Hii Sir,   
  
I am Sudha Rajendran,

I attended the first bootcamp session, and it was a great experience. I learned a lot about your company's workflows, which I found very interesting. Your focus on a generative AI platform for creating multilingual and multi-modal content is impressive. You cover many aspects of content, including translation, creation, transformation, and discovery.

I like how your workflows combine AI and human expertise to ensure high-quality results. It’s not just about using AI; you offer many customizable features, like allowing users to add glossaries and personalize translations. The options for choosing words, automatic page setup, and layout formatting are excellent, supporting more than 25 file formats.

The design is truly impressive! The design features are creative and allow users to translate and create stunning designs, such as posters and social media graphics. Additionally, the writing module is especially interesting because it helps users brainstorm ideas, rewrite content, create drafts, and translate them into different languages.

Additionally, your audio workflows enable speech-to-text and text-to-speech conversions. The "Chatbooks" concept is innovative, letting users turn their content into a chat-friendly format. Users can upload documents, whether they are single files or entire catalogs, and interact with the content in any language. For example, a document in Chinese can be discussed in a different language.

Finally, "Ailaysa" is a collaborative platform for project management, managing linguistic assets, team coordination, assignment tracking, and integration. Overall, the session gave me valuable insights into your company's creative solutions.

You taught the basics of the Django framework and guided us through a simple project called "movie\_portal." This helped us understand how to start a project and run it using Django commands. You also emphasized the importance of table relationships, which are crucial for organizing and managing data effectively.

**My work:**  
  
I am working on a backend project called Library Management System using the Django framework. Following your instructions, I created the necessary table relationships in the program.  
  
I created total eight tables, Here the table names,

* Author
* Publisher
* Category
* Book
* Member
* BorrowBooks
* Reservation
* Penalty
* Inventory
* Review

Three types of Relationships in table:

* **One-to-one**. When each item in each table only appears once. For example, each employee can have only one company car to use.
* **One-to-many**. When one item in one table can have a relationship to multiple items in another table. For example, each purchase order can include multiple products.
* **Many-to-many**. When one or more items in one table can have a relationship to one or more items in another table. For example, each order can have multiple products, and each product can appear on many orders.

**Relationships in my Django models:**

* **Author and Book:** One-to-Many relationship (one author can write multiple books, but each book has only one author).
* **Publisher and Book:** One-to-Many relationship (one publisher can publish multiple books, but each book is published by only one publisher).
* **Category and Book**: Many-to-Many relationship (one book can belong to multiple categories, and each category can include multiple books).
* **Members and BorrowBooks:** One-to-Many relationship (one member can borrow multiple books, but each borrowing record relates to one specific member).
* **Book and BorrowBooks**: One-to-Many relationship (one book can be borrowed multiple times by different members, but each borrowing entry is linked to only one book).
* **Member and Penalty**: One-to-Many relationship (one member can have multiple penalties, but each penalty is linked to only one member).
* **BorrowBooks and Penalty**: One-to-Many relationship (one borrowing entry can have multiple penalties, but each penalty is linked to only one borrowing entry).
* **Member and Review**: One-to-Many relationship (one member can write multiple reviews, but each review is linked to only one member).
* **Book and Review**: One-to-Many relationship (one book can have multiple reviews, but each review is linked to only one book).

**CODING:**  
  
**Model.py/library\_access**

from django.db import models

from django.contrib.auth.models import User

class Author(models.Model):

    """

    To Store Information About Authors

    """

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

    bio = models.TextField(blank=True)

    def \_\_str\_\_(self):

        return self.name

class Publisher(models.Model):

    """

    To Store Information About Publishers

    """

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

    address = models.TextField(blank=True)

    def \_\_str\_\_(self):

        return self.name

class Category(models.Model):

    """

    To Store Information About Categories

    """

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

    def \_\_str\_\_(self):

        return self.name

class Book(models.Model):

    """

    To Store Information About Categories

    """

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

    author = models.ForeignKey(Author, related\_name='books', on\_delete=models.CASCADE) # Many to one Relationship

    publisher = models.ForeignKey(Publisher, on\_delete=models.CASCADE)

    isbn = models.CharField(max\_length=15, unique=True)

    publish\_date = models.DateField()

    categories = models.ManyToManyField(Category) # Many to Many Relationships

    def \_\_str\_\_(self):

        return self.title

class Member(models.Model):

    """

    To Store Information About Member of the Library

    """

    first\_name = models.CharField(max\_length=50)

    last\_name = models.CharField(max\_length=50)

    email = models.EmailField(unique=True)

    phone = models.CharField(max\_length=15)

    join\_date = models.DateField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f'{self.first\_name} {self.last\_name}'

class BorrowBooks(models.Model):

    """

    To Store Information About Borrowed book details

    """

    book = models.ForeignKey(Book, related\_name='borrows', on\_delete=models.CASCADE) # Many to one Relationship

    member = models.ForeignKey(Member, related\_name='borrows', on\_delete=models.CASCADE) # Many to one Relationship

    loan\_date = models.DateField(auto\_now\_add=True)

    return\_date = models.DateField(null=True, blank=True)

    def \_\_str\_\_(self):

        return f'{self.book.title} borrowed by {self.member.first\_name} {self.member.last\_name}'

class Reservation(models.Model):

    """

    To Store Information About Users Who Wants To Reserve Books That Are Currently Unavailable.

