PUNITH KUMAR B – Contribution for the Project

GROUP 7

Project Title: Newsgram: Infinite Posts Feed

Git hub Repo of the project: https://github.com/BOSS-009/Newsgram

Role: Front-End Developer, UI/UX Implementation

Responsibilities Overview: As a front-end developer for the Newsgram project, I was responsible for designing, implementing, and integrating the core user interface and interactive features of the news feed. My work was focused on creating a dynamic and user-friendly experience, including post display, like/dislike functionality, post sorting, and integration with the infinite scroll feature.

Modules and Components

1. PostCard.jsx I developed the PostCard.jsx component to represent a single blog post. This component is responsible for displaying all post-related information and handling user interactions.

- **Like & Dislike Functionality:** I implemented the useState hook to manage the like and dislike counts. The logic ensures that a user can only have one active reaction (either a like or a dislike) at a time.
- **Conditional Rendering:** The component dynamically changes the appearance of the like and dislike buttons based on the user's selection, providing clear visual feedback.
- **Metadata:** I ensured the card correctly displays the post's title, body snippet, author, and creation date.

File: /components/PostCard.jsx

2. PostList.jsx I worked on the PostList.jsx component, which manages the display of all posts and provides sorting options.

- **Dynamic Sorting:** I implemented a state-driven sorting mechanism with three distinct options: **Recent**, **Liked**, and **Alphabetical**. This allows users to easily organize the content based on their preference.

- **UI/UX:** I added conditional styling to the sorting buttons to highlight the active sort order. The component also handles the rendering of the LoadingSkeleton and the "end of posts" message to improve the user experience.

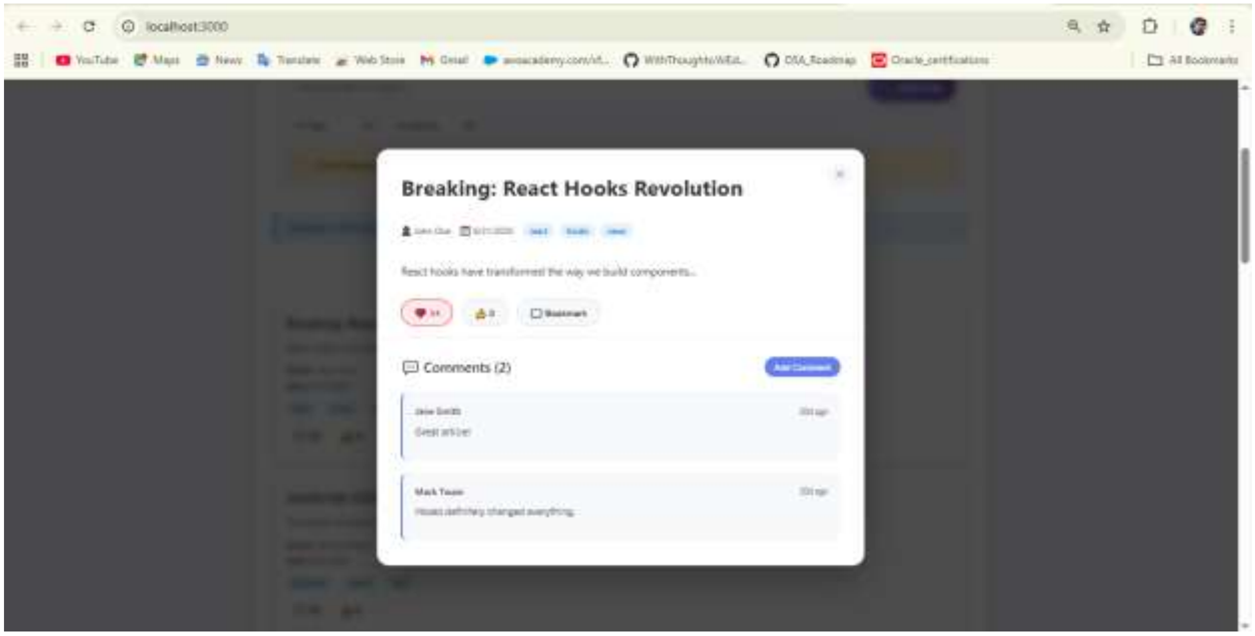
File: /components/PostList.jsx

3. App.js I integrated the new features and components into the main App.js file, centralizing the data flow and logic.

- **Data Integration:** I modified the post-fetching logic to include an initial, randomly generated number of likes and dislikes for each post, making the feed feel more realistic and interactive upon first load.
- **Component Composition:** I ensured that the fetched data and hooks (useFetchPosts and useInfiniteScroll) were correctly passed down as props to the PostList component.

