create a Django admin interface for a hypothetical application managing a list of books in a library

 Defining the models.

 Registering the models with the admin interface.

 Customizing the admin interface for better usability.

**Models.py**

from django.db import models

class Book(models.Model):

title = models.CharField(max\_length=200)

author = models.CharField(max\_length=100)

published\_date = models.DateField()

isbn\_number = models.CharField(max\_length=13, unique=True)

pages = models.IntegerField()

cover = models.ImageField(upload\_to='covers/')

def \_\_str\_\_(self):

return self.title

**admin.py**

from django.contrib import admin

from .models import Book

class BookAdmin(admin.ModelAdmin):

list\_display = ('title', 'author', 'published\_date', 'isbn\_number')

search\_fields = ('title', 'author', 'isbn\_number')

list\_filter = ('published\_date',)

ordering = ('-published\_date',)

Creating and interacting with a database in Django involves several steps, including setting up the database, defining models, making migrations, and performing basic CRUD (Create, Read, Update, Delete) operations.

# settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

Models.py

from django.db import models

class Book(models.Model):

title = models.CharField(max\_length=200)

author = models.CharField(max\_length=100)

published\_date = models.DateField()

isbn\_number = models.CharField(max\_length=13, unique=True)

pages = models.IntegerField()

cover = models.ImageField(upload\_to='covers/')

def \_\_str\_\_(self):

return self.title

CRUD operation

from django.shortcuts import render, get\_object\_or\_404, redirect

from .models import Book

from .forms import BookForm

# Create a new book

def create\_book(request):

if request.method == 'POST':

form = BookForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('book\_list')

else:

form = BookForm()

return render(request, 'create\_book.html', {'form': form})

# Read (list) all books

def book\_list(request):

books = Book.objects.all()

return render(request, 'book\_list.html', {'books': books})

# Read (detail) a single book

def book\_detail(request, pk):

book = get\_object\_or\_404(Book, pk=pk)

return render(request, 'book\_detail.html', {'book': book})

# Update a book

def update\_book(request, pk):

book = get\_object\_or\_404(Book, pk=pk)

if request.method == 'POST':

form = BookForm(request.POST, request.FILES, instance=book)

if form.is\_valid():

form.save()

return redirect('book\_detail', pk=book.pk)

else:

form = BookForm(instance=book)

return render(request, 'update\_book.html', {'form': form})

# Delete a book

def delete\_book(request, pk):

book = get\_object\_or\_404(Book, pk=pk)

if request.method == 'POST':

book.delete()

return redirect('book\_list')

return render(request, 'delete\_book.html', {'book': book})

python manage.py shell

from myapp.models import Book

# Create a new book

new\_book = Book(

title='The Great Gatsby',

author='F. Scott Fitzgerald',

published\_date='1925-04-10',

isbn\_number='9780743273565',

pages=180

)

new\_book.save()

# Read records

books = Book.objects.all()

book = Book.objects.get(id=1)

books\_by\_author = Book.objects.filter(author='F. Scott Fitzgerald')

# Update a record

book = Book.objects.get(id=1)

book.pages = 200

book.save()

# Delete a record

book = Book.objects.get(id=1)

book.delete()

**crud\_example.py**

from django.core.management.base import BaseCommand

from myapp.models import Book

class Command(BaseCommand):

help = 'Perform CRUD operations on Book model'

def handle(self, \*args, \*\*kwargs):

# Create a new book

new\_book = Book(

title='The Great Gatsby',

author='F. Scott Fitzgerald',

published\_date='1925-04-10',

isbn\_number='9780743273565',

pages=180

)

new\_book.save()

self.stdout.write(self.style.SUCCESS('Successfully created a new book'))

# Read records

books = Book.objects.all()

self.stdout.write(self.style.SUCCESS(f'All books: {books}'))

# Update a record

book = Book.objects.get(id=1)

book.pages = 200

book.save()

self.stdout.write(self.style.SUCCESS('Successfully updated the book'))

# Delete a record

book = Book.objects.get(id=1)

book.delete()

self.stdout.write(self.style.SUCCESS('Successfully deleted the book'))

 **Views**: For handling CRUD operations in response to web requests.

 **Django Shell**: For direct interaction and testing.

 **Management Commands**: For automated or repetitive tasks.