    """

    user = models.ForeignKey(Member, related\_name='reservations', on\_delete=models.CASCADE) # Many to one Relationship

    book = models.ForeignKey(Book, related\_name='reservations', on\_delete=models.CASCADE) # Many to one Relationship

    reservation\_date = models.DateField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"{self.user.username} reserved {self.book.title}"

class Penalty(models.Model):

    """

    To Store Information About Managing Penalties For Late Book Returns

    """

    user = models.ForeignKey(Member, related\_name='penalties', on\_delete=models.CASCADE) # Many to one Relationship

    borrowing = models.ForeignKey(BorrowBooks, related\_name='penalties', on\_delete=models.CASCADE) # Many to one Relationship

    amount = models.DecimalField(max\_digits=6, decimal\_places=2)

    paid = models.BooleanField(default=False)

    def \_\_str\_\_(self):

        return f"Penalty for {self.user.username}"

class Inventory(models.Model):

    """

    To Store Information About tracking the number of available copies of each book.

    """

    book = models.ForeignKey(Book, related\_name='inventory', on\_delete=models.CASCADE) # Many to one Relationship

    quantity = models.PositiveIntegerField()

    def \_\_str\_\_(self):

        return f"{self.book.title} - {self.quantity} available"

class Review(models.Model):

    """

    To store Information About User Reviews and Ratings for Books.

    """

    user = models.ForeignKey(Member, related\_name='reviews', on\_delete=models.CASCADE)  # Many to one relationship

    book = models.ForeignKey(Book, related\_name='reviews', on\_delete=models.CASCADE)  # Many to one relationship

    rating = models.PositiveIntegerField()

    comment = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"Review for {self.book.title} by {self.user.username}"

**admins.py/ library\_access**  
  
 from django.contrib import admin

from library\_manage.models import Author, Publisher, Category, Book, Member, BorrowBooks, Penalty, Review

# Register your models here.

admin.site.register(Author)

admin.site.register(Publisher)

admin.site.register(Category)

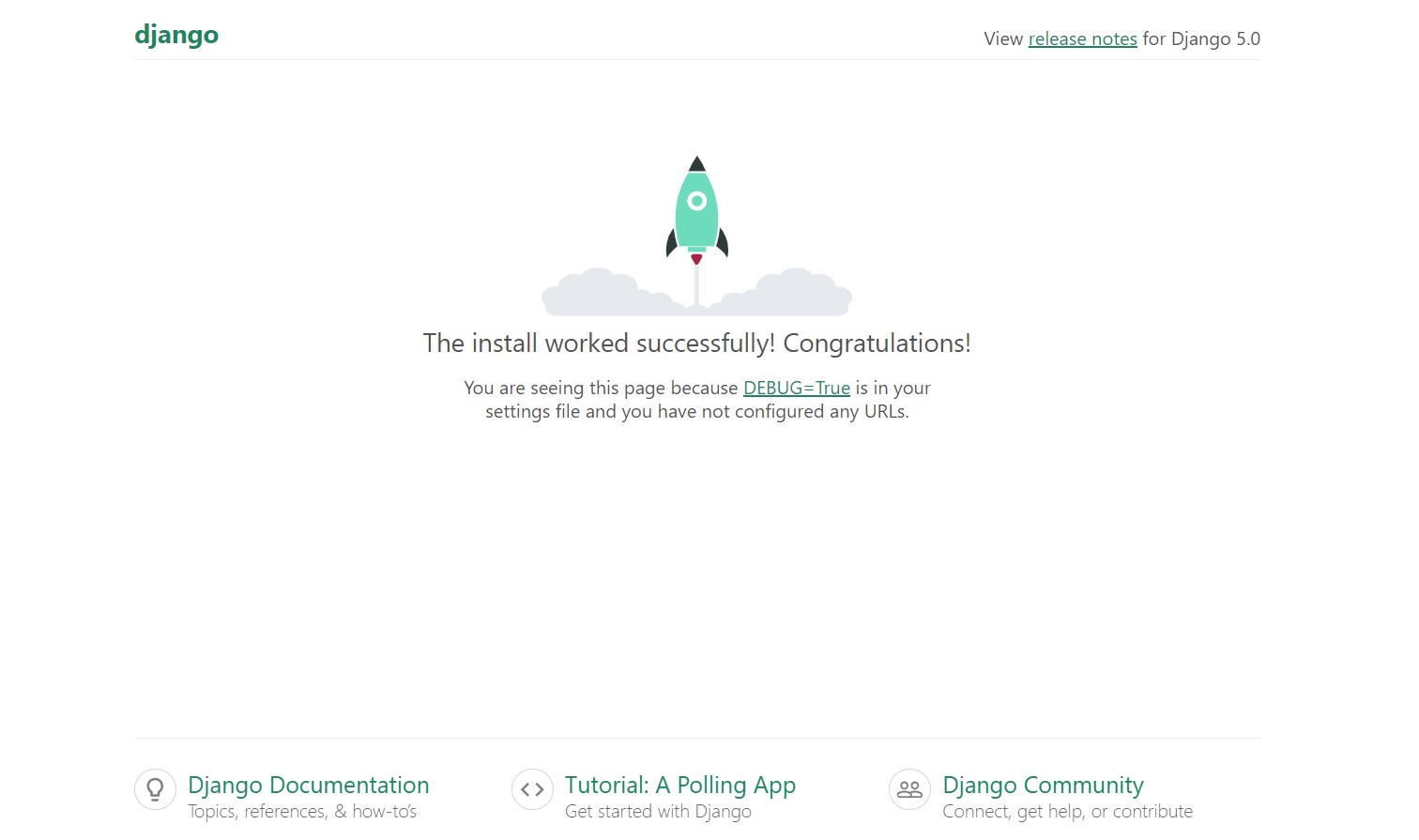
admin.site.register(Book)

