Project 1: "Community Platform"

Project Overview:

Create a community platform using Django REST Framework where users can interact through JSON APIs. The project will allow students to explore building APIs for user registration, posting, commenting, liking, and user interaction functionalities.

Core Features:

User Registration and Authentication:

- API endpoints for user registration, login, and logout.
- API for password reset functionality.
- API to manage user profiles (GET, PUT).

User Interaction:

Posting Content:

- API endpoints to create, edit, and delete posts (POST, PUT, DELETE).
- API to upload images and manage optional tags (POST, PUT).

Commenting:

- API endpoints for posting comments and nested comments (POST).
- API to edit and delete comments (PUT, DELETE).

Liking:

- API to like or unlike posts and comments (POST, DELETE).

Content Feed:

- API to retrieve a feed of recent posts (GET).
- API to filter the feed by tags or search keywords (GET with query parameters).
- API to sort the feed by newest, most liked, or most commented (GET with sorting parameters).

User Profiles:

- API to retrieve user profile data, including posts, comments, and liked content (GET).
- API to follow/unfollow users and manage personalized feeds (POST, DELETE).
- API for sending direct messages between users (POST).

Notifications:

- API for managing notifications (GET, POST, DELETE).
- GET requests should return all new notifications since the last time notifications were received
- POST requests allow for retrieving notifications in any time frame

Optional Features:

- APIs for creating and managing groups, forums, media galleries, events, etc. (CRUD operations).
- REST API integration for interaction with external applications.

Project Structure:

Models:

- `User`, `Post`, `Comment`, `Like`, `Profile`, `Notification`, and optional models like `Group`, `Event`, `Media`.

Views:

- API views (using Django REST Framework ViewSets or APIViews) for CRUD operations on posts, comments, and profiles.
- Function-based API views for actions like liking a post, following a user, etc.

Serializers:

- Serializers for all models to handle JSON data conversion.

URLs:

- URL patterns mapped to API endpoints for each action.

Assessment Criteria:

- API functionality: Correct implementation of endpoints for registration, posting, commenting, liking, etc.
- Code structure: Proper use of DRF views, serializers, and reusable components.
- Customization: Encouraging students to add custom endpoints or modify existing ones.
- API documentation: Clear and detailed documentation for each endpoint.
- Optional feature integration: Bonus for students who implement additional APIs.

Project 2: "E-commerce Platform"

Project Overview: Develop an e-commerce platform using Django REST Framework where users can interact with the online store through JSON APIs. This project covers API development for product browsing, cart management, checkout, order processing, and inventory management.

Core Features:

User Registration and Authentication:

- API endpoints for user registration, login, and profile management (POST, PUT, GET).

Product Management:

- API endpoints for admins to create, edit, and delete products (POST, PUT, DELETE).
- API to manage product images, descriptions, tags, and inventory (CRUD operations).
- API for category management with JSON schema validation (CRUD operations).
- Implement a category model with a tree structure. Each category should have an additional field named attributes_schema which contains jsonschema of product attributes
- Each product should have a category (only leaf categories are allowed) and attribute field (in JSON format) which should be validated with category jsonschema.

Shopping Cart:

- API endpoints to add, update, or remove items from the cart (POST, PUT, DELETE).
- API for persisting the cart between sessions (using database).

Checkout Process:

- API endpoints for reviewing the cart and proceeding to checkout (GET, POST).
- API to create orders and clear the cart upon checkout completion (POST).

Order Management:

- API for users to view past orders and track current ones (GET).
- API for admins to manage orders, update statuses, and handle refunds (PUT, DELETE).

Notifications:

- API for managing notifications (GET, POST, DELETE).
- GET requests should return all new notifications since the last time notifications were received
- POST requests allow for retrieving notifications in any time frame
- Admins receive notifications of new orders, and users get notifications of any status changes in their orders

Product Search and Filtering:

- API to search for products by name or description and filter by price range, category, or tags (GET with query parameters).

Optional Features:

- APIs for managing discounts, coupons, product reviews, wish lists, etc. (CRUD operations).

Project Structure:

Models:

- `User`, `Product`, `Cart`, `Order`, `Payment`, and optional models like `Review`, `Coupon`.

Views:

- API views (using Django REST Framework) for CRUD operations and cart management.

- Function-based API views for checkout, payment processing, and order management.

Serializers:

- Serializers for all models to handle JSON data conversion.

URLs:

- URL patterns mapped to API endpoints for product catalog, cart, checkout, and order management.

Assessment Criteria:

- API functionality: Correct implementation of endpoints for product catalog, cart, checkout, etc.
- Code structure: Proper use of DRF views, serializers, and reusable components.
- Customization: Encouraging students to add custom endpoints or modify existing ones.
- API documentation: Clear and detailed documentation for each endpoint.
- Optional feature integration: Bonus for students who implement additional APIs.

Project 3: "Healthcare Appointment System"

Project Overview: Create a healthcare appointment system using Django REST Framework where patients and doctors interact with the platform via JSON APIs. This project focuses on building APIs for user roles, appointment scheduling, and medical record management.

Core Features:

User Roles:

- API endpoints for different user roles: Admin, Doctor, and Patient (CRUD operations).
- Role-based access control implemented through DRF permissions and view restrictions.

Doctor Management:

- API for admins to manage doctor profiles (CRUD operations).
- API for doctors to manage their availability and appointment slots (GET, PUT).

Patient Management:

- API endpoints for patients to manage their profiles (CRUD operations).
- API to view available doctors and book appointments (GET, POST).

Appointment Scheduling:

- API to browse available doctors and book appointments based on availability (GET, POST).
- API for doctors to view and manage their appointment schedule (GET, PUT).
- API for appointment reminders via email or SMS (POST).

Medical Records:

- API for doctors to create and update patient medical records (POST, PUT).
- API for patients to view their medical history (GET).

Optional Features:

- APIs for managing prescriptions, billing, and integration with external health record systems or insurance providers (CRUD operations).

Project Structure:

Models:

- `User` (extended with roles), `Doctor`, `Patient`, `Appointment`, `MedicalRecord`, and optional models like `Prescription`, `Bill`.

Views:

- API views (using Django REST Framework) for CRUD operations on doctors, patients, and appointments.
- Function-based API views for scheduling, notifications, and reminders.

Serializers:

- Serializers for all models to handle JSON data conversion.

URLs:

- URL patterns mapped to API endpoints for role-specific actions, appointment booking, and management.

Assessment Criteria:

- API functionality: Correct implementation of endpoints for user roles, appointment booking, and schedule management.
- Code structure: Proper use of DRF views, serializers, and role-based access control.
- Customization: Encouraging students to add custom endpoints or modify existing ones.
- API documentation: Clear and detailed documentation for each endpoint.
- Optional feature integration: Bonus for students who implement additional APIs.