📚 Library Management System — API Design & Implementation (Optimized)

🧾 Project Description

The **Library Management System API** is a RESTful service built with **Django REST Framework (DRF)** to manage books, authors, users, and borrowing records in a digital library.

Core Functionalities

- **Library Resources:** Create/update/delete **books**, **authors**, **categories** (Librarian only)
- **User Management:** Manage users and roles (`admin` = Librarian, `user` = Member)
- **Borrowing System:** Members can borrow and return books
- **Authentication:** JWT-based login & registration with **Djoser + SimpleJWT**
- **Documentation:** Swagger UI and ReDoc via **drf-yasg**

🗄️ Database Schema

1) Category

- `id` (Auto, PK)
- `name` (CharField, unique)

2) Author

- `id` (Auto, PK)
- `name` (CharField)

- `biography` (TextField, optional)

3) Book

- `id` (Auto, PK)
- `title` (CharField)
- `author` (FK → Author)
- `ISBN` (CharField, unique)
- `category` (FK → Category)
- `availability_status` (Boolean, default `true`) → `true = available`, `false = borrowed`

4) CustomUser (Librarian & Member)

- `id` (BigAutoField, PK)
- `email` (EmailField, **unique**, login field)
- `first_name` / `last_name` (CharField)
- `phone_number` (optional)
- `address` (optional)
- `membership_date` (DateField, `auto_now_add=True`)
- `password` (hashed)
- `role` (`"admin"` = Librarian, `"user"` = Member)
- `is_active`, `is_staff`, `is_superuser`

> Login via email (USERNAME_FIELD="email"). Members are just users with role="user".

5) BorrowRecord

- `id` (Auto, PK)
- `book` (FK → Book)
- `user` (FK → CustomUser)

- `borrow_date` (`auto_now_add=True`)
- `return_date` (nullable, set on return)
- `is_returned` (Boolean, default `false`)

Relationships

- Book → Author / Category (many-to-one)
- BorrowRecord → Book / CustomUser (many-to-one)

🔗 API Endpoints

1️⃣ Authentication (Djoser + JWT)

- `POST /auth/users/` → Register user (`role="user"` by default)
- `POST /auth/jwt/create/` → Login → get tokens
- `POST /auth/jwt/refresh/` → Refresh token
- `POST /auth/jwt/verify/` → Verify token

2️⃣ User Management

- `GET /api/users/me/` → Get own profile
- `PUT /api/users/me/` → Update own profile
- `GET /api/users/` → List all users (**Librarian only**)
- `GET /api/users/{id}/` → Retrieve a user (**Librarian only**)
- `PUT /api/users/{id}/` → Update user, incl. role (**Librarian only**)
- `DELETE /api/users/{id}/` → Delete a user (**Librarian only**)

3️⃣ Library Resources (Books / Authors / Categories)

Books

- `GET /api/books/` → List books (filter by `category`, `author`, `available`)
- `POST /api/books/` → Create book (**Librarian only**)

- `GET /api/books/{id}/` → Retrieve book
- `PUT /api/books/{id}/` → Update book (**Librarian only**)
- `DELETE /api/books/{id}/` → Delete book (**Librarian only**)

Authors

- `GET /api/authors/` → List authors
- `POST /api/authors/` → Create author (**Librarian only**)
- `GET /api/authors/{id}/` → Retrieve author
- `PUT /api/authors/{id}/` → Update author (**Librarian only**)
- `DELETE /api/authors/{id}/` → Delete author (**Librarian only**)

Categories

- `GET /api/categories/` → List categories
- `POST /api/categories/` → Create category (**Librarian only**)
- `GET /api/categories/{id}/` → Retrieve category
- `PUT /api/categories/{id}/` → Update category (**Librarian only**)
- `DELETE /api/categories/{id}/` → Delete category (**Librarian only**)