File: /App.js

SCREENSHOTS


Breaking: React Hooks Revolution

React hooks have transformed the way we build components...

Comments (2)

Jane Smith

Great article!

Mark Twain

Hooks definitely changed everything.

Fig 1: Full Post View with Like functionality

This screenshot demonstrates the Like functionality within the Post Modal. When a user opens a post, a modal appears displaying the full content and interactive buttons. The like button, featuring a heart icon, is clearly visible with a count of 34, which showcases the core interactive functionality that I developed.

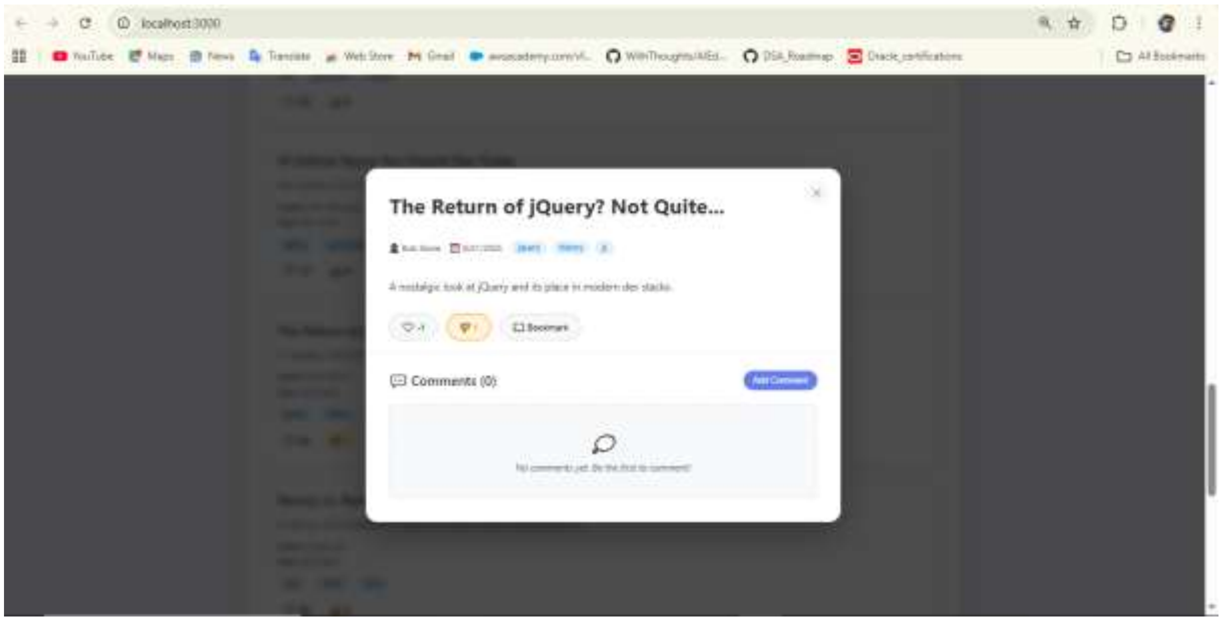

localhost:3000
The Return of jQuery? Not Quite...
A nostalgic look at jQuery and its place in modern dev stacks.
-1
Bookmark
Comments (0)
Add Comment
No comments yet. Be the first to comment!

Fig 2: Full Post View with Dislike Functionality

This image shows the post modal, which appears when a user clicks on a post. It clearly showcases the like, dislike, and bookmark buttons. The dislike button is visibly active, with a -1 count, demonstrating the functionality I implemented.

