instance into a human-readable dict.

The serializer converts foreign keys into a representative string (using **represent_related()**), flattens many-to-many relations into comma-separated lists of names, and converts file fields to their stored filenames. The function uses select_related and prefetch_related to minimize DB queries when re-loading the instance by primary key.

Parameters: **instance** – A saved Django model instance (must have a valid pk).

Returns: Mapping of field name -> serializable value.

Return type: dict

`bookprocess.utils.`**`notify`**(`request`=None, `command`=None, `msg`='', `level`='info')

Display a message via Django messages or a management command.

The helper centralizes how messages are emitted so callers can be ignorant of the current execution context (web request vs. manage command). When `request` is provided the Django messages API is used; when `command` is provided the management command's style helpers are used; otherwise the message prints to stdout.

Parameters:

- **request** (*Optional[django.http.HttpRequest]*) – Optional request object; when provided the message is added to the request using Django's messages framework.
- **command** (*Optional[django.core.management.BaseCommand]*) – Optional management command instance; when provided the message is written to the command's stdout using the styled helper.
- **msg** (*str*) – The message text to display.
- **level** (*str*) – One of `'info'`, `'success'`, `'warning'` or `'error'`.

Quick Navigation

Core Modules

- models - Database models (Book, Author, Genre, etc.)
- views - Web views and API endpoints
- admin - Admin interface customization

Business Logic

- services - Service layer and business logic
- utils - Utility functions and helpers

Management

- management - Custom management commands
- apps - Application configuration

Indices and tables

- **genindex**
- **modindex**

Index

B

C

R

S

T

Python Module Index

b