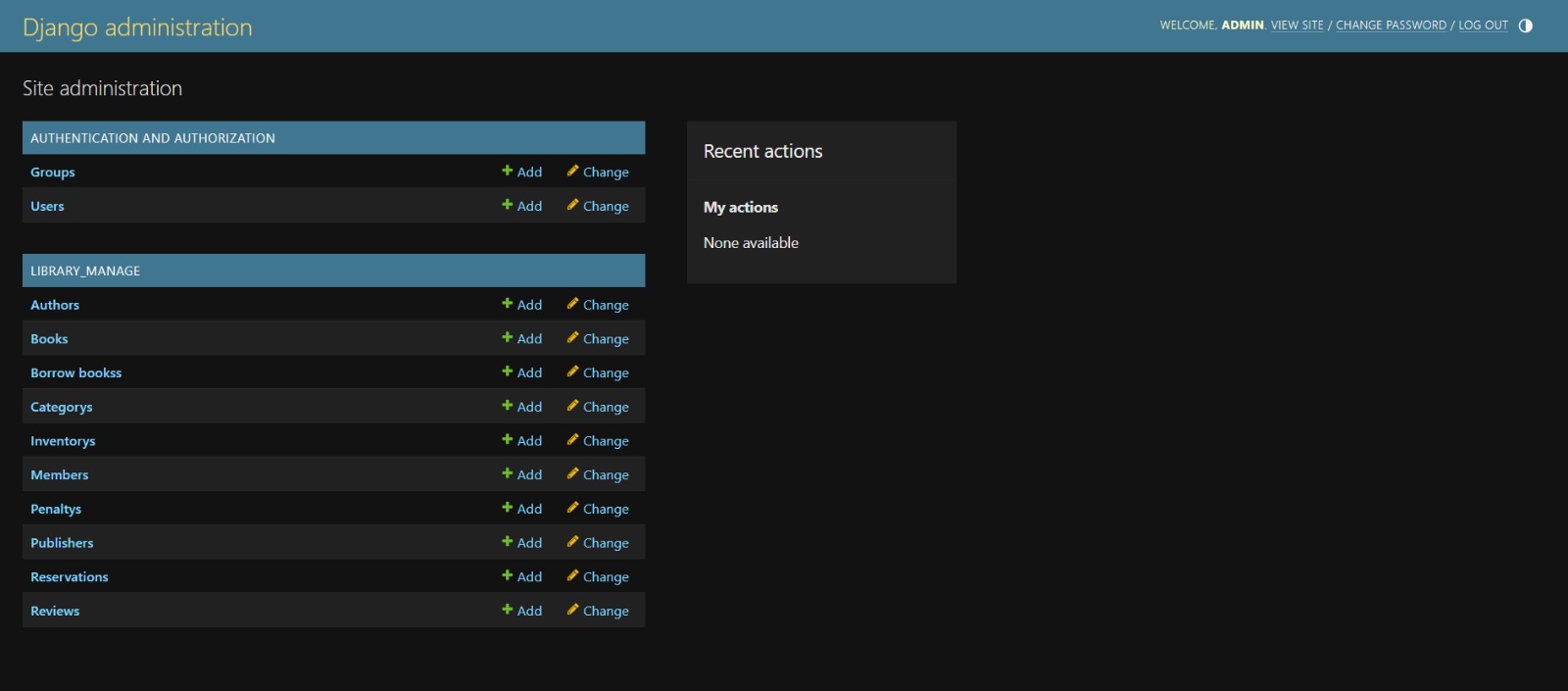
admin.site.register(Member)

