

Design Report (ido030)




Hand-drawn diagram of the Users table. The table has four columns: id(unique), Username, Password, and authority level.

id(unique)	Username	Password	authority level
------------	----------	----------	-----------------

The decision was to make four fields, the unique id, username, password and authoritylevel.

Authoritylevel is meant to differentiate users from admins and other levels so as to assign privileges based on authority. All users have authority level 0 and the admin has authoritylevel 1



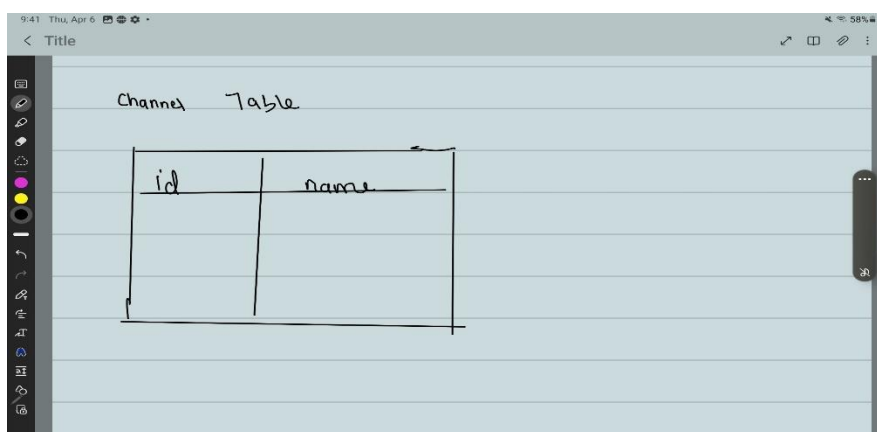
Hand-drawn diagram of the Posts Table. The table has six columns: id, channelname, Data, username, UPvotes, and downvotes.

id	channelname	Data	username	UPvotes	downvotes
----	-------------	------	----------	---------	-----------

Post have these fields as needed,

Every message has upvote(like) or dislike count

We store the channel it was created in channelname so that we can render messages specific to a channel



Hand-drawn diagram of the Channel Table. The table has two columns: id and name.

id	name
----	------

Channel only had id and name.

I decided it would be easier and more efficient (for a small system) to not store messages in individual channel tables, rather have a field in posts and replies table, such that we can filter using sql queries

Also, this would make my search functionality more straightforward as I do not need to JOIN multiple channel tables to search across the system, they are all either in posts or replies and I can decide to pull all their data or filter based on need.

9:40 Thu, Apr 6 58%

< Title

Replies Table

id	PostID	data	Username	upvotes	down votes	Channelname

In this table we have a bunch of things in common with posts utilizing the same logic.

The main difference is the foreign key, the postid

Every comment must have this as it must be mapped to a message and displayed as such.

I also reused this for my logic for nested replies, the original reply would be the postid for a nested reply and I would be able to display the nested reply this way.

It's pretty straightforward and with the right implementation on the frontend, it would work as planned

In react, I would use react-router-dom to connect the pages as routes and navigate.

I would have 6 major pages: Landing page, register, signin, chat, admin, search.

All of these pages would be essential for the basic functionality. The chat page would be for users to communicate within the system, the admin page for admins to do their admin roles while also monitoring the messages in the system.

The search page would be for searching

I would divide my react files into components, pages, stylings.

For my server, it would connect to the DB and handle all the requests for data storage and retrieval from the frontend.