1.File structure and functionality:

File directory structure: see last of this report.

Functionalities:

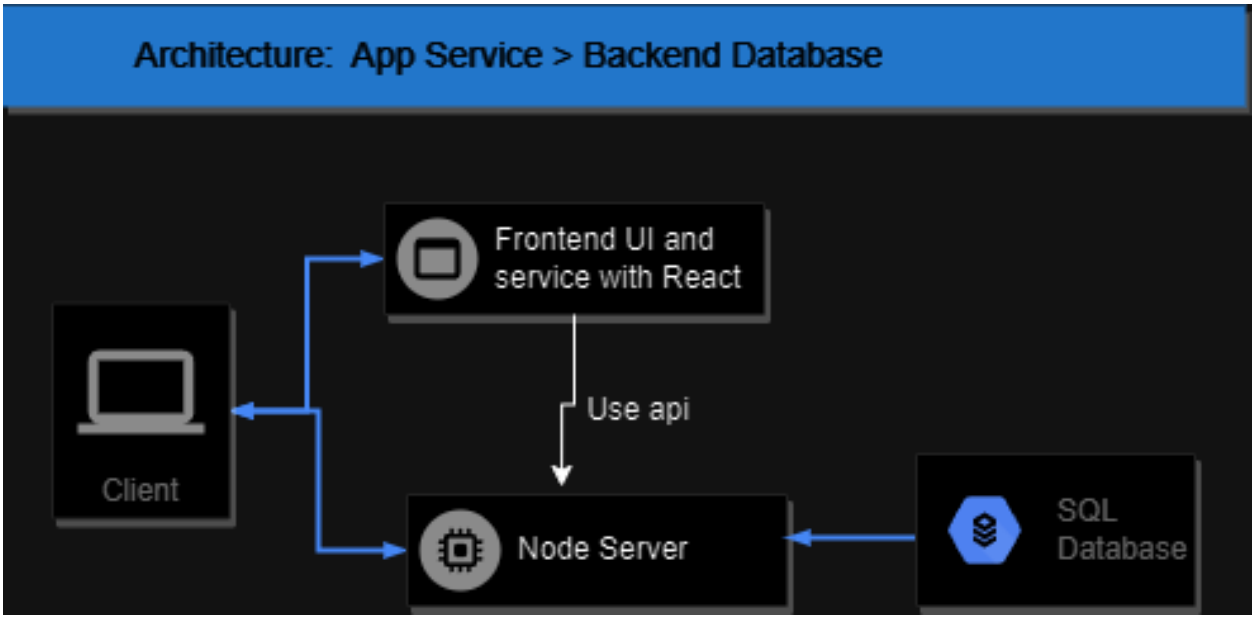
I have implemented all part1–5 in the requirements, to clarify, I have allowed 2 layers of reply, and part5 I chose to let every user have a rank , from beginner to medium to expert, based on the number of posts they've created(0–4 posts beginner, 5 posts medium, over 5 posts expert).

2.Architecture

Class Diagram:(**<u>at last part of the report,</u>** before the file structures.)

As the diagram shows, I have implemented the replies to allow 2 layers, so any user can reply to a comment of a post, and the number of replies on nested layer is not limited. But it does not show user info on 2nd layer reply, so cannot tell which user replied(a limitition in this project).

Basic Architecture diagram:



3. Decision taken – Database structure:

I have designed the database to contain 4 tables: Users, Channels, Posts and Comments, and to retrieve content/user everything should be related with matching information.

I'll explain how I designed the relation for the tables in database:

* In User DB, every user has anauto–incremented id when register an account.

When a user create content, the user id is saved with content data:

for example when user submit a post, it'll insert Userid and post content into the Posts table, so user info and the post are related.

Similarly for Comment, when user submit a reply, Userid and postid(that user is replying to) are inserted to Comment table, so that comment and user(sender) is related.

In the same table For second layer reply, the writer of reply and comment id are inserted when reply ,with parent_comment_id (which the user is replying to) are saved into comment database, so 2nd layer comment is related to a certain 1st layer comment.

4. Decision taken – React Client Main Components, and backend API

Structure of database/ why/how normalized the database, the tables relations?
Why designed the component UI like this/ come first related to the other page?

Frontend:

Use ReactDOM.createRoot for creating React root component; Render <RouterProvider> for router context, so router is guarded by RequireAuthRoute.js, ; Based on LocalStorage, if there is a user logged in then go to homepage, if not redirect to login ;
Navigate using <Navigate> component,;

API and backend:

Use Axios for requests using set baseURL; Use Express and to be run at port 3001;
Packages: CORS, body-parser,multer to upload picture files.
Database: user:root, password:admin. Use createPool to connect to database.
Router: have users, posts, channels, and comments for router

User info security:

Hashed password with SHA-256 to register and login to router to authenticate user.

