Blog Application Backend Documentation

By: Paras Singhal
Parassinghal1@gmail.com
Github:parasgroot2005

Content:

Introduction:

Introducing my blog app backend powered by Node.js and Express.js. With robust data validation and error handlers, it offers a seamless blogging experience. Create, share, and connect with ease, while ensuring the security and accuracy of your content. Join us in the world of blogging made simple.Introducing my blog app powered by Node.js and Express.js. With robust data validation, it offers a seamless blogging experience. Create, share, and connect with ease, while ensuring the security and accuracy of your content. Join us in the world of blogging made simple.

Project Initiation:

1. After Cloning the repo, run “npm install”.
2. Run “npm run dev” to start the server.
3. Open postman to test the Api’s.

Tech Stack Used:

1. Node.js: A JavaScript runtime environment that allows running JavaScript code outside the web browser.
2. Express.js: A web application framework for Node.js, used to build the backend of the application.
3. TypeScript: A typed superset of JavaScript that adds static typing and enhances tooling support.
4. MongoDB: A popular NoSQL database used for storing and retrieving data efficiently.
5. Celebrate: A middleware library used for data validation in Node.js and Express.js applications.
6. RESTful API: A design pattern used to create APIs that follow the principles of Representational State Transfer (REST), enabling CRUD operations (Create, Read, Update, Delete) over HTTP.
7. HTTP requests: Used to facilitate communication between the frontend and backend, enabling data transfer and interaction.

How To Use The API's;

➔ Create a Blog:

- ◆ Method: POST
- ◆ Endpoint: /posts
    - Request body:
    - title (required): The title of the blog (string).
    - author (required): The author of the blog (string).
    - content (required): The content of the blog (string).
    - createdAt (optional): The creation date of the blog (date).
    - updatedAt (optional): The last update date of the blog (date).
    - tags (optional): An array of tags associated with the blog (array of strings).
    - isPublished (required): The publication status of the blog (boolean).
- ◆ Response: The created blog object.

➔ Get all Blogs:

- ◆ Method: GET
- ◆ Endpoint: /posts
- ◆ Response: An array of all blog objects.

➔ Get a specific Blog by ID:

- ◆ Method: GET
- ◆ Endpoint: /posts/:id
    - Path parameter:
    - id: The ID of the blog to retrieve.
- ◆ Response: The blog object with the specified ID.

➔ Update a Blog by ID:

- ◆ Method: PUT
- ◆ Endpoint: /posts/:id
  - Path parameter:
  - id: The ID of the blog to update.
- ◆ Request body: The updated blog object with the fields to modify.
- ◆ Response: The updated blog object.

➔ Increment the View Count of a Blog by ID:

- ◆ Method: PATCH
- ◆ Endpoint: /posts/count/:id
  - Path parameter:
  - id: The ID of the blog to increment the view count.
- ◆ Response: The blog object with the incremented view count.

➔ Delete a Blog by ID:

- ◆ Method: DELETE
- ◆ Endpoint: /posts/:id
  - Path parameter:
  - id: The ID of the blog to delete.
- ◆ Response: A success message indicating the deletion was successful.

Testing The API's:

➔ Launch Postman: Open Postman on your computer.
➔ Create a new request: Click on the "New" button in the top-left corner of the Postman window to create a new request.

➔ Set the request method and URL:
- *For creating a blog:*
- Method: POST
- URL: Enter the URL for creating a blog, e.g., http://localhost:PORT/posts/
- Add the required data inside the body in json format.

- *For getting all blogs:*
- Method: GET
- URL: Enter the URL for getting all blogs, e.g., http://localhost:PORT/posts.

- *For getting a specific blog by ID:*
- Method: GET
- URL: Enter the URL for getting a specific blog by ID, e.g., http://localhost:PORT/posts/POST_ID.
- Replace BLOG_ID with the actual ID of the blog you want to retrieve.

- *For updating a blog by ID:*
- Method: PUT
- URL: Enter the URL for updating a blog by ID, e.g., http://localhost:PORT/posts/POST_ID.
- Replace BLOG_ID with the actual ID of the blog you want to update.
- Need to update the whole body, so need to put every value of every required object.

- *For incrementing the view count of a blog by ID:*
- Method: PATCH
- URL: Enter the URL for incrementing the view count of a blog by ID, e.g., http://localhost:PORT/posts/count/POST_ID.
- Replace BLOG_ID with the actual ID of the blog you want to update the view count for.
- Every time we hit this api

- *For deleting a blog by ID:*
- Method: DELETE
- URL: Enter the URL for deleting a blog by ID, e.g., http://localhost:PORT/posts/POST_ID.
- Replace BLOG_ID with the actual ID of the blog you want to delete.

➔ Set request headers (if necessary): Depending on your API setup, you may need to include specific headers, such as "Content-Type: application/json". Click on the "Headers" tab below the URL field and enter the required headers.

➔ Set request body (if necessary): For the "create" and "update" endpoints, you need to provide a request body in JSON format. Click on the "Body" tab, select "raw", choose JSON from the dropdown, and enter the necessary data based on the API documentation.

- ➔ Send the request: Click on the "Send" button to send the request to the API.
- ➔ View the response: Postman will display the response in the "Response" pane. You can inspect the status code, response body, and headers returned by the API.
- ➔ Test different scenarios: Repeat steps 2 to 7 for different endpoints, adjusting the request method, URL, headers, and request body as required. Test various scenarios by modifying the request parameters or data to cover different use cases.

Remember to replace PORT(in this case 3000) with the actual port number your server is running on and POST_ID with the appropriate post ID for the specific endpoints. Refer to the API documentation for more details on the expected request formats and the responses for each endpoint.

EXTRAS:

In addition to the core CRUD operations, I have added some extra functionalities to enhance these blog API:

- HTTP Requests(that takes authentication token before accessing the api): Integrated HTTP requests into your API, making it easier to interact with the backend from the frontend. This enables seamless communication between the client-side and server-side applications, allowing users to create, retrieve, update, and delete blogs using standard HTTP methods.
- View Counter API: Implemented a view counter feature that allows you to track the number of views a particular blog has received. This functionality provides insights into the popularity and engagement of your blog posts. By including a separate endpoint for incrementing the view count of a blog by ID, you can keep track of how many people have viewed a specific blog.

With these added features, we can gain a deeper understanding of our blog's performance and user engagement. The HTTP requests facilitate easy integration with your frontend application, enabling users to interact with our API seamlessly. The view counter API provides valuable metrics that can help us analyze the success and reach of our blog posts.