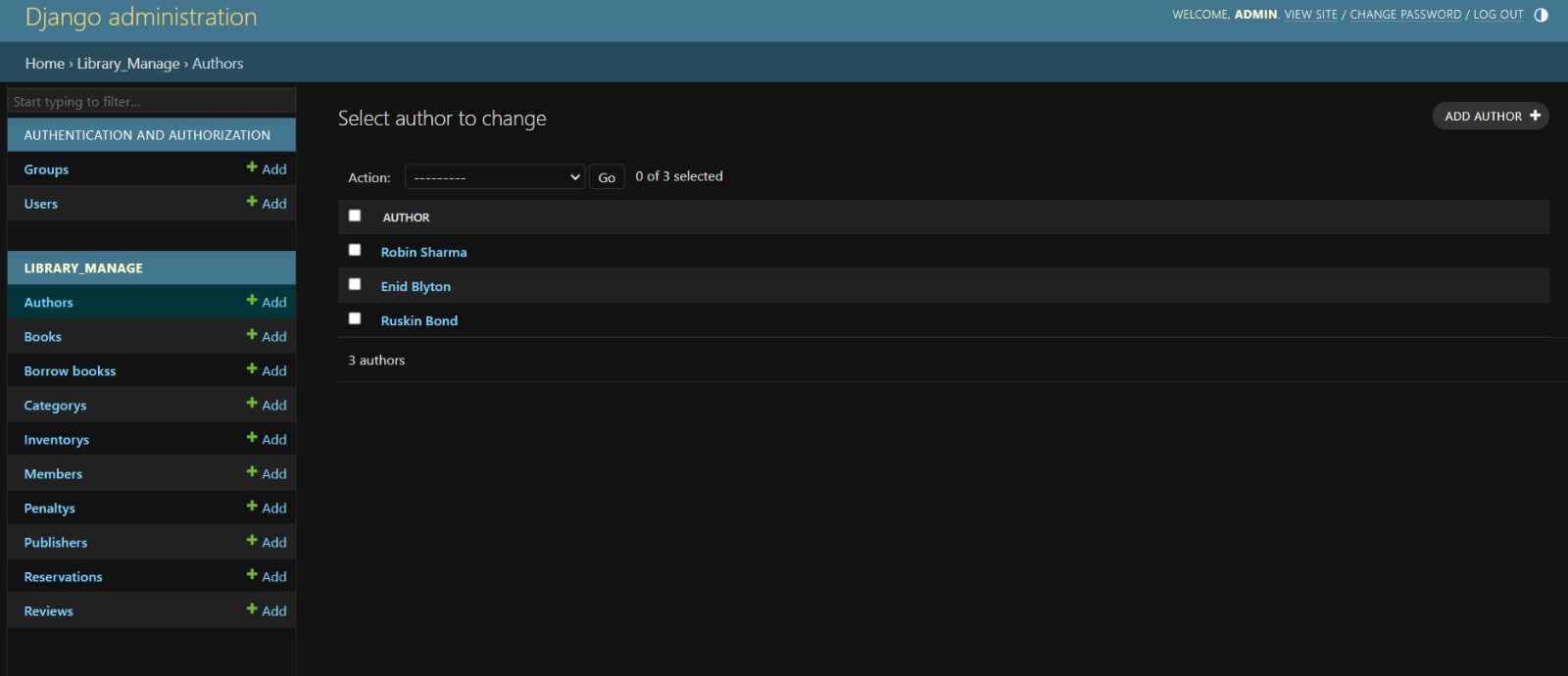
admin.site.register(BorrowBooks)

admin.site.register(Penalty)

admin.site.register(Review)  

**OUTPUT:**





**I**n the Second Session to study the ORM,

Object-Relational Mapping (ORM) lets you use code to work with a database instead of writing SQL queries. It turns database tables into objects in your programming language, making it easier to read and write data. This approach simplifies database tasks and helps keep your code organized.

These methods are get(), first(), last(), all(), filter(), exclude(), values(), order\_by(), count(),create(),delete(), Q objects, \_\_contains,\_\_icontains,\_\_isnull,\_\_endswith, Chaining and  
relations.

