Project Design report

I designed a 3 layer chat forum using mySQL as a database server connected to a NodeJS server for the backend and a NodeJS server for React for the front end.

Database:

I created a mySQL database “postdb” with tables for Users, Channels, Posts, Replies, and Vote tracking for posts and replies (stored separately). I built this into a function in server.js (61-116)

Users houses: id, uName, uPass, isMod. Id is the primary key and auto increments. uName and uPass are stored in clear text and up to 64 characters. A moderator account is hard coded in the setup. There were no provisions made to create other moderators through the app, they can be modified in the database by changing isMod from 0(default for new users) to 1.

Channel houses: id, channelName, description. Id is the primary key and auto increments. Channels are removable by mods, but the posts and replies inside the channel are not effected by the deletion of a channel.

Posts houses: id, timeStamp, topic, data, VotePost, uID, uName, channelID. Primary key is id. TimeStamp records the time of creation the post. Topic and data are the title and body of the post, VotePost is the current number of thumbs up/down (starts at 0, the user can up or down vote their own posts), uID and uName manage whether the post can be deleted (only by Mods and post owners), and a channelID to organize similar posts. If a post is deleted, the body of the post is modified to read: ‘Post removed by: <user>’ where <user> is the name of either the moderator or the owner of the post.

Reply houses: id, timestamp, data, parentID, VoteReply, uID and uName. The id, timestamp, data, uID and uName are all similarly implemented as they were in the posts table. VoteReply is the reply equivalent of VotePost, and parentID links the reply to the post is attached to. Replies can be deleted, and have a similar output with the name of the user or moderator listed as to who removed the data.

VotePost and VoteReply house: id, timestamp, parentID, uID, and direction. Id for each vote is unique, but can be updated by clicking on the same direction as previously voted to move the vote back to a neutral position, or clicking in the opposite direction to change the vote from positive to negative or negative to positive. Id, timestamp, parentID and uID all have similar implementations to previous tables.

I designed a landing page with a check for a jwt for authentication. If a database is not detected a prompt to initialize the database will appear. If there is a database available, a login screen requests a user name and password, with a link to create a user. All users created through React are non-moderators.

After login a navigation bar loads at the top of the page and the default view of the Channels page appears, with all current channels listed in the order they were created. At the bottom of the page users can create new channels. Moderators can remove channels from this page.

After selecting (or creating and then selecting a channel) the user is navigated to the posts in that channel. As with the channels view, users can view the posts in that channel, but they can also vote on a post, reply to a post, see the best reply for each post and the number of replies. The user can navigate to see all the replies for a post or add a new post in the same channel. Navigation to the channels is through a link on the navigation bar. Moderators and post owners can remove posts from this page.

When viewing all replies for a post, you can see all of the replies listed below their related post, on this page replies can be added and voted on. Moderators and reply owners can remove replies from this page.

On the Navigation bar, links to the Channels/default page, a search Modal and a Sign Out button are available. The Search Modal allows you to search for text(excludes user names), users, and the best and worst posts and replies.

Database queries are all handled in the backend with requests sent from the front end to exposed routes.