Estructura del Proyecto app Alquiler de libros backend con Django

Para la creación de esta app se usará Django en el backend y Vue en el frontend.

Django

Se desarrollará una API sin utilizar Django Rest Framework (DRF), ya que no se considera necesario implementarlo para un proyecto de esta magnitud.

1. aplicaciones

Shared	Artefactos compartidos
Books	Gestión de libros
Rents	Gestión de alquileres
Users	Gestión de usuarios

1.2 Books

isbn(*u)	str
title	str
slug	str
autor	str

description	str
available	boolean
Category	charfield(Choice)

- El isbn debe ser único para cada libro

- available indica si el libro está disponible TRUE o FALSE si no lo está.

- ISBN representa el código ISBN (International Standard Book Number) del libro y debe ser único para cada uno.
- CATEGORY_CHOICES = [
('fantasy', 'Fantasía'),
('scifi', 'Ciencia Ficción'),
('horror', 'Terror'),
('thriller', 'Suspenso'),
('historical', 'Histórica'),
('classic', 'Clásico'),
('manga', 'Manga'),
]

1.3 Rents

book	fk → Book
user	fk → User
start_rent	datetime
end_rent	datetime
return	datetime

- book será una clave ajena de books
- user será una clave ajena de users
- start rent será un datetime de cuando comenzó el alquiler del libro
- end_rent un datetime de cuándo se debe entregar
- return un datetime de cuando se entregó el libro

1.4 Users

Token	uuid

- Token será un uuid que se le proporcionará al usuario este cambiará cada vez que inicie sesión para garantizar que si alguien obtiene el token no pueda usarlo.

2. Carga de datos

Para cargar los datos iniciales de la aplicación en la base de datos se usará el siguiente comando just:

```
just load-data
```

En caso de no disponer de just se usará el siguiente comando en la terminal señalando cada archivo json de carga de datos manualmente:

```
python manage.py loaddata <ruta>/< nombre_del_archivo>.json
```

3. URLS

EL backend en django es una api por lo que todos los modelos o la extensa mayoría devolverán una respuesta en formato JSON

3.1 books.urls

api/books/ ⇒ books.views.book_list()

- Listado de los libros disponibles en el sistema.

GET request	**JSON** response (200)
	book(◔) book(◔) book(◔)

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed

/api/books/?isbn=335353&autor=cervantes/ ⇒ books.views.book_list()

Listado de los libros disponibles en el sistema filtrando por los parámetros de la petición querystring.

GET request	**JSON** response (200)
isbn title autor	book book book

- isbn,title y autor son parámetros de la petición querystring.
- Se puede filtrar por uno, por otro o por todos.

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed

api/books/34356/ ⇒ books.views.book_detail()

- Detalle del libro con isbn 34356.

Get request	JSON response (200)
	isbn title autor description available

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed
404	Book not found

api/books/add ⇒ books.views.add_bookl()

- Solo los perfiles administrativos pueden añadir libros.
- Añade un nuevo libro a la base de datos.

Header	Post request	JSON response (request)
Token (bearer)	isbn title autor description	isbn title

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token

401	profile does not have administrator permissions.
404	Book not found

api/books/remove ⇒ books.views.remove_book()

- Solo los perfiles administrativos pueden eliminar un libro.

Header	Post request	JSON response (request)
Token (bearer)	isbn	book deleted succesfully

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
401	profile does not have administrator permissions.
404	Book not found

3.2 rentals.urls

api/rentals/ ⇒ rentals.views.view_rents()

- Muestra todas las rentas del usuario actual

Header	Post request	JSON response (request)

Token (bearer)		Rentals …

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed

api/rentals/rent_book ⇒ rentals.views.rent_book()

- Añade un nuevo alquiler

Header	Post request	JSON response (request)
Token (bearer)	isbn start_rent(datetime)	title start rent end rent

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
409	The book is already rented.
404	Book not found

api/rentals/return_rent ⇒ rentals.views.return_rent()

Devuelve el libro alquilado

Header	Post request	JSON response (request)
Token (bearer)	pk(rent) return_rent(datetime)	Devuelto con éxito

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
404	Rent not found

3.3 users.urls

api/users/register ⇒ users.views.register()