**In the Third Session, How to code the serialisers.py and how to serialize and deserialize the data with ORM,  
  
serializers.py/library\_manage**  
  
from rest\_framework import serializers

from .models import Author, Publisher, Category, Book, Member, BorrowBooks, Penalty, Review

class AuthorSerializer(serializers.ModelSerializer):

    class Meta:

        model = Author

        fields = ('id', 'name', 'bio')

    #exclude option also here - Don want to serialize the particular field

    #exclue=['name']

class PublisherSerializer(serializers.ModelSerializer):

    class Meta:

        model = Publisher

        fields = ['id', 'name', 'address']

class CategorySerializer(serializers.ModelSerializer):

    class Meta:

        model = Category

        fields = ['id', 'name']

class BookSerializer(serializers.ModelSerializer):

    author = AuthorSerializer()

    publisher = PublisherSerializer()

    categories = CategorySerializer(many=True)

    class Meta:

        model = Book

        fields = ['id', 'title', 'author', 'publisher', 'isbn', 'publish\_date', 'categories']

class MemberSerializer(serializers.ModelSerializer):

    class Meta:

        model = Member

        fields = ['id', 'first\_name', 'last\_name', 'email', 'phone', 'join\_date']

class BorrowBooksSerializer(serializers.ModelSerializer):

    book = BookSerializer()

    member = MemberSerializer()

    class Meta:

        model = BorrowBooks

        fields = ['id', 'book', 'member', 'loan\_date', 'return\_date']

class PenaltySerializer(serializers.ModelSerializer):

    user = MemberSerializer()

    borrowing = BorrowBooksSerializer()

    class Meta:

        model = Penalty

        fields = ['id', 'user', 'borrowing', 'amount', 'paid']

class ReviewSerializer(serializers.ModelSerializer):

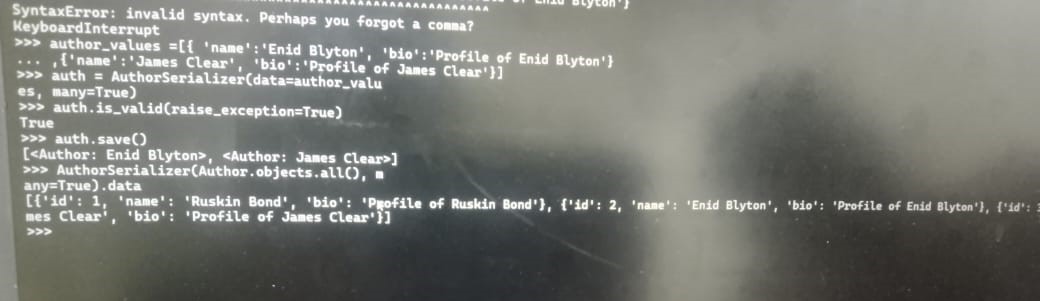
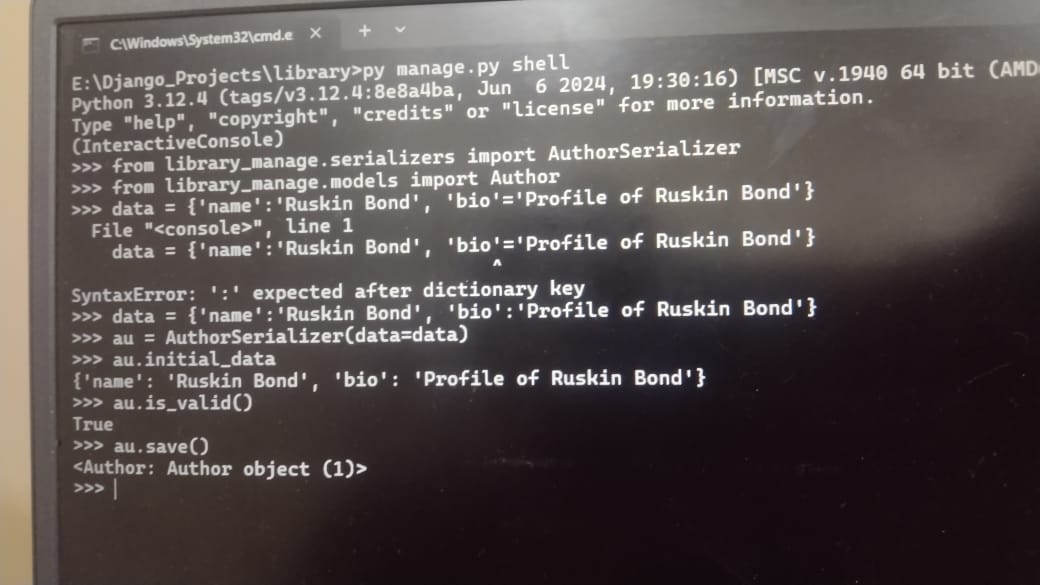
    user = MemberSerializer()

    book = BookSerializer()

    class Meta:

        model = Review

        fields = ['id', 'user', 'book', 'rating', 'comment', 'created\_at']

  
  
  
  
 1

In the Session 4, to the learn the seriazables methods are

* 1. Representation of Serializer
  2. Nested Serizalisers
  3. Source Keyword
  4. to\_internal\_value
  5. run\_Validation

In the Session 5,

How to import PyPi packages in the python file eg: fuzzywuzzy  
  
HTTP METHODS: GET, POST, PUT, DELETE, PATCH  
  
Django give class:  
   
