Project Outline: Blogging Platform API

1. Project Idea

The goal of this project is to develop a robust, scalable, and fully functional **Blogging Platform API** using **Django** and **Django REST Framework (DRF)**. This API will serve as the backend for any modern frontend application (React, Vue, mobile app, etc.), providing all necessary functionalities for content creation, management, and retrieval.

2. Key Features

The platform will include two main functional areas: User/Authentication and Content Management.

User & Authentication

- **User Registration & Login:** Secure system using JWT (JSON Web Tokens) for authentication.
- **User Roles:** Differentiate between standard users (readers) and authors (users who can create posts).
- **Profile Management:** Users can view their own profiles.

Content Management (Blog Posts)

- **Post CRUD:** Full Create, Read, Update, and Delete operations for blog posts.
- **Draft/Publish Status:** Posts can be saved as drafts or published.
- **Categorization:** Posts can be assigned to categories.
- **Filtering:** Specific endpoints to retrieve posts based on an Author or a Category.
- **Search:** Ability to search for posts by title or content keywords.

3. Technical Stack and API

- **Primary Framework:** Django (Python)
- **API Framework:** Django REST Framework (DRF)
- **Database (Local):** SQLite (for development)
- **Database (Production):** PostgreSQL (required for Heroku/PythonAnywhere deployment)
- **Authentication:** Simple JWT (for token-based authentication)

4. Models, API Endpoints, and Technical Design

A. Database Models (Django ORM)

Model	Fields	Relationships	Notes

User	username, email, password, first_name, last_name, is_author (boolean)	N/A	Extend Django's built-in AbstractUser. is_author determines if they can create/edit posts.
Category	name, slug	N/A	Used for filtering and organizing posts. slug for clean URLs.
Post	title, slug, content, status (Draft/Published), created_at, updated_at	author (ForeignKey to User), category (ForeignKey to Category)	The core content model. Permissions are based on the author field.

B. API Endpoints

All endpoints will be prefixed with /api/v1/.

Endpoint	**Method**	**Description**	**Authentication Required**
/auth/register/	POST	Register a new user.	No
/auth/login/	POST	Authenticate and return JWT token.	No
/posts/	GET	List all **published** posts.	No
/posts/	POST	Create a new blog post.	Yes (Author)
/posts/<slug>/	GET	Retrieve a single post.	No
/posts/<slug>/	PUT/PATCH	Update a post.	Yes (Author, must

			be post owner)
/posts/<slug>/	DELETE	Delete a post.	Yes (Author, must be post owner)
/posts/category/<category_slug>/	GET	List all published posts for a given category.	No
/posts/author/<username>/	GET	List all published posts for a given author.	No
/categories/	GET	List all categories.	No

5. 5-Week Project Plan

This plan breaks down the project into manageable weekly sprints, with an estimated time allocation for each major task.

📅 Week 1: Setup and Core Models (20% Complete)

Task	Details
Project Setup	Initialize Django project, create virtual environment, install basic dependencies (Django, djangorestframework).
Database Configuration	Configure development database (SQLite).
Model Creation	Define User, Category, and Post models in models.py.
Admin Panel	Register all models in the Django Admin for easy initial data management.
Testing	Write basic model tests to ensure data integrity.

📅 Week 2: Authentication and Basic API (40% Complete)

Task	Details
Install Authentication	Install and configure djangorestframework-simplejwt.
Auth Endpoints	Implement register, login, and refresh token endpoints.
User Serializers/Views	Create serializers and views for user list (admin only) and user profile retrieval.
Permission Class	Implement a custom permission class to check for is_author status.

📅 Week 3: Post CRUD Functionality (65% Complete)

Task	Details
Post Serializer	Create PostSerializer (including nested data for author and category).
Post ViewSet	Implement the PostViewSet with full CRUD operations.
Permission Integration	Apply permissions to ensure: 1) Only authors can create. 2) Only the post owner can update/delete. 3) Only published posts are publicly readable.
Slug Generation	Implement automatic slug generation on post creation/title update.

📅 Week 4: Filtering and Custom Endpoints (85% Complete)

Task	Details
Category Endpoints	Implement ListAPIView for listing categories.
Filtering by Category	Implement the /posts/category/<slug>/ endpoint using get_queryset filtering.

Filtering by Author	Implement the /posts/author/<username>/ endpoint.
Search Functionality	Integrate Django REST Framework's SearchFilter to allow keyword searches on post titles and content.

📅 Week 5: Polish, Testing, and Deployment (100% Complete)

Task	**Details**
Comprehensive Testing	Write unit and integration tests for all major views and permissions.
Documentation	Generate API documentation (e.g., using drf-spectacular for OpenAPI/Swagger documentation).
Deployment Prep	Finalize requirements.txt, create Procfile for Heroku, configure production settings (environment variables, security keys).
Deployment	Deploy the application to the chosen service (Heroku or PythonAnywhere) and confirm all endpoints are functional.

6. Important Planning Notes

Production Database

It is critical to switch from SQLite to **PostgreSQL** (or a similar production-grade database) before deployment. Cloud platforms like Heroku/PythonAnywhere will reset SQLite databases regularly.

Permissions

The core logic of this project relies on custom DRF permissions. I will need to define:

1. IsAuthenticatedAuthor: Allows POST only if the user is logged in and user.is_author is true.
2. IsOwnerOrReadOnly: Allows PUT/PATCH/DELETE only if the user is the post's author, otherwise read-only access.

URL Structure

I will use the slug field in the URL for better SEO and readability (e.g., /posts/my-first-blog-post-slug/) rather than using primary keys (IDs).