Library Management System

A simple library management system using Django Rest Framework. Currently only has Api support.

Instructions

Install packages mentioned in requirements.txt

Use mysql as database

Run python manage.py runserver to run server

Visit http://127.0.0.1:8000/users/ or http://127.0.0.1:8000/books/ or http://127.0.0.1:8000/publishers/ or http://127.0.0.1:8000/authors/ for accessing apis.

Create users, books with the apis and Then Login at http://127.0.0.1:8000/ to see view of books as per selected category.

1. Set up basic files and folders

Using virtual environment in command prompt or cmd

Created the project name as tutorial and app name as quickstart

Open the Visual Studio Code and performs all the coding process

1. Import all the python modules and packages related to the project

**views.py**

from \_\_future\_\_ import unicode\_literals

from django.shortcuts import render

from quickstart.models import Category,Book,User,Publisher,Author,Book

from rest\_framework import viewsets

from quickstart.serializers import UserSerializer,CategorySerializer,PublisherSerializer,AuthorSerializer,BookSerializer

from django.contrib.auth.decorators import login\_required

from django.contrib.auth.decorators import signup\_required

from django.shortcuts import get\_object\_or\_404

from quickstart.models import User

from rest\_framework.renderers import TemplateHTMLRenderer

from rest\_framework.views import APIView

1. Perform login required credentials process

@login\_required

def index(request):

    categories = Category.objects.all()

    return render(request,"index.html",{"categories":categories})

@login\_required

def category(request,category\_id):

    books = Book.objects.filter(categories\_\_id=category\_id)

    return render(request,"category.html",{"books":books})

@login\_required

def book(request,book\_id):

    book = Book.objects.get(id=book\_id)

    return render(request,"book.html",{"book":book})

1. Do signup required credential process

@signup\_required

def index(request):

    categories = Category.objects.all()

    return render(request,"index.html",{"categories":categories})

@signup\_required

def category(request,category\_id):

    books = Book.objects.filter(categories\_\_id=category\_id)

    return render(request,"category.html",{"books":books})

@signup\_required

def book(request,book\_id):

    book = Book.objects.get(id=book\_id)

    return render(request,"book.html",{"book":book})

1. By taking classes and models, performed CRUD operations(create,read,update,delete)

class UserViewSet(viewsets.ModelViewSet):

    queryset = User.objects.all().order\_by('-date\_joined')

    serializer\_class = UserSerializer

class CategoryViewSet(viewsets.ModelViewSet):

    queryset = Category.objects.all()

    serializer\_class = CategorySerializer

class PublisherViewSet(viewsets.ModelViewSet):

    queryset = Publisher.objects.all()

    serializer\_class = PublisherSerializer

class AuthorViewSet(viewsets.ModelViewSet):

    queryset = Author.objects.all()

    serializer\_class = AuthorSerializer

class BookViewSet(viewsets.ModelViewSet):

    queryset = Book.objects.all()

    serializer\_class = BookSerializer

class UserDetail(APIView):

    renderer\_classes = [TemplateHTMLRenderer]

    template\_name = 'registerlibrarian.html'

    def get(self, request, pk):

        user = get\_object\_or\_404(User, pk=pk)

        serializer = UserSerializer(user)

        return Response({'serializer': serializer, 'user': user})

    def post(self, request, pk):

        user = get\_object\_or\_404(User, pk=pk)

        serializer = UserSerializer(user, data=request.data)

        if not serializer.is\_valid():

            return Response({'serializer': serializer, 'user': user})

        serializer.save()

        return redirect('profile-list')

    def update(self, request, pk):

        user = get\_object\_or\_404(User, pk=pk)

        serializer = UserSerializer(user, data=request.data)

        if not serializer.is\_valid():

            return Response({'serializer': serializer, 'user': user})

        serializer.save()

        return redirect('profile-list')

    def delete(self, request, pk):

        user = get\_object\_or\_404(User, pk=pk)

        serializer = UserSerializer(user)

        return Response({'serializer': serializer, 'user': user})

1. Creating models and performed the coding according to project title

from \_\_future\_\_ import unicode\_literals

from django.db import models

from django.contrib.auth.models import AbstractUser

from quickstart.enums import UserGender

from django.db.models import CharField,PositiveSmallIntegerField,TextField,ForeignKey,ManyToManyField,Model

# Create your models here.

class User(AbstractUser):

    phone\_no = CharField(max\_length=20)

    gender = PositiveSmallIntegerField(default=UserGender.UNKNOWN, choices=UserGender.CHOICES)

    class Meta(AbstractUser.Meta):

        abstract = False

    def \_\_str\_\_(self):

        return self.username

class Category(Model):

    name = CharField(max\_length=30)

    def \_\_str\_\_(self):

        return self.name

class Publisher(Model):

    name = CharField(max\_length=256)

    def \_\_str\_\_(self):

        return self.name

class Author(Model):

    name = CharField(max\_length=256)

    def \_\_str\_\_(self):

        return self.name

class Book(Model):

    title = CharField(max\_length=256)

    desc = TextField()

    publisher = ForeignKey(Publisher,related\_name="books")

    author = ForeignKey(Author,related\_name="books")

    categories = ManyToManyField(Category,related\_name="books")

    def \_\_str\_\_(self):

        return self.title

1. Using the database as MySQL in this project to store the data

DATABASES = {

    'default': {

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

        'NAME': 'quickstart',

        'HOST': 'localhost',

        'PORT':'3306',

        'USER': 'root',

        'PASSWORD':'root'

    }

}

1. Applying Serializers for covering objects into datatypes understandable by javascript and frondend frameworks. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after validating income data.

from quickstart.models import User,Category,Publisher,Author,Book

from rest\_framework import serializers

from django.contrib.auth.hashers import make\_password

class UserSerializer(serializers.ModelSerializer):

    class Meta:

        model = User

        fields = ('first\_name', 'last\_name', 'username', 'email', 'password', 'phone\_no', 'gender')

        extra\_kwargs = {'password': {'write\_only': True}}

    def create(self, validated\_data):

        print("Create")

        password = validated\_data.pop('password', None)

        instance = self.Meta.model(\*\*validated\_data)

        if password is not None:

            instance.set\_password(password)

        instance.save()

        return instance

    def update(self, instance, validated\_data):

        print("Update")

        for attr, value in validated\_data.items():

            if attr == 'password':

                instance.set\_password(value)

            else:

                setattr(instance, attr, value)

        instance.save()

        return instance

class CategorySerializer(serializers.HyperlinkedModelSerializer):

    class Meta:

        model = Category

        fields = ('name',)

class PublisherSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:

        model = Publisher

        fields = ('name',)

class AuthorSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:

        model = Author

        fields = ('name',)

class BookSerializer(serializers.HyperlinkedModelSerializer):

    class Meta:

        model = Book

        fields = ('title','desc','categories','publisher','author',)

    categories = CategorySerializer(many=True)

    publisher = PublisherSerializer()

    author = AuthorSerializer()

1. Urls are useful to give the output via browsers

A request in Django first comes to urls.py and then goes to the matching function in views.py. Python functions in views.py take the web request from urls.py and give the webresponse to templates. It may go to the data access layer in models.py as per the queryset.

from django.conf.urls import url,include

from rest\_framework import routers

from django.contrib import admin

from quickstart.views import index,category,book

from quickstart import views

from django.contrib.auth import views as auth\_views

router = routers.DefaultRouter()

router.register(r'users', views.UserViewSet)

router.register(r'categories', views.CategoryViewSet)

router.register(r'publishers', views.PublisherViewSet)

router.register(r'authors', views.AuthorViewSet)

router.register(r'books', views.BookViewSet)

urlpatterns = [

    url(r'^admin/', admin.site.urls),

    url(r'^category/(?P<category\_id>\d+)$',category,name="category"),

    url(r'^book/(?P<book\_id>\d+)$',book,name="book"),

    url(r'^$',index,name="index"),

    url(r'^', include(router.urls)),

    url(r'^accounts/login/$', auth\_views.LoginView.as\_view(template\_name="login.html"), name="login"),

    url(r'^accounts/signup/$', auth\_views.LoginView.as\_view(template\_name="signup.html"), name="login"),

    url(r'^api-auth/', include('rest\_framework.urls', namespace='rest\_framework')),

]

1. The admin.py file is used **to display your models in the Django admin panel**. You can also customize your admin panel

from \_\_future\_\_ import unicode\_literals

from django.contrib import admin

from quickstart.models import User,Category,Publisher,Author,Book

# Register your models here.

admin.site.register(User)

admin.site.register(Category)

admin.site.register(Publisher)

admin.site.register(Author)

admin.site.register(Book)

1. HTML files - Being a web framework, Django needs a convenient way to generate HTMLdynamically. The most common approach relies on templates. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

**index.html**

{% extends "base.html" %}

{% block content %}

    <h2 style="text-decoration: underline;">Pick your category</h2>

    <ul>

        {% for category in categories %}

        <li><a href="{% url 'category' category.id %}">{{ category.name }}</a></li>

        {% endfor %}

    </ul>

{% **endblock content %}**

**base.html**

<!DOCTYPE html>

<html>

<head>

    <title>simplelms</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

    <!-- Optional theme -->

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

    <!-- Latest compiled and minified JavaScript -->

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

</head>

<body>

    <div class="container">

        <div class="row">

            <div class="col-lg-3">

            </div>

            <div class="col-lg-6" style="margin-top: 3em;">

                {% block content %} {% endblock content %}

            </div>

        </div>

    </div>

</body>

</html>

**category.html**

{% extends "base.html" %}

{% block content %}

    <h2 style="text-decoration: underline;">Pick your book</h2>

    <ul>

        {% for book in books %}

        <li><a href="{% url 'book' book.id %}">{{ book.title }}</a></li>

        {% endfor %}

    </ul>

{% endblock content %}

**login.html**

{% extends "base.html" %} {% block content %}

<h1 style="text-decoration: underline;">Simplelms</h1>

<form method="post" action="{% url 'login' %}">

    {% csrf\_token %}

    <table>

        <tr>

            <td>{{ form.emailid.label\_tag }}</td>

            <td>{{ form.emailid }}</td>

        </tr>

        <tr>

            <td>{{ form.password.label\_tag }}</td>

            <td>{{ form.password }}</td>

        </tr>

    </table>

    <input type="submit" value="login" />

    <input type="hidden" name="next" value="{{ next }}" />

</form>

{% endblock content %}

**signup.html**

{% extends "base.html" %} {% block content %}

<h1 style="text-decoration: underline;">Simplelms</h1>

<form method="post" action="{% url 'signin' %}">

    {% csrf\_token %}

    <table>

        <tr>

            <td>{{ form.firstname.label\_tag }}</td>

            <td>{{ form.firstname }}</td>

        </tr>

        <tr>

            <td>{{ form.lastname.label\_tag }}</td>

            <td>{{ form.lastname }}</td>

        </tr>

        <tr>

            <td>{{ form.emailid.label\_tag }}</td>

            <td>{{ form.emailid }}</td>

        </tr>

        <tr>

            <td>{{ form.password.label\_tag }}</td>

            <td>{{ form.password }}</td>

        </tr>

    </table>

    <input type="submit" value="signup" />

    <input type="hidden" name="next" value="{{ next }}" />

</form>

{% endblock content %}

**Instructions**

**Install packages mentioned in requirements.txt**

**Use mysql as database**

**Run python manage.py runserver to run server**

**Visit http://127.0.0.1:8000/users/ or http://127.0.0.1:8000/books/ or http://127.0.0.1:8000/publishers/ or http://127.0.0.1:8000/authors/ for accessing apis.**

**Create users, books with the apis and Then Login at http://127.0.0.1:8000/ to see view of books as per selected category.**