**Chat App – Documentation**

Git Link: https://github.com/seanearls/3813ICT-Assignment1.git

**Git**

**Layout**

The chat app was stored in a folder named assignment1. In this folder I created a server folder for storing all routes, JSON data, the listen file (server initiation), and the server file (imports, and requirement paths).

Inside the src/app folder I stored all the component folders, model folder, and services folder.

**Version Control**

My method for version control was to commit after any successful changes/implementations to the app. I tried to commit and push any file in which I had implemented a new function or changes to a template.

I had sometimes forgotten to commit for a short while which resulted in me having to commit all files with a message detailing everything I could remember changing.

I was mostly successful with committing the changes. However, I should have been more careful as it had resulted in some changes being committed without a record of what was changed.

**Data Structures**

**Groups**

Groups are made up of 5 components ID, gName, users, assistants, and channel.

***ID –*** The ID of a group is represented as a number. This was used when routing to an instance of a group or group/channel.

***gName* –** The gName (group name) is a string that represents the name of the group to be displayed on the groups, groupAdmin, and chat components. This was also used for finding and deleting a group.

***Users –*** The users part of the group is an array of User data structures. This is used to store the user’s included in the group so that the groups page can display only the groups that a user is a part of.

***Assistants –*** The assistants is an array of user’s names that are an assistant of the group. This is used to allow group assistants to view the configure group button.

***Channel –*** Channel stores an array of channel data structures. These are the channels that can be accessed on the chat component.

**Users**

Users are made up of 3 components: username, email, and role.

***Username –*** Username is a string used for the user login. Usernames are used for comparisons to check whether a user is already registered, in a group, in a channel, deleting a user, and adding/removing a user from a channel or group.

***Email –*** The email is a string that is currently only used when registering a user.

***Role –*** A user’s role can be user, admin, or super. These are used for checking permissions of a user. For example, a normal user cannot create groups but an admin or super can.

**Channels**

A channel is made up of 3 components: ID, cName, and users.

***ID* –** ID is represented as a number. This is used for routing to a channel and is used as an index when creating a new channel.

***cName –*** The cName (channel name) is a string that represents the name of a channel. This is the name displayed on the group/channel lists. It is also used as an index when deleting a channel.

***Users –*** Users is an array of usernames. This is used for storing which users have access to a group.

**REST API**

***authRoute –*** The auth route is used for the user login. The route requests a username from the login component and compares is with the users.json file’s users’ usernames. If successful, the route sets user.valid to true then returns the user object. If the user object is returned the login component’s onLogin function checks user.valid. If user.valid is true and navigates to the groups component. If user.valid is false the login component will alert that the username is incorrect.

***usersRoute –*** The users route is used for sending all users to the component that it is called. When a getUsers function is called, the component subscribes to the data that is received (the list of users).

***newUserRoute –*** The new user route is used for creating a new user. It requests the username, email, and role that was input in the admin component, new user form. If the username is already in use the route will error and let the user know the username is already in use. If the username is not taken, the route will write the new user into the users.json file. The route sends the new user and a boolean, “registered” back to the component. Registered is used for checking whether the new user was registered successfully.

***editUserRoute –*** The editUserRoute is currently not functional but will be implemented in phase 2 of the assignment.

***deleteUserRoute –*** The delete user route is used for deleting a current user. The route requests the selected username from the admin component. An empty object is created to store the users.json data for storing then rewriting the users.json file. The route then reads the users.json file and matches the received username with the corresponding username in the file. This user is then deleted from the created object and this new data is then written into the users.json file.

***groupsRoute –*** The groups route is used for sending the groups.json data to a component. The route reads the groups.json file and stores it in an object variable, groups. Groups is then sent back to the requesting component so that it can be used.

***newGroupRoute –*** The new group route is creating a new group. The route requests the current admins and supers and stores in in a variable, users. A new group name is requested, and a new ID is requested which is the most recent group’s ID + 1. The route then reads the groups.json file and stores it in an empty object named newGroup. The requested information is then pushed into the newGroup object as a new object. The newGroup object is then stringified and written into the groups.json file. The route then sends the new group and a success checking Boolean (groupMade) back to the component to indicate wheter the group creating was successful or not.

***deleteGroupRoute –*** The delete group route is used for deleting a group. The route first requests the name of the group that is being deleted and creates a variable ‘toDelete’ for storing the updated group data. The route reads the groups.json file and stores it in the toDelete object. Next it loops through the groups in the toDelete object to find the group with the matching group name, this group is then deleted.

A new variable, ‘newData’ is then initiated storing a stringified ‘toDelete’ with the deleted group no longer included. The newData object is then written to the groups.json file. Finally, the route sends back the deleted group, and a Boolean, ‘deleted’ to indicate whether the deletion was a success.

***newChannelRoute –*** The new channel route is used for creating a new channel in a group. The route creates an empty newChannel object to store the the updated groups.json data with the new channel added to the group. The route requests the current group name (for indexing the group the channel will be added to), a new ID for the new channel, a new channel name, an empty object of messages (to be implemented in phase 2) and the current admins/supers to be stored as users in the group.

The route then reads the group.json file and stores the data in the newChannel object. Next it loops through the groups in the newChannel object’s data and finds the group which name matches the requested group name. The requested data is then pushed into the corresponding group in the newChannel object.

The newChannel object is then stringified and written into the groups.json file. Finally, the route sends back the new channel name and a Boolean, ‘channelMade’ to confirm whether the process was successful.

***deleteChannelRoute –*** The delete channel route removes a channel from a group. The route requests the name of the channel to be deleted and stores it in a variable called ‘deleted’, the name of the group the channel is in as ‘groupName’, and creates an empty object called toDelete for processing the new data.

The route reads the groups.json file and stores the data in the toDelete object.

Next the route loops through the groups and finds the one with the group name matching the requested group name. Then it loops through the channels to find which channel shares a name with the requested channel name. The corresponding channel is then deleted from the group.

The data is then stringified into a new variable called ‘newData’. The newData variable is then written into groups.json. Finally, the route sends back the channel, and a Boolean, ‘deleted’ to confirm the process was successful.

**Angular Architecture**

The angular chat app includes 6 components: app component, login component, admin component, group component, group admin component, and chat component.

Three models were used: group model, channel model, and user model.

The two services created were: group service and user service