1. APIView – Logic Control  
2. Model Viewset  
3. Generics

How to pass the urls in the Postman API   
  
 **Views.py**

from django.http.response import Http404

from rest\_framework.views import APIView

from rest\_framework.response import Response

from django.http.response import JsonResponse

from .models import Author, Category, Publisher, Book, Member, BorrowBooks, Penalty, Review

from .serializers import AuthorSerializer, CategorySerializer, PublisherSerializer, BookSerializer, MemberSerializer, BorrowBooksSerializer, PenaltySerializer, ReviewSerializer

# Create your views here.

class AuthorView(APIView):

    def post(self, request):

        data = request.data

        serializer = AuthorSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Author Added Successfully",safe=False)

        return JsonResponse("Failed to Add Author", safe=False)

    def get\_author(self, pk):

        try:

            author = Author.objects.get(id=pk)

            return author

        except Author.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_author(pk)

            serializer = AuthorSerializer(data)

        else:

            data = Author.objects.all()

            serializer = AuthorSerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = AuthorSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Author Created Successfully", safe=False)

        return JsonResponse("Failed to Add Author", safe=False)

    def put(self, request, pk=None):

        author\_to\_update = Author.objects.get(id=pk)

        serializer = AuthorSerializer(instance=author\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Author Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Author")

    def delete(self, request, pk=None):

        author\_to\_delete = Author.objects.get(id=pk)

        author\_to\_delete.delete()

        return JsonResponse("Author Deleted Successfully", safe=False)

class CategoryView(APIView):

    def post(self, request):

        data = request.data

        serializer = CategorySerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Category Added Successfully",safe=False)

        return JsonResponse("Failed to Category Author", safe=False)

    def get\_category(self, pk):

        try:

            category = Category.objects.get(id=pk)

            return category

        except Category.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_author(pk)

            serializer = CategorySerializer(data)

        else:

            data = Category.objects.all()

            serializer = CategorySerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = CategorySerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Category Created Successfully", safe=False)

        return JsonResponse("Failed to Add Category", safe=False)

    def put(self, request, pk=None):

        category\_to\_update = Category.objects.get(id=pk)

        serializer = CategorySerializer(instance=category\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Category Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Category")

    def delete(self, request, pk=None):

        category\_to\_delete = Category.objects.get(id=pk)

        category\_to\_delete.delete()

        return JsonResponse("Category Deleted Successfully", safe=False)

class PublisherView(APIView):

    def post(self, request):

        data = request.data

        serializer = PublisherSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Publisher Added Successfully",safe=False)

        return JsonResponse("Failed to Publisher Author", safe=False)

    def get\_publisher(self, pk):

        try:

            publisher = Publisher.objects.get(id=pk)

            return publisher

        except Publisher.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_publisher(pk)

            serializer = PublisherSerializer(data)

        else:

            data = Publisher.objects.all()

            serializer = PublisherSerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = PublisherSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Publisher Created Successfully", safe=False)

        return JsonResponse("Failed to Add Publisher", safe=False)

    def put(self, request, pk=None):

        publisher\_to\_update = Publisher.objects.get(id=pk)

        serializer = PublisherSerializer(instance=publisher\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Publisher Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Publisher")

    def delete(self, request, pk=None):

        publisher\_to\_delete = Publisher.objects.get(id=pk)

        publisher\_to\_delete.delete()

        return JsonResponse("Publisher Deleted Successfully", safe=False)

class BookView(APIView):

    def post(self, request):

        data = request.data

        serializer = BookSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Book Added Successfully",safe=False)

        return JsonResponse("Failed to Add Book", safe=False)

    def get\_book(self, pk):

        try:

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

            return book

        except Book.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_book(pk)

            serializer = BookSerializer(data)

        else:

            data = Book.objects.all()

            serializer = BookSerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = BookSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Book Created Successfully", safe=False)

        return JsonResponse("Failed to Add Book", safe=False)

    def put(self, request, pk=None):

        book\_to\_update = Book.objects.get(id=pk)

        serializer = BookSerializer(instance=book\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Book Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Book")

    def delete(self, request, pk=None):

        book\_to\_delete = Book.objects.get(id=pk)

        book\_to\_delete.delete()

        return JsonResponse("Book Deleted Successfully", safe=False)

class MemberView(APIView):

    def post(self, request):

        data = request.data

        serializer = MemberSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Member Added Successfully",safe=False)

        return JsonResponse("Failed to Member Book", safe=False)

    def get\_member(self, pk):

        try:

            member = Member.objects.get(id=pk)

            return member

        except Member.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_member(pk)

            serializer = MemberSerializer(data)

        else:

            data = Member.objects.all()

            serializer = MemberSerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = MemberSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Member Created Successfully", safe=False)

        return JsonResponse("Failed to Add Member", safe=False)

    def put(self, request, pk=None):

