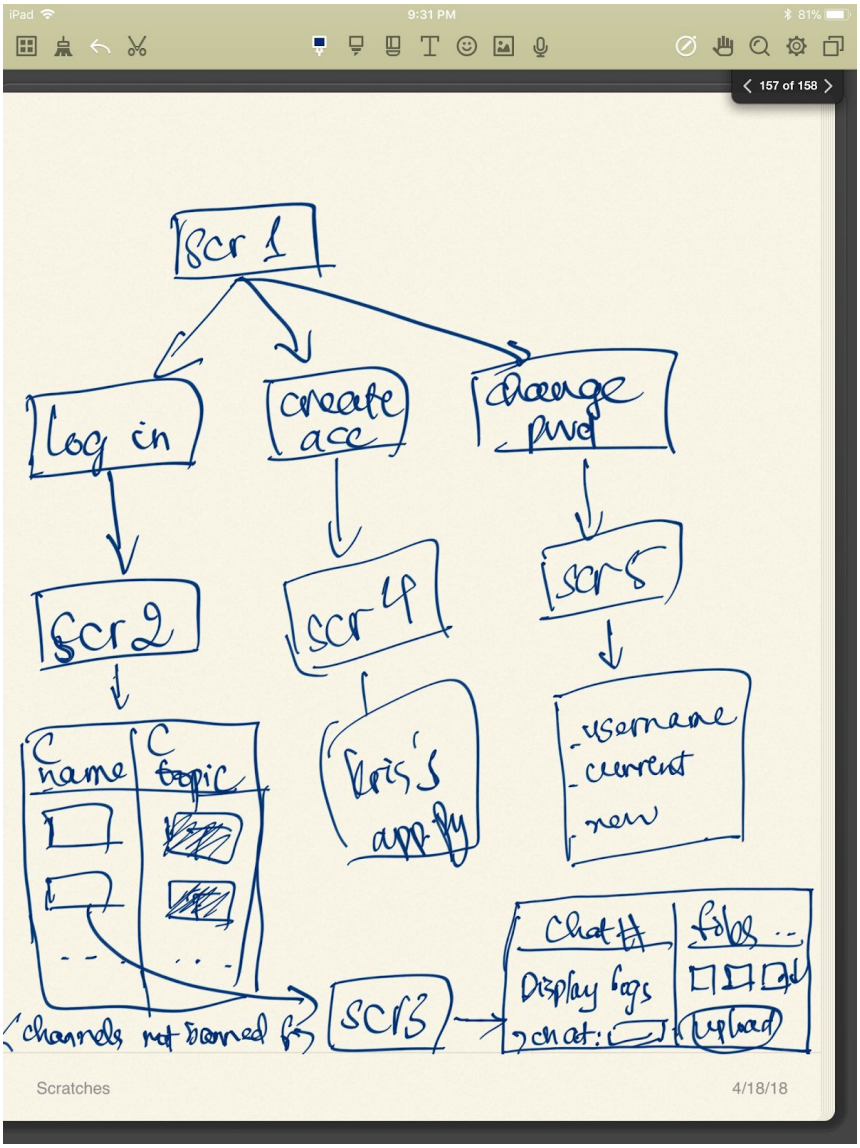
One of the major designing problems that we encountered while designing this project, is to figure out how to connect the three servers that we have on hand: file server, chat server, and web server, and that how to make all the servers share one database. At first we thought about sending all the information from the web server to the chat server, and let chat server do all the job of processing data and updating database, however we were concerned about problems that might occur while sending sensitive data over from the web server to the chat server, therefore we decided to make both web server and chat server to connect to and update the database. Although sending sensitive information within the same machine is in general considered as safe, however there might still be issues occur in the middle of transmission, for instance when user registered, the username and password are not successfully sent over to the chat server, which would cause problems and inconvenience for the user etc.. Considering various undesirable situations might happen, we decided to let web server to access and update the database directly upon almost all behaviors, including receiving user registration, authentication, creating channels, and file processing.

The second issue is about designing the database. How many tables should we have for the database, and also what information should we store for each user, channel, chats, and files? Initially we only have tables for 'user' and 'channels', and we were planning on putting all the chats and files corresponding to each channel in the 'channels' table. However this creates lots of inconvenience in fetching chatting data from the database, and after reconsideration we decided to add a 'chats' table and a 'files' table. The final structure of our database looks like below. All fields are either of type 'VARCHAR' or 'INTEGER', however the 'content' field for 'chats' is set to 'BLOB' for the convenience of storing encrypted data.

user	**channels**	**chats**	**fiiles**
id	id	id	id
username	channelname	channelname	filename
password	members	user_id	uploader
status	admins	content (BLOB)	channelname
channels	topics		
blocked	banned		
banned	filenames		
uploaded files			
channeladmin			

The third issue we had is about designing the interface for the web server so that the frontend and backend can collaborate better and also make it easier for us to implement. After consideration we have decided to put the chatting window, member list, and file processing window in the same page; join channel, admin display, and channel display on the same page; and finally, register, login on the same page. In our implementation, the channels which the current logged in user are admin of, will be specially marked with a star. The list of blocked users for each user is show whenever you click on the 'block' button on the page after user logged in. The flow of our interface is designed in a straightforward way such that all the information can be easily found and fetched for users. Following scratch shows our frontend designs:



To prevent SQL Injection attacks, we used prepared statement for every SQL query that involves user input. We also uses the 'escape' function from 'jinja.utils' to prevent cross-site scripting attacks, so that even if scripts are input by user, it will never be executed. All the message logs and user passwords are encrypted before storing into the database, and the file uploaded to the file server is also encrypted. Both uses symmetric encryption schemes, and the keys are stored secretly without sharing anyone. For channels, only admins can set the topics, ban users, or add another admin. In terms of file deletion, only uploaders have the privilege to delete the files. The change password feature is also implemented in a way that, user must enter the correct old password before they can change the password. This helps to prevent attackers from using the cookies of the currently logged in user and change the user's password.