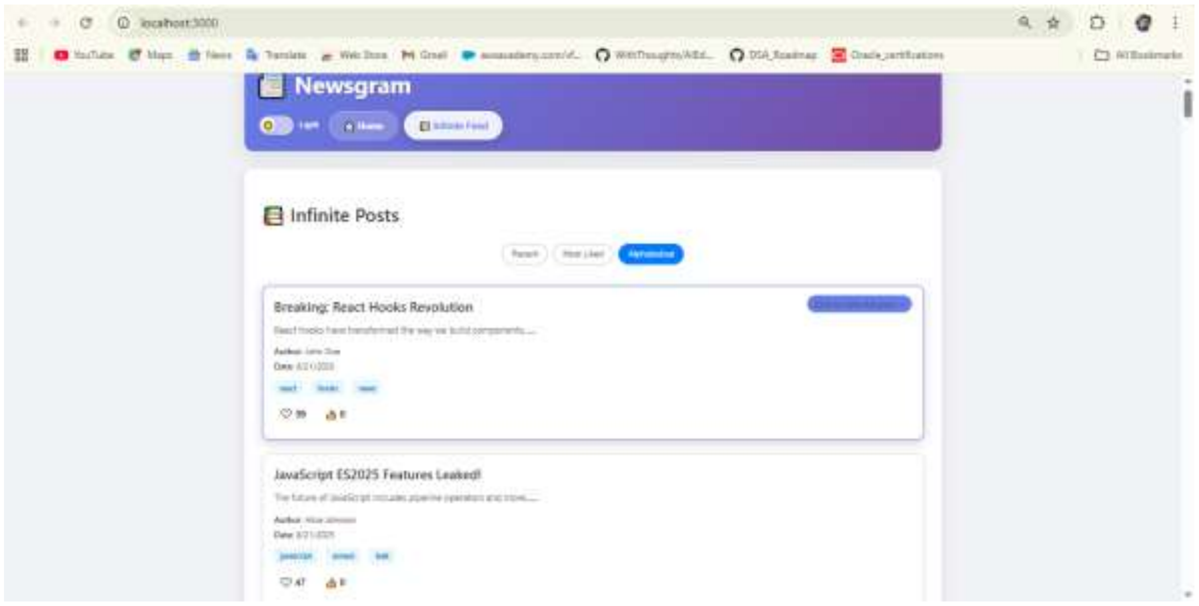
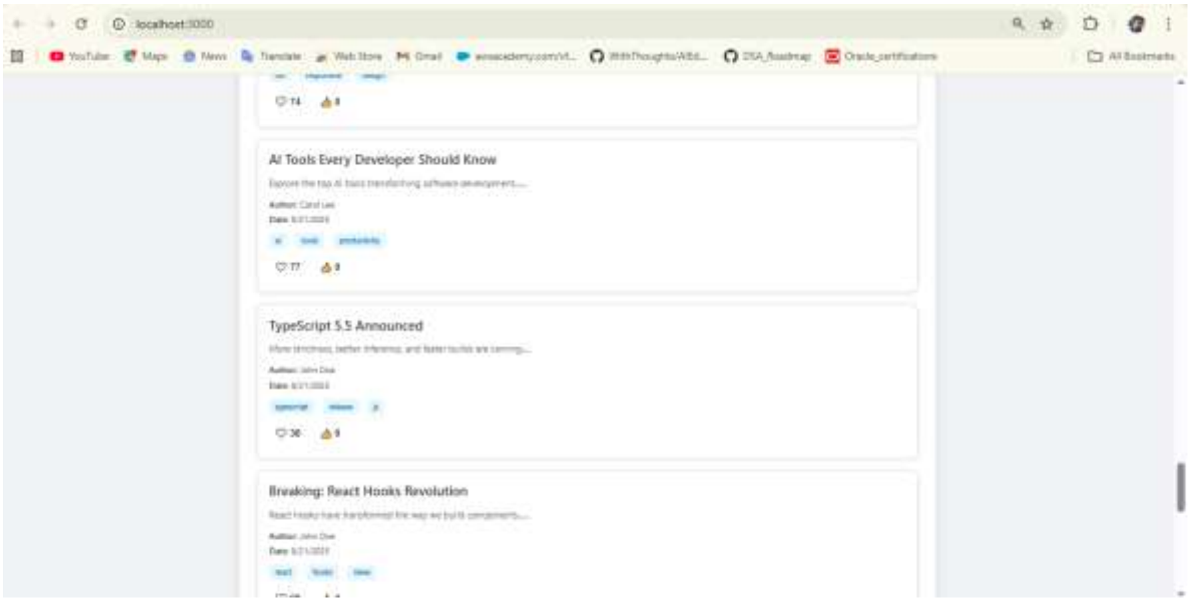


Fig 3: Post Sorting Functionality

This screenshot demonstrates the dynamic sorting feature. The "Alphabetical" button is highlighted, showing that the posts are currently sorted by their title, ensuring a clean and organized user interface.



AI Tools Every Developer Should Know

Explore the top AI tools transforming software development....

