Abdulrehman Ismaeel

Design Report

The backend is built with Node.js and Express. I used RESTful APIs to handle users, posts, and replies. I chose MySQL because it's reliable and relational, which made it easier to manage relationships between users, posts, and nested replies.
Main tables:

- users: Stores usernames, hashed passwords, and roles (either regular user or admin).
- posts: Represents the main content/questions. Each post has a title, content, a uid for the author, and timestamps.
- replies: Supports threaded discussions. Each reply belongs to a post and can also have a parent_reply_id, allowing for nested replies.
- Instead of a separate votes table, I kept things simple by adding thumbs_up and thumbs_down counters directly inside the replies table.

I also implemented an "accepted answer" feature. In the replies table, I added an is_accepted field that gets toggled when the post author (or an admin) marks a reply as the solution. This helps highlight helpful answers and adds structure to long discussions.
Admins can delete any post or reply. I used middleware to check for admin access before allowing delete requests, so regular users can't abuse the system.

Pages and components:

- Homepage: Shows a list of recent posts with clickable links to each.
- PostView: Shows a single post and all related replies (rendered recursively for nesting).
- ReplyForm: Lets users reply to posts or other replies.
- SearchPage: Supports keyword search, user filters, and shows top contributors based on post/reply activity.
- User Badge: Based on post/reply stats, users are shown as Newbie, Contributor, or Expert.

I made sure the UI was responsive and intuitive. On most pages, the navbar only shows a search icon to keep things clean. When clicked, it takes you to the full search page with filters.
The badge system adds a bit of fun and community to the app, encouraging users to engage more. Badges are determined on the client side by checking how many posts/replies a user has made and how well their replies are rated.