        member\_to\_update = Member.objects.get(id=pk)

        serializer = MemberSerializer(instance=member\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Member Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Member")

    def delete(self, request, pk=None):

        member\_to\_delete = Member.objects.get(id=pk)

        member\_to\_delete.delete()

        return JsonResponse("Member Deleted Successfully", safe=False)

class BorrowBooksView(APIView):

    def post(self, request):

        data = request.data

        serializer = BorrowBooksSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Member Added Successfully",safe=False)

        return JsonResponse("Failed to Member Book", safe=False)

    def get\_BorrowBooks(self, pk):

        try:

            borrow = BorrowBooks.objects.get(id=pk)

            return borrow

        except BorrowBooks.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_BorrowBooks(pk)

            serializer = BorrowBooksSerializer(data)

        else:

            data = BorrowBooks.objects.all()

            serializer = BorrowBooksSerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = BorrowBooksSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Borrow Books Created Successfully", safe=False)

        return JsonResponse("Failed to Add Borrow Books", safe=False)

    def put(self, request, pk=None):

        borrow\_to\_update = BorrowBooks.objects.get(id=pk)

        serializer = BorrowBooksSerializer(instance=borrow\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Borrow Book Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Borrow Book")

    def delete(self, request, pk=None):

        borrow\_to\_delete = BorrowBooks.objects.get(id=pk)

        borrow\_to\_delete.delete()

        return JsonResponse("Borrow Book Deleted Successfully", safe=False)

class PenaltyView(APIView):

    def post(self, request):

        data = request.data

        serializer = PenaltySerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Penalty Added Successfully",safe=False)

        return JsonResponse("Failed to Add Penalty", safe=False)

    def get\_Penalty(self, pk):

        try:

            penalty = Penalty.objects.get(id=pk)

            return penalty

        except Penalty.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_Penalty(pk)

            serializer = PenaltySerializer(data)

        else:

            data = Penalty.objects.all()

            serializer = PenaltySerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = PenaltySerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Penalty Created Successfully", safe=False)

        return JsonResponse("Failed to Add Penalty", safe=False)

    def put(self, request, pk=None):

        penalty\_to\_update = Penalty.objects.get(id=pk)

        serializer = PenaltySerializer(instance=penalty\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Penalty Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Penalty")

    def delete(self, request, pk=None):

        penalty\_to\_delete = Penalty.objects.get(id=pk)

        penalty\_to\_delete.delete()

        return JsonResponse("Penalty Deleted Successfully", safe=False)

class ReviewView(APIView):

    def post(self, request):

        data = request.data

        serializer = ReviewSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Review Added Successfully",safe=False)

        return JsonResponse("Failed to Add Review", safe=False)

    def get\_Penalty(self, pk):

        try:

            review = Review.objects.get(id=pk)

            return review

        except Review.DoesNotExist:

            raise Http404

    def get(self, request, pk=None):

        if pk:

            data = self.get\_Penalty(pk)

            serializer = ReviewSerializer(data)

        else:

            data = Review.objects.all()

            serializer = ReviewSerializer(data, many=True)

        return Response(serializer.data)

    def post(self, request):

        data = request.data

        serializer = ReviewSerializer(data=data)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Review Created Successfully", safe=False)

        return JsonResponse("Failed to Add Review", safe=False)

    def put(self, request, pk=None):

        review\_to\_update = Review.objects.get(id=pk)

        serializer =  ReviewSerializer(instance=review\_to\_update, data=request.data, partial=True)

        if serializer.is\_valid():

            serializer.save()

            return JsonResponse("Review Updated Successfully", safe=False)

        return JsonResponse("Failed to Update Review")

    def delete(self, request, pk=None):

        Review\_to\_delete =  Review.objects.get(id=pk)

        Review\_to\_delete.delete()

        return JsonResponse("Review Deleted Successfully", safe=False)  
  
**urls.py/library\_manage**

from django.urls import path

from .views import AuthorView, CategoryView, PublisherView, BookView, MemberView, BorrowBooksView, PenaltyView, ReviewView

urlpatterns =[

    path("add\_post\_author/", AuthorView.as\_view(), name='add-post-author'),

    path("upd\_del\_author/<int:pk>/", AuthorView.as\_view(), name ='upd-del-author'),

    path("add\_post\_cate/", CategoryView.as\_view(), name='add-post-cate'),

    path("upd\_del\_cate/<int:pk>/", CategoryView.as\_view(), name='add-del-author'),

    path("add\_post\_publisher/", PublisherView.as\_view(), name='add-post-publisher'),

    path("upd\_del\_publisher/<int:pk>/", PublisherView.as\_view(), name='add-del-publisher'),

    path("add\_post\_book/", BookView.as\_view(), name='add-post-book'),

    path("upd\_del\_book/<int:pk>/", BookView.as\_view(), name='add-del-book'),

    path("add\_post\_member/", MemberView.as\_view(), name='add-post-member'),

    path("upd\_del\_member/<int:pk>/", MemberView.as\_view(), name='add-del-member'),