Author: Carol Lee

Date: [illegible]

TypeScript 5.5 Announced

More strictness, better inference, and faster builds are coming...

Author: John Doe

Date: [illegible]

Breaking: React Hooks Revolution

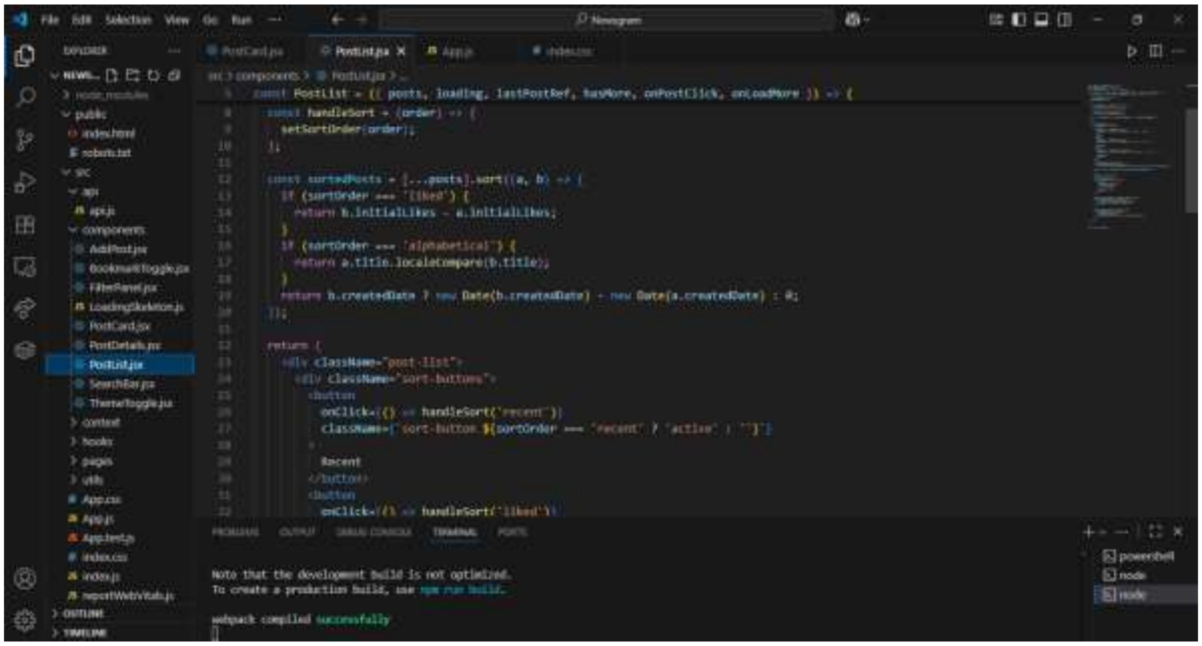
React Hooks have transformed the way we build components....

Author: John Doe

Date: [illegible]

Fig 4: The Infinite Scroll Feature

This screenshot captures the main feed as the user scrolls down. The continuous list of posts demonstrates the infinite scroll functionality, where new content is loaded seamlessly as the user reaches the bottom of the page.



```
const PostList = ({ posts, loading, lastPostRef, hasMore, onPostClick, onLoadMore }) => {
  const handleSort = (order) => {
    setSortOrder(order);
  };

  const sortedPosts = [...posts].sort((a, b) => {
    if (sortOrder === 'liked') {
      return b.initialLikes - a.initialLikes;
    }
    if (sortOrder === 'alphabetical') {
      return a.title.localeCompare(b.title);
    }
    return b.createdDate ? new Date(b.createdDate) - new Date(a.createdDate) : 0;
  });

  return (
    <div className="post-list">
      <div className="sort-buttons">
        <button
          onClick={() => handleSort('recent')}
          className={`sort-button ${sortOrder === 'recent' ? 'active' : ''}`}
        >
          Recent
        </button>
        <button
          onClick={() => handleSort('liked')}
```

```
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Fig 5 : Post List Component Code

This code snippet from the PostList.jsx file shows the initial state management for the posts, including the sorting order, loading state, and error handling. It highlights my work in managing the local component state.

Development Setup

To run and test the application locally, follow these steps:

1. Install dependencies: npm install
2. Run the React App: npm start

You may need to run the mock API server as a separate step depending on the project configuration. For a standard json-server setup, the command is:

```
npx json-server-watch db.json --port 5000
```