Steps to Start Writing the Code from Scratch

Here's how you can systematically approach building your Social Media Post Management System with the JavaScript functionality you've outlined.

---

1. Understand the Core Requirements

- **Post Management**: Add, edit, like, comment, and search posts.
- **Rendering Posts**: Dynamically update the UI to display posts and their interactions.
- **Input Handling**: Throttle search input and handle post and comment inputs.
- **Edit History**: Maintain a history of edits for each post.

---

2. Plan the Code Structure

Divide your functionality into these core modules:

- **Post Management**: Handle adding, editing, liking, commenting, and searching posts.
- **Rendering Functions**: Update the DOM to display posts dynamically.
- **Utility Functions**: Implement throttling to optimize user input handling.
- **Event Handlers**: Connect UI events (button clicks, input changes) to post management logic.

---

3. Start with Initialization

- Create the skeleton structure of your JavaScript file.
- Define the `createPostManager` function to manage post operations.

---

4. Implement Post Management

- Use closures in `createPostManager` to encapsulate the posts array.
- Write methods to:
    - **Add Posts**: Create posts with unique IDs, likes, comments, and edit history.
    - **Edit Posts**: Update post content and save previous content to the edit history.
    - **Like Posts**: Increment the likes count for a post.
    - **Add Comments**: Append comments to a post.
    - **Search Posts**: Filter posts based on a keyword.

---

5. Render Posts

- Write the `renderPosts` function to dynamically update the DOM.
- Display post content, likes, comments, and buttons for actions (like, edit, comment).
- Create reusable templates for posts and comments.

---

6. Handle Input Events

- Implement a `throttle` utility function to handle frequent search input events efficiently.
- Create event listeners for:
    - Adding new posts.
    - Searching posts.
    - Liking and commenting on posts.
    - Editing posts via a prompt.

---

7. Implement Throttling for Search

- Write the `throttle` function to limit the frequency of search operations.
- Attach it to the search input field to handle `input` events efficiently.

---

8. Test and Debug

- Test each feature individually:
    - Add, edit, like, and comment on posts.
    - Search posts and ensure results are updated correctly.
    - Check the functionality of edit history.
- Debug any issues using `console.log` and browser dev tools.

---

9. Optimize Code

- Ensure your code is:
    - Modular: Separate logic into functions.
    - Maintainable: Add comments and meaningful variable names.
    - Performant: Optimize DOM updates and event handling.

Suggested Order to Write the Code

1. **Post Manager Initialization (`createPostManager`)**
2. **Rendering Functions (`renderPosts`)**
3. **Utility Functions (`throttle`)**
4. **Event Listeners (e.g., for adding posts, search)**
5. **Testing and Debugging**

Tools to Assist

- **Console Logs**: Debug outputs at critical points.
- **Browser DevTools**: Inspect DOM updates and debug JavaScript logic.

With this structured approach, you'll have a clean, modular, and maintainable JavaScript codebase for your Post Management System. Let me know if you need help with any specific part!