- El usuario podrá registrarse poniendo sus credenciales nombre, apellidos, correo electrónico, nombre de usuario y contraseña.
- Se deberá proporcionar un Bearer token único para cada usuario en su primer registro.
- Control a nivel de backend si falta algún campo que rellenar

Header	Post request	JSON response (request)
	name lastname username password email	Token(Bearer)

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields

api/users/login ⇒ users.views.login()

- El usuario podrá iniciar sesión insertando su nombre de usuario o correo y su contraseña.
- Se deberá proporcionar un nuevo Bearer token único en cada inicio de sesión.
- Control a nivel de backend si el usuario inserta mal su contraseña, usuario o le falta algún campo que rellenar al iniciar sesión.

Header	Post request	JSON response (request)
Token(Bearer)	username password	Mensaje de éxito

Devuelve una respuesta JSON con una clave error y un mensaje informativo atendiendo a los siguientes casos (el orden de los errores es importante):

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields
400	Invalid authentication token
401	Unregistered authentication token
404	User not found

api/users/logout ⇒ users.views.logout()

- El usuario podrá cerrar sesión.
- Cuando el usuario cierre sesión se deberá eliminar su Bearer token.

Header	Post request	JSON response (request)
Token(Bearer)		Mensaje de éxito

HTTP status	Error
405	Method not allowed
400	Invalid Json
400	Missing required fields
400	Invalid authentication token

4. Serializadores

Serializadores para Books y para rentals Además de para user:
BOOKS:

- isbn
- title
- autor
- description
- available

RENTALS:

- pk(pk-rental)
- book
- user
- start rent
- end_rent
- return

USERS:

- id (pk-user)
- username
- first_name
- last_name
- email

5. Administración

Los siguientes modelos deben estar accesibles desde la interfaz administrativa de Django:

- books.Book
- rentals.Rental
- users.User
- users.Token

Estructura de carpetas

```
├── backend/                    # Carpeta para el proyecto Django
│   ├── manage.py                 # Archivo para ejecutar comandos de Django
│   ├── alquiler libros/          # Proyecto Django
│   │   ├── __init__.py
│   │   ├── settings.py           # Configuración de Django
│   │   ├── urls.py               # URLs globales
│   │   ├── wsgi.py
│   │   ├── asgi.py
│   │   ├── apps/                 # Aplicaciones de Django
│   │   │   ├── shared/           # Artefactos compartidos (funcionalidad común)
│   │   │   │   ├── decorators.py        # Decoradores compartidos
│   │   │   │   ├── __init__.py
│   │   │   │   └── utils.py      # Funciones utilitarias comunes
│   │   │   ├── books/            # Gestión de libros
│   │   │   │   ├── __init__.py
│   │   │   │   ├── models.py       # Modelos de libros
│   │   │   │   ├── views.py        # Lógica de vistas de libros
```

```
│   │   │   │   ├── serializers.py        # Serializadores de libros
│   │   │   │   ├── urls.py               # URLs de libros
│   │   │   │   ├── admin.py              # Registro de libros en el admin de Django
│   │   │   ├── rents/                    # Gestión de alquileres
│   │   │   │   ├── __init__.py
│   │   │   │   ├── models.py             # Modelos de alquileres
│   │   │   │   ├── views.py              # Lógica de vistas de alquileres
│   │   │   │   ├── serializers.py        # Serializadores de alquileres
│   │   │   │   ├── urls.py               # URLs de alquileres
│   │   │   │   ├── admin.py              # Registro de alquileres en el admin de Django
│   │   │   ├── users/                    # Gestión de usuarios
│   │   │   │   ├── __init__.py
│   │   │   │   ├── models.py             # Modelos de usuarios
│   │   │   │   ├── views.py              # Lógica de vistas de usuarios
│   │   │   │   ├── decorators.py               # Decoradores de usuario
│   │   │   │   ├── serializers.py        # Serializadores de usuarios
│   │   │   │   ├── urls.py               # URLs de usuarios
│   │   │   │   ├── admin.py              # Registro de usuarios en el admin de Django
│   │   └── requirements.txt              # Dependencias del backend (Django, etc.)
```