Post Generator

A full-stack application for generating and managing posts, leveraging modern JavaScript technologies for both frontend and backend development.

Table of Contents

Overview

Post Generator enables users to generate, manage, and interact with posts via a modern web interface. It combines a React + Vite frontend with a Node.js/Express backend, and connects to a MongoDB database for persistent storage.

Architecture

```
post-generator/
├── backend/      # Node.js/Express server and API
├── Frontend/     # React + Vite frontend application
├── package.json  # Project metadata and scripts
```

Backend

- RESTful API built with Express.js
- MongoDB integration via Mongoose
- Authentication via JWT
- Modular routing and middleware for scalability

Frontend

- Built with React (Vite-powered)

- Modern SPA architecture
- ESLint-configured for code quality

Features

- User authentication (JWT)
- Post creation, editing, deletion
- Modern, responsive UI
- API endpoints for CRUD operations
- Environment-based configuration
- Modular and scalable structure

Tech Stack

- **Frontend:** React, Vite, ESLint
- **Backend:** Node.js, Express, Mongoose
- **Database:** MongoDB
- **Testing/Dev Tools:** Nodemon, Dotenv, ESLint
- **Other:** Axios, JWT, Groq SDK, Passport, OAuth

Directory Structure

```
backend/
├── Routers/        # API route handlers
├── middlewares/    # Express middlewares
├── config.js       # Configuration variables
├── db.js           # Database connection setup
├── index.js        # Main server entry point

Frontend/
├── public/         # Static assets
├── src/            # React components and logic
├── index.html      # Main HTML file
├── package.json    # Frontend dependencies
```

Installation

Prerequisites

- Node.js (>= 18.x)
- npm or yarn
- MongoDB (local or remote instance)

Clone the Repository

```
git clone https://github.com/TejaBudumuru3/post-generator.git
cd post-generator
```

Install Dependencies

Root dependencies:

```
npm install
```

Backend dependencies:

```
cd backend
npm install
```

Frontend dependencies:

```
cd ../Frontend
npm install
```

Usage

Running Backend

1. Configure your `.env` file in the `backend/` directory (see `.env.example` if available).
2. Start the backend server:

   ```
   npm run start
   ```

   or

   ```
   nodemon backend/index.js
   ```

Running Frontend

1. From the `Frontend/` directory:

   ```
   npm run dev
   ```

2. Visit the local address provided (usually `http://localhost:5173/`).

API Reference & Detailed Functionality

Functionality Overview

Post Generator is a web application that allows users to:

- Register and authenticate securely.
- Generate multiple social media posts/tweets on a given topic in various tones using a generative AI backend (Groq SDK).
- View generated posts in a user-friendly, carousel-style UI.
- Manage sessions (login/logout) with JWT authentication.
- Interact via a responsive React frontend communicating with an Express REST API backend.

The backend uses real-time AI content generation, pulling data from the internet and generating tweets/posts that are fact-focused, transparent, and customizable in tone. The application is designed for users who want to create impactful, data-driven social media content efficiently.

API Endpoints

All backend routes are prefixed with `/user`.

Authentication & User Management

POST `/user/signup`

Registers a new user.

Request Body:

```json
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123",
  "fname": "John",
  "lname": "Doe"
}
```

Response:

- `201 Created` on success
- `409 Conflict` if user/email exists

POST `/user/login`

Authenticates a user and sets a JWT token cookie.

Request Body:

```
{
  "email": "john@example.com",
  "password": "password123"
}
```

Response:

- `200 OK` with user info and token in cookies
- `401 Unauthorized` on invalid credentials

DELETE `/user/logout`

Logs out the user by clearing the JWT token cookie.

Response:

- `200 OK` and confirmation message

Post Generation

POST `/user/GenerateData`

Generates five tweets/posts on a given topic, in a specified tone.

Authentication: Requires JWT token cookie.

Request Body:

```
{
  "question": "climate change policy",
  "tone": "sympathetic" // Optional, defaults to "neutral"
}
```

Response:

```
{
  "ans": "Tweet1~Tweet2~Tweet3~Tweet4~Tweet5",
  "message": "Data generated successfully"
}
```

- Each tweet is separated by the `~` character.
- Tweets are context-aware, fact-based, and generated by Groq SDK with web data.

User Details

GET `/user/getDetails`

Fetches user profile data (requires authentication).

Response:

```
{
  "data": {
    "_id": "...",
    "name": "...",
    "email": "...",
    // other user fields
  }
}
```

Authentication Middleware

All protected endpoints use a JWT token stored in cookies:

```
const token = req.cookies.token;
const Decoded = jwt.verify(token, JWT_SECRET);
// Checks existence and validity, attaches user info to req.user
```

If the token is missing or invalid, endpoints return `401 Unauthorized`.

Example Usage (Frontend)

- Upon signup/login, the user receives a JWT token.
- The React frontend calls `/user/GenerateData` with the desired topic and tone.
- The backend returns five tweets, which are split and displayed as a carousel.
- Logout clears the session.

AI Content Generation Logic

- Uses the Groq SDK with custom system prompts.
- Searches web data for the latest, verified information.
- Produces tweets that are concise, impactful, and transparent, exposing truths and promoting awareness.
- Each tweet includes hashtags and may use emojis.

Typical User Flow

1. User signs up or logs in.
2. User inputs a topic/question and selects a tone.
3. Frontend sends a POST request to `/user/GenerateData`.
4. Backend authenticates the user, queries the AI, and returns five tweets.
5. Frontend displays the posts in a carousel.
6. User can log out at any time.

Error Handling

- All endpoints return descriptive error messages and appropriate HTTP status codes.
- Missing or invalid tokens result in `401 Unauthorized`.
- Missing required fields result in `400 Bad Request`.

Development

- **Lint code:** `npm run lint` (Frontend)
- **Build frontend:** `npm run build`
- **Preview frontend:** `npm run preview`
- **Testing:** (Add tests as needed)

Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature-name`)
3. Commit your changes (`git commit -am 'Add new feature'`)
4. Push to the branch (`git push origin feature-name`)
5. Create a Pull Request

License

This project is licensed under the ISC License. See the LICENSE file for details.

Authors & Acknowledgments

- Developed by TejaBudumuru3
- Built with open source technologies

Contact

For issues or feature requests, please use the GitHub Issues page.