    path("add\_post\_borrow/", BorrowBooksView.as\_view(), name='add-post-borrow'),

    path("upd\_del\_borrow/<int:pk>/", BorrowBooksView.as\_view(), name='add-del-borrow'),

    path("add\_post\_penalty/", PenaltyView.as\_view(), name='add-post-penalty'),

    path("upd\_del\_penalty/<int:pk>/", PenaltyView.as\_view(), name='add-del-penalty'),

    path("add\_post\_review/", ReviewView.as\_view(), name='add-post-review'),

    path("upd\_del\_review/<int:pk>/", ReviewView.as\_view(), name='add-del-review'),

**urls.py/library**

from django.contrib import admin

from django.urls import path, include

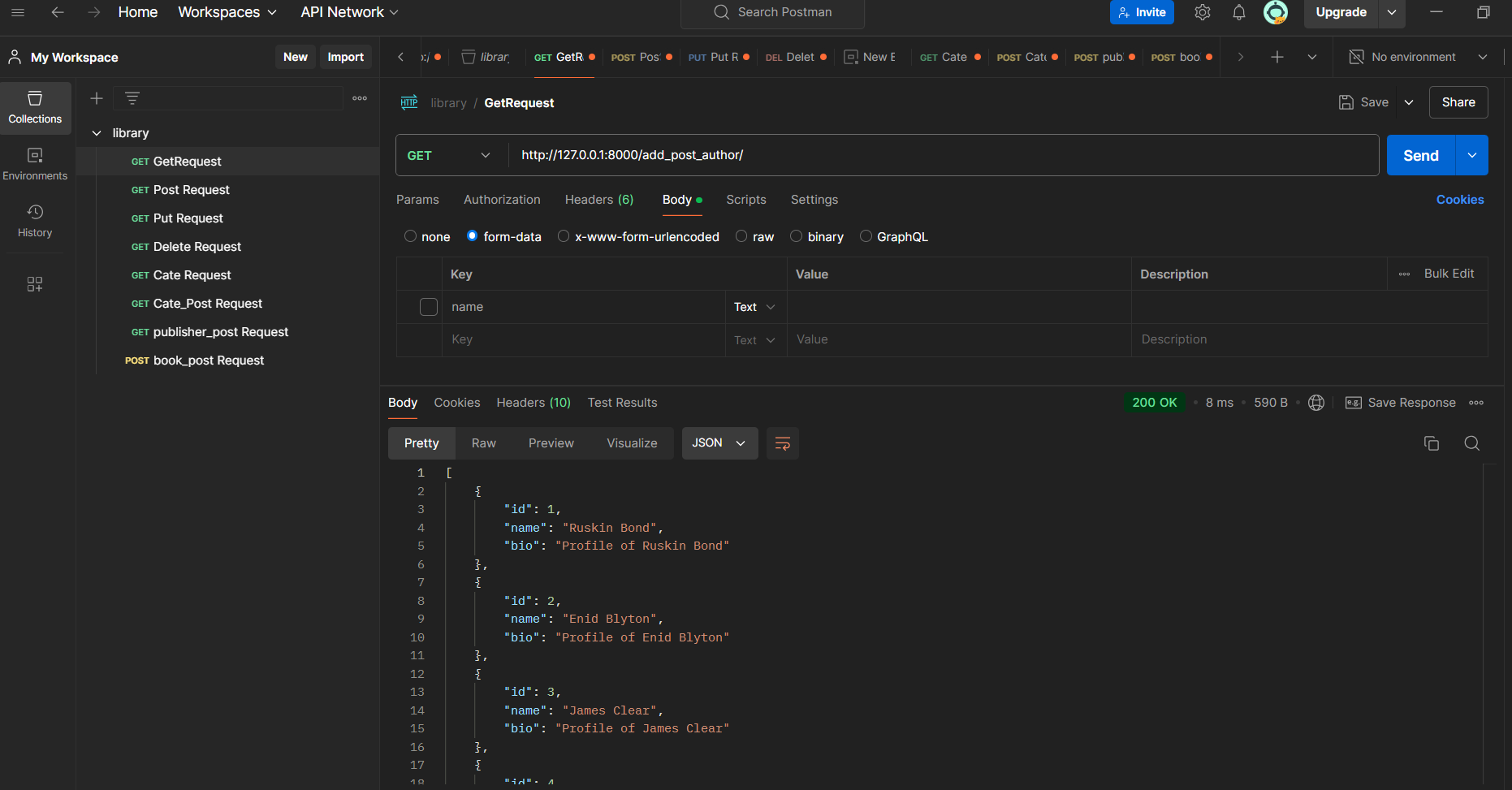
urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('library\_manage.urls')),

**]**

**OUTPUT:**

****