Backend: blog-service

Implement the PostController

File to be edited: postController.js in controllers folder

1. Implement the `updatePost` Function

- **Objective**: Allow authorized users to update a post.
- **Steps**:
    1. Retrieve the `tags`, `categories`, `title`, and `content` from `req.body`.
    2. Use `Post.findById(req.params.id)` to find the post by ID from the database.
        - If the post is not found, return a `404 Not Found` response with an appropriate message.
    3. Check if the current user (`req.user.id`) matches the post's author:
        - If they do not match, return a `403 Forbidden` response with an appropriate message.
    4. Prepare an `updatedData` object:
        - Add `title` and `content` to the object if provided.
        - If `tags` are provided:
            - Call `createOrGetTags` to retrieve or create the associated tag IDs.
            - Add the tag IDs to `updatedData`.
        - If `categories` are provided:
            - Call `createOrGetCategories` to retrieve or create the associated category IDs.
            - Add the category IDs to `updatedData`.
    5. Use `Post.findByIdAndUpdate` to update the post with the `updatedData` object and return the updated post.
    6. Respond with a success message and the updated post.
    7. Use a `try...catch` block to handle errors and return a `500 Internal Server Error` response in case of failures.

---

2. Implement the `deletePost` Function

- **Objective**: Allow authorized users to delete a post and its associated data.
- **Steps**:

1. Use `Post.findById(req.params.id)` to find the post by ID from the database.
   - If the post is not found, return a `404 Not Found` response with an appropriate message.
2. Check if the current user (`req.user.id`) matches the post's author:
   - If they do not match, return a `403 Forbidden` response with an appropriate message.
3. Remove associated likes and comments:
   - Use `Like.deleteMany` to delete all likes for the post.
   - Use `Comment.deleteMany` to delete all comments for the post.
4. Delete the post itself:
   - Use `post.deleteOne()` to remove the post from the database.
5. Clean up unused tags and categories:
   - Call `cleanUpTags` to remove tags that are no longer associated with any posts.
   - Call `cleanUpCategories` to remove categories that are no longer associated with any posts.
6. Respond with a success message indicating the post and its associated data were deleted successfully.
7. Use a `try...catch` block to handle errors and return a `500 Internal Server Error` response in case of failures.

Implement the Posts Router

File to be edited: postRoutesRouter.js in routes folder

1. Import functions implemented in postController.js that are associated with update and delete post functionalities
2. Add routes for Update and Delete functionalities

Frontend: Post Functionalities

Delete Functionality

File to be edited for Delete Functionality: BlogPost.jsx

This guide explains the implementation process for the `handleDelete` function, which deletes a post and includes authorization checks and error handling.

---

1. Retrieve User Authentication Information

- **Objective**: Access the currently logged-in user's authentication details.
- **Steps**:
    1. Use `localStorage.getItem("auth_user")` to retrieve the stored user information.
    2. Parse the JSON string into an object using `JSON.parse()`.
    3. Extract the `token` for authentication and `currentUserId` to identify the user.

---

2. Check for User Authentication

- **Objective**: Ensure the user is logged in before proceeding.
- **Steps**:
    1. Verify that the `token` exists.
    2. If the token is missing, display an alert message:
       "Authentication token not found. Please log in."
    3. Terminate further execution by returning early.

---

3. Verify User Authorization

- **Objective**: Allow only the author of the post to delete it.
- **Steps**:
    1. Compare the `author` of the post with `currentUserId`.
    2. If they do not match, display an alert message:
       "You are restricted to delete this post, as you are not the owner."
    3. Prevent unauthorized deletion by returning early.

---

4. Send DELETE Request

- **Objective**: Communicate with the API to delete the post.
- **Steps**:
    1. Construct the API URL using the post ID:
       `${import.meta.env.VITE_API_URL}/api/posts/${id}`.
    2. Use the `fetch` API to send a `DELETE` request:
        - Include the `Authorization` header with the token.
        - Specify the HTTP method as `DELETE`.
    3. Await the response from the server.

---

5. Handle API Response

- **Objective**: Check if the post was deleted successfully.
- **Steps**:
    1. Use `response.ok` to verify the response status.
    2. If the response is not successful:
        - Throw a new error with the message:
          "Failed to delete the post. Please try again."
    3. Display an alert message on success:
       "Post deleted successfully!"

---

6. Redirect to Home Page

- **Objective**: Navigate to the home page after the post is deleted.
- **Steps**:
    1. Use the `navigate("/")` function to redirect the user to the home page.
    2. Ensure this step occurs only after successful deletion.

---

7. Handle Errors

- **Objective**: Gracefully handle any errors during the process.
- **Steps**:
    1. Use a `try...catch` block to wrap the logic.
    2. In the `catch` block:
        - Display the error message using `alert()`.
        - Log the error for debugging purposes, if necessary.

Update Functionality

File to be edited for update Functionality: PostEditor.jsx

1: Check the condition where `!formData.id` is validating

1. Remove this condition so that update functionality can also be implemented here

---

2: Identify Where the `apiUrl` and method are Defined

1. Locate the part of the `handleSubmit` function where the `apiUrl` and `method` variables are being defined.

3: Update the `apiUrl` Variable

1. Modify the `apiUrl` to dynamically check if `formData.id` exists.
2. If `formData.id` is truthy:
    - Set the `apiUrl` to include the `formData.id` in the endpoint.
3. If `formData.id` is falsy:
    - Set the `apiUrl` to the base URL for creating a new post.

4: Update the `method` Variable

1. Check if `formData.id` exists.
2. If `formData.id` is truthy:
    - Set the `method` to `"PUT"`.
3. If `formData.id` is falsy:
    - Set the `method` to `"POST"`
