Design Report

NPM Packages used for the backend:

- express: for express framework
- mysql: for connecting to mysql database
- nodemon: for automatically restarting application
- body-parser: for parsing request json body
- cors: for Cross-origin resource sharing

NPM packages used for frontend:

- create-react-app: for react
- react-router-dom: for routing
- bootstrap: for ui styles
- react-icons: for thumbsup and thumbsdown buttons

First the backend is run then the frontend. At the root of the frontend there is the landing page which includes login. Users can either login or create a new account. After loging in the user is presented with a main page that has 3 panels, left panel is for channels, middle panel for questions and right panel for comments.

Users can create new channels, and after selecting that channel they can post new questions. After selecting a question from the list they can post comments for it. Users can also search for keywords in channels, questions and comments and search for users who posted questions and comments. They can also like or dislike comments. One user is created automatically by the backend (username: admin, password: admin). This users has the power to delete channels, questions and comments and the trash icons only appear for this user. Without loging in a user cannot access the main page of the program.

All the routes of the frontend are:

- / ---> the landing page including login
- /mainwindow ---> the 3 panel main window (cannot be accessed without login)
- /signup ---> the signup page

The database and its tables are created using MySql upon running server.js as the backend service.

The tables include:

- **Users** (user_id, username, password, image) ---> Used to store users, their credentials and profile images.
- **Channels** (channel_id, name, created_by) ---> Used to store channel names and the user that created it.
- **Questions** (question_id, question, image, channel_id, created_by, like_count, dislike_count) ---> used to store questions, screenshots and references the channel, the user that created it and the count of likes and dislikes.
- **Comments** (comment_id, comment, image, question_id, sub_comment_id, created_by, like_count, dislike_count) ---> Used to store comments, their screenshot, the question they belong to, the user who created it and the count of likes and dislikes for each comment.
- **CommentLikes** (comment_id, created_by, is_like) ---> used to store likes and dislikes for comments. Each row holds the id of the comment that was liked and the username of the user who liked it. The is_like field is a boolean that decides if the user has liked or disliked the comment. The primary key for each row is (comment_id, created_by). This way a user can either like or dislike a specific comment only once.
- **QuestionLikes** (question_id, created_by, is_like) ---> used to store likes for questions. Works the same way as CommentLikes table but this time for question likes and dislikes.

I implemented the channels, questions, comments, commentlikes and questionlikes to have the created_by column that is a foreign key referencing the username in users table.

The comments table includes a sub_comment_id that was meant to reference a comment row in the same table. So if we link comments together we would have :
(comment1 --> comment2 --> comment3 --> comment4 -->… --> null)

I couldn’t implement the nested comments in the frontend so the sub_comment_id field is null for all comments for now.