Documentation for Software Frameworks Assignment Phase 1 (3813ICT)

Git

Git Layout

At the root of the Git repository is the node modules for the Angular app and its node modules. In the Angular app is the src folder containing the Angular components, HTML and CSS. The Angular app folder also contains the node.js server that handles HTTP requests.

Version Control Methodology

Throughout development, I tried my best to only commit working code to ensure that the app compiles if a revert to a previous version. This was particularly with main, as I would work on main directly when making small changes but I would use branches for more complex tasks. The branches allow testing of new functionality without affecting main, which would then be merged into main when they are completed. This allows main to be always be capable of building when pulled from and is best practice when working with a team.

Data Structures

The main data structures for this Angular project are:

- Users
- Groups
- Channels
- Permissions

Users is an array containing 0 or more user objects. Each user has the following attributes:

- `username`: string
- `email`: string
- `id`: number
- `role`: string

A User can have the roles: default, super admin, group admin, group assistant.

Groups is an array containing 0 or more group objects. Each group has the following attributes:

- `id`: number
- `groupName`: string

Channels is an array containing 0 or more group objects. Each channel has the following attributes:

- `id`: number
- `channelName`: string

Users, Groups and Channels are stored in /server/data as serialised JSON files

Permissions is an array of permission objects. Each permission object contains a `groupId` (a reference to the `id` of a group object) and an array of members that will have the permission level. Each member will have a `userId` (a reference to an id of a User) and an array of `channelIds` (a reference to an id of a Channel), which are the channels the user is subscribed to.

```
{groupId: number, members: [
    {userId: number, channelIds: [number]
]}
```

REST API

For the following routes, the parameters and return values are as stated in previously in the "data structures" section, unless explicitly stated

Auth route

Route overview	**This route checks if a user exists in a predefined JSON. If the user exists, it will call a login function to log the user in; returning an error if they do not.**
Route path	/auth
Method	POST
Parameter/s	`username`
Return value/s	if (auth): `{valid: true, user: {username, email, id, role}}` if !(auth): `{valid: false, errors:{}}`
Description	This route takes a `username` in the http request. If there is a match in the users.json file then the route returns `valid: true` and the user information. If there is no match, valid: false is returned.

Set permissions route

Route overview	**This route gives app permissions to a user depending on the value of user.role**
Route path	/setPermissions
Method	POST
Parameter/s	`user: {username, email, id, role}`
Return value/s	***See below code block for proper formatting
Description	This route takes a `user` and returns the `permissions` object where default users have no permissions. Relevant permissions are returned according to the user `roles`.

```
{
    rolePermissions: {
        // super admin
        createUser: boolean,
        removeUser: boolean,
        upgradeUserPerm: boolean,
        // group admin
        createGroup: boolean,
        createChannel: boolean,
        createUserInChannel: boolean,
        inviteUserToChannel: boolean,
        deleteGroup: boolean,
        deleteChannel: boolean,
        removeUserFromChannel: boolean,
        upgradeToGroupAssist: boolean,
        // group assist
        addUserToChannel: boolean,
        removeUserToChannel: boolean,
        createChannelInGroup: boolean
    }
}
```

Create User

Route overview	**Creates a new user if the username doesn't exist**
Route path	/user/create
Method	POST

Route overview	**Creates a new user if the username doesn't exist**
Parameter/s	`{user: {username, email, id: undefined, role}}`
Return value/s	No user match: `users` user match: `error`
Description	The route reads `users.json`, and if there is no `user` match then it will add the `user` to `users` and assign a `userId`. It writes `users` to `users.json` and returns `users`

Update User

Route overview	**Updates a User**
Route path	/user/:id
Method	POST
Parameter/s	Request parameters: `id` Request body: `{user: {username, email, id, role}}`
Return value/s	`users`
Description	A `userID` and a `user` object is received and `users.json` is read. It matches the `userID` and `id` of the `user` object and replaces the values with `user`. `users` is written to users.json and the route responds with this return value.