News Application: Project Plan

1. Project Overview

This document outlines the planning and design for a Django-based news application. The application will serve as a platform for independent journalists and publishers to share articles. It will feature a user-role system with distinct permissions for Readers, Journalists, and Editors, a subscription service for users to follow publications and journalists, and an automated system for article dissemination via email and a social media API. A key aspect of this project is the design of a modern, "newspaper-themed" user interface.

2. Functional Requirements

- **User Management**: Implement a custom user model with three distinct roles: Reader, Journalist, and Editor.
- **Content Management**: Journalists can create, view, update, and delete their own articles and newsletters. Editors can approve all content.
- **Subscription System**: Readers can subscribe to publishers and individual journalists.
- **Dissemination**: When an editor approves an article or newsletter, it's sent to all relevant subscribers via email and posted to an X account.
- **API**: A RESTful API allows third-party clients to retrieve subscribed articles and newsletters.

3. Non-Functional Requirements

- **Performance**: The application is efficient and responsive, even with numerous users and content.
- **Security**: All access control and user permissions are strictly enforced.
- **Scalability**: The database schema is normalized to support a large number of users and articles.
- **Code Quality**: The code is well-formatted, readable, modular, and adheres to PEP 8 standards.
- **Database**: The final database uses MariaDB.
- **UI/UX**: The front end has a modern, newspaper-style theme that is intuitive and easy to use.

4. Database Schema Design

- **CustomUser**: `username`, `email`, `password`, `role`, `subscriptions_publishers` (Many-to-Many), `subscriptions_journalists` (Many-to-Many).
- **Publisher**: `name`, `description`.

- **Article**: `title`, `content`, `approved`, `author` (Foreign Key), `publisher` (Foreign Key).
- **Newsletter**: `title`, `content`, `approved`, `author` (Foreign Key), `publisher` (Foreign Key).

5. Front-End UI/UX Plan

- **Typography**: Use a classic serif font for headlines and a clean sans-serif font for body text.
- **Color Palette**: A minimalist palette of black, white, and a single accent color for buttons and links.
- **Layout**: A grid-based layout that mimics a newspaper, with a clear and persistent navigation bar.
- **Visual Elements**: Use subtle textures and high-quality images to reinforce the newspaper theme.

6. Back-End Architecture Plan

- **Django Project Structure**: A single Django project with a `news` app.
- **Views**: Function-based views for the front end and class-based API views for the back end.
- **APIs**: Use Django REST Framework for the API endpoints and `requests_oauthlib` for X API integration.
- **Signals**: The `post_save` signal on the `Article` and `Newsletter` models handles email and X API dissemination.

7. API Design

The RESTful API will provide a simple and clear way for third-party clients to get data.

- **GET `/api/v1/articles/`**: Retrieves articles for a subscribed user.
- **GET `/api/v1/newsletters/`**: Retrieves newsletters for a subscribed user.
- **GET `/api/v1/articles/<int:pk>/`**: Retrieves a single article by its ID.
- **GET `/api/v1/newsletters/int:pk/**: Retrieves a single newsletter by its ID.

8. Development Workflow

1. **Project Setup**: Initialize the Django project and `news` app.
2. **Model Creation**: Define the models and run migrations.
3. **Authentication**: Implement the custom user model with roles and permissions.
4. **Front-End Design**: Create all HTML templates and style them with Tailwind CSS.
5. **Dissemination Logic**: Implement the email and X API integration using signals.
6. **RESTful API**: Create serializers, API views, and URLs.

7. **Unit Testing**: Write unit tests to verify the API and signal handlers.
8. **Deployment**: Migrate the database to MariaDB.