4️⃣ Borrowing

- `POST /api/borrow/` → Borrow a book (Member)

```
{ "book": <book_id> }
```

- `PUT /api/borrow/{id}/return/` → Return a book (Member; only borrower)
- `GET /api/borrow/my/` → List own borrow records (Member)
- `GET /api/borrow/` → List all borrow records (**Librarian only**)
- `GET /api/borrow/{id}/` → Retrieve borrow record (**Librarian or owner**)
- `DELETE /api/borrow/{id}/` → Delete borrow record (**Librarian only**)

Borrowing Rules

- Book must be available (`availability_status=true`)
- Borrow → `availability_status=false`
- Return → `return_date=today` , `is_returned=true` , `availability_status=true`

🔐 Roles & Permissions

Action	Librarian (`admin`)	Member (`user`)
Manage Books/Authors/Category	✅	❌
Borrow/Return Books	❌	✅
Manage Users	✅	❌
View Books	✅	✅
View Own Profile	✅	✅
View All Borrow Records	✅	❌

🧪 Example Requests

Register Member

```
{
  "email": "john@example.com",
  "password": "mypassword123",
  "first_name": "John",
  "last_name": "Doe"
}
```

Login

```
{ "email": "john@example.com", "password": "mypassword123" }
```

Create Book (Librarian)

```
{
  "title": "Clean Code",
  "author": 1,
  "ISBN": "9780132350884",
  "category": 1,
  "availability_status": true
}
```

Borrow Book (Member)

```
{ "book": 1 }
```

Return Book (Member)

```
PUT /api/borrow/1/return/
```

⚙️ Implementation Notes

- `CustomUser` extends `AbstractBaseUser + PermissionsMixin`
- `USERNAME_FIELD="email"`; `role` defines Librarian/Member
- Use **IsAuthenticated** for general access, **custom role check** for admin-only actions
- JWT tokens via **SimpleJWT**, user registration/login via **Djoser**
- Auto-update book availability on borrow/return

📄 Documentation

- Swagger UI → `/swagger/` (interactive)
- ReDoc → `/redoc/` (clean static)
- Ensure docstrings on all ViewSets and Serializers

🧭 Testing Checklist

1. Register Member → login → get JWT
2. Create Librarian superuser → add Category, Author, Book
3. Borrow book as Member → check availability update
4. Return book → check availability update
5. Test restricted endpoints as Member → 403 forbidden
6. Test borrow history `/api/borrow/my/` vs `/api/borrow/`

📂 Project Structure

```
LibraryManagement/
│
├── library_management/          # Main project config (settings, URLs, WSG
I)
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│
├── api/                    # API versioning & routing
│   ├── __init__.py
│   ├── urls.py
│
├── users/                    # Authentication & Profile Management
│   ├── migrations/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── serializers.py
```

```
│   ├── views.py
│   ├── signals.py
│   ├── urls.py
│
├── library/                # Books, authors, categories, borrow records
│   ├── migrations/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   ├── permissions.py
│   ├── urls.py
│
├── fixtures/               # Sample data
│   └── fixtures.json
│
├── media/                  # (Optional) Local media files
│
├── Dockerfile
├── docker-compose.yml
├── requirements.txt
├── manage.py
```

📄 Create Project

```
# 1. Create project directory and enter it
mkdir library_management
cd library_management

# 2. Create and activate virtual environment
python3 -m venv venv
```

```
# On Linux/macOS:
source venv/bin/activate
# On Windows (cmd):
# venv\Scripts\activate
# On Windows (PowerShell):
# .\venv\Scripts\Activate.ps1

# 3. Upgrade pip for safety
pip install --upgrade pip

# 4. Install core dependencies
pip install django djangorestframework djoser pillow django-filter
pip install djangorestframework-simplejwt  # JWT support

# 5. (Optional) Install Cloudinary for image handling
pip install cloudinary django-cloudinary-storage

# 6. Start Django project named 'library_management' (replace 'core' if you w
ant)
django-admin startproject library_management .

# 7. Start apps
python manage.py startapp users
python manage.py startapp library
python manage.py startapp api

# 8. Create initial migrations
python manage.py makemigrations
python manage.py migrate

# 9. Create superuser for admin access
python manage.py createsuperuser

# 10. Run development server to check setup
python manage.py runserver
```