**Library Management System - Django Project Setup Guide**

**1. Introduction**

This document provides step-by-step instructions on setting up and running the Library Management System in Django, using MySQL as the database. It includes all necessary configurations, dependencies, and how to restore a MySQL backup using MySQL Workbench.

**2. Project Overview**

The Library Management System (LMS) is built using **Django**, **Django REST Framework**, and **MySQL** for backend operations. It allows for:

* Admin operations to manage books and user data.
* A student view that lists all books.
* Use of JWT for secure authentication (optional).

**3. Requirements**

Before running the project, make sure you have the following installed:

1. **Python 3.6 or higher**
   * [Download Python](https://www.python.org/downloads/)
2. **Visual Studio Code (VS Code)**
   * [Download Visual Studio Code](https://code.visualstudio.com/)
3. **MySQL**
   * [Download MySQL](https://dev.mysql.com/downloads/installer/)
4. **MySQL Workbench** (for backup/restore)
   * [Download MySQL Workbench](https://dev.mysql.com/downloads/workbench/)

## ****4. Setting Up the Django Project****

### **Step 1: Install Dependencies**

1. **Open VS Code**.
2. **Open the integrated terminal** (View → Terminal).
3. **Create a virtual environment**:

python -m venv venv

source venv/bin/activate # For Windows, use venv\Scripts\activate

1. **Install the required libraries**:

pip install django djangorestframework mysqlclient djangorestframework-jwt

Step 2: Create Django Project and App

1. Create a new Django project:

**django-admin startproject library\_management**

**cd library\_management**

1. Create the library app:

**python manage.py startapp library**

1. Open the project folder in VS Code using:

**code .**

**5. Set Up MySQL Database**

Step 3: Create the MySQL Database

1. Open **MySQL Workbench** and connect to your MySQL server.
2. **Create a new database** for the project:

**CREATE DATABASE library\_db;**

**6. Configure Database in Django**

Step 4: Configure Database in settings.py

1. In **library\_management/settings.py**, modify the DATABASES configuration as follows:

**DATABASES = {**

**'default': {**

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

**'NAME': 'library\_db', # Database name**

**'USER': 'your\_mysql\_user', # MySQL username**

**'PASSWORD': 'your\_mysql\_password', # MySQL password**

**'HOST': 'localhost',**

**'PORT': '3306',**

**}**

**}**

1. **Install MySQL client** if not installed:

**pip install mysqlclient**

**7. Create Models for Admin and Books**

Step 5: Define Models

In **library/models.py**, create the Admin and Book models:

from django.db import models

# Create your models here.

# library/models.

class Book(models.Model):

    title = models.CharField(max\_length=255)

    author = models.CharField(max\_length=255)

    published\_date = models.DateField()

    isbn\_number = models.CharField(max\_length=20)

    page\_count = models.IntegerField()

    def \_\_str\_\_(self):

        return self.title

# library/models.py

from django.db import models

class Admin(models.Model):

    name = models.CharField(max\_length=255)

    email = models.EmailField(unique=True)

    password = models.CharField(max\_length=255)  # We will hash this password

    def \_\_str\_\_(self):

        return self.email

class Book(models.Model):

    title = models.CharField(max\_length=255)

    author = models.CharField(max\_length=255)

    year = models.IntegerField()

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

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

    availability = models.BooleanField(default=True)

    def \_\_str\_\_(self):

        return self.title

# library/models.py

from django.db import models

class Admin(models.Model):

    # Your fields for the Admin model

    pass

class Book(models.Model):

# Your fields for the Book model

    pass

class Student(models.Model):

    name = models.CharField(max\_length=255)

    email = models.EmailField()

    # Add any other relevant fields for the student model

    books = models.ManyToManyField(Book)  # If students have many books

    def \_\_str\_\_(self):

        return self.name

**8. Create Serializers for Admin and Book**

Step 6: Create Serializers

Create **library/serializers.py**:

# library/serializers.py

from rest\_framework import serializers

from .models import Admin, Book, Student

class AdminSerializer(serializers.ModelSerializer):

    class Meta:

        model = Admin

        fields = '\_\_all\_\_'

class BookSerializer(serializers.ModelSerializer):

    class Meta:

        model = Book

        fields = '\_\_all\_\_'

class StudentSerializer(serializers.ModelSerializer):

    class Meta:

        model = Student

        fields = '\_\_all\_\_'

**9. Set Up Views for Admin and Books**

Step 7: Create Views

In **library/views.py**, create views for managing the Admin and Book models:

# library/views.py

from rest\_framework import viewsets

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import status

from .models import Admin, Book, Student

from .serializers import AdminSerializer, BookSerializer, StudentSerializer

# Admin ViewSet

class AdminViewSet(viewsets.ModelViewSet):

    queryset = Admin.objects.all()

    serializer\_class = AdminSerializer

# Book ViewSet

class BookViewSet(viewsets.ModelViewSet):

    queryset = Book.objects.all()

    serializer\_class = BookSerializer

# Student View to list all books

class StudentView(APIView):

    def get(self, request):

        books = Book.objects.all()

        serializer = BookSerializer(books, many=True)

        return Response(serializer.data)

from django.http import HttpResponse

def home(request):

    return HttpResponse("Welcome to the Library Management System!")

**10. Configure URL Routing**

Step 8: Set Up URL Routing

1. In **library/urls.py**, define the API routes:
2. Include the **library/urls.py** in **library\_management/urls.py**:

# library/urls.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import AdminViewSet, BookViewSet, StudentView

router = DefaultRouter()

router.register(r'admins', AdminViewSet)

router.register(r'books', BookViewSet)

urlpatterns = [

path('api/', include(router.urls)), # Register the viewsets with the router

path('api/student/books/', StudentView.as\_view(), name='student-books'), # Student view to list all books

]

# library\_management/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('library.urls')), # Include the library app's URLs

]

# library\_management/urls.py

from django.contrib import admin

from django.urls import path, include

from library.views import home # Import the home view

urlpatterns = [

path('admin/', admin.site.urls),

path('', home, name='home'), # Add the home view to the root URL

path('api/', include('library.urls')), # Include the library app's URLs for the API

]

**11. Create and Apply Migrations**

Step 9: Apply Migrations

Run the following commands to create the necessary database tables:

**python manage.py makemigrations**

**python manage.py migrate**

**12. Run the Django Server**

Step 10: Start the Development Server

Run the server using:

**python manage.py runserver**

Now, the server will be running at <http://localhost:8000>.

**13. Testing the Endpoints Using Postman or cURL**

You can test the API endpoints with **Postman** or **cURL**:

1. **Admin Signup** (POST request to /api/admins/)
2. **Admin Login** (POST request to /api/token/ for JWT token)
3. **CRUD Operations for Books** (Admin access to /api/books/)
4. **Student View** (GET request to /api/student/books/ to view all books)

**14. MySQL Database Backup & Restore Using MySQL Workbench**

Step 11: Backup the Database

1. Open **MySQL Workbench**.
2. Connect to your MySQL server.
3. Right-click on the database library\_db and select **"Data Export"**.
4. Choose to export the database as a .sql file.

Step 12: Restore the Database from Backup

1. In **MySQL Workbench**, go to **Data Import**.
2. Select the .sql backup file you created.
3. Choose the target database (e.g., library\_db), then click **Start Import**.

**15. Final Project Structure**

The project structure should look like this:

library\_management/

├ ── library/

│ ├ ── migrations/

│ ├ ── \_\_init\_\_.py

│ ├ ── admin.py

│ ├ ── apps.py

│ ├ ── models.py

│ ├ ── serializers.py

│ ├ ── tests.py

│ ├ ── views.py

│ └ ── urls.py

├ ── library\_management/

│ ├ ── \_\_init\_\_.py

│ ├ ── settings.py

│ ├ ── urls.py

│ └ ── wsgi.py

├ ── manage.py

└ ── db.sqlite3

**Conclusion**

This guide provides a detailed overview of setting up and running the Library Management System in Django, including creating the necessary MySQL database and restoring from a backup using MySQL Workbench.