Workflow:

a.Login

Normal user(students) need to log in to see any posts, and if without account, user needs to register following the link.
Admin can use the account Username: admin, Password: admin to log in and see the admin page to delete user, channel, post, replies.

b. Register

User provide 3 pieces of data,username, nickname,password, and submit to register, then user needs to log in with those credentials.

c.Main page(Channels of Posts)

Click lefttop corner to load all posts

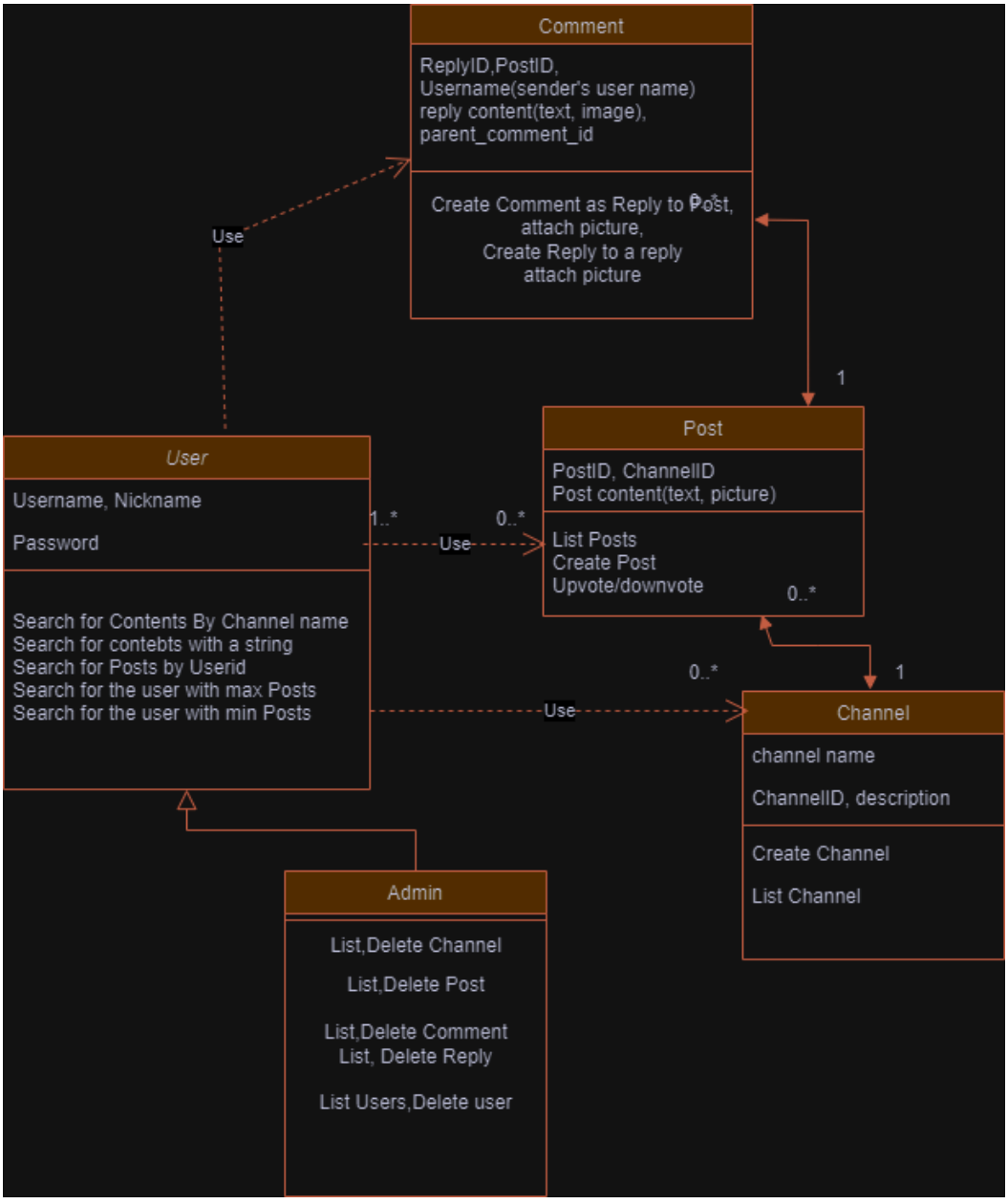
d.Post details page

Can upload pictures and text content

e.Admin page

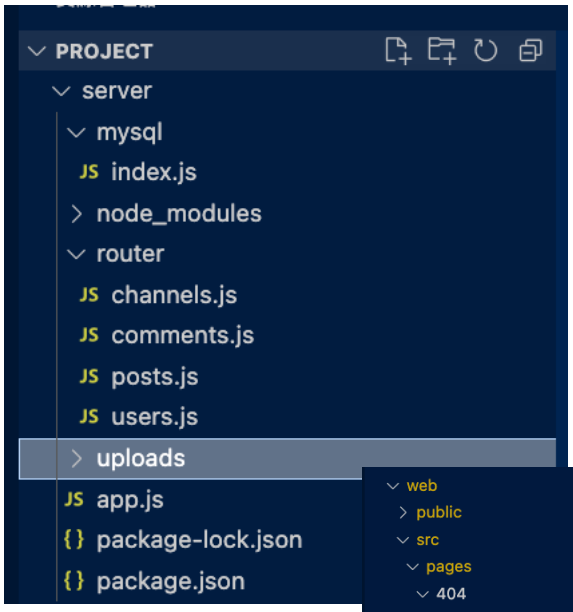
Admin can enter this page and admin pages only, (having no other functionability), after login, to delete either channels, posts or users.

Class diagram:



Directory structure:

Expand server/



The backup-project.zip has the whole project as original.
But In my submission to be clear,
I renamed server's package.json to package-server.json
and same for package-lock.json

Expand web/

In submission files changed the index.js names exact the same as their folder name,(there are 2 files I changed to some else names since it's repeated name as the red names in screenshot.), and having only one index.js in the project.

For initializing the database, need to execute script in

node_project.